dsrep

January 8, 2022

1 Data Science Research Methods Report-2

Note: Some processed Dataframes have been stored as pickle files which will be loaded later on so the rprocessing code need not be run again and again.

2 Extra Libraries to Install

1. Ipython
2. Tabula

2.1 Introduction

The PAMAP2 Physical Activity Monitoring dataset (available here) contains data from 9 participants who participated in 18 various physical activities (such as walking, cycling, and soccer) while wearing three inertial measurement units (IMUs) and a heart rate monitor. This information is saved in separate text files for each subject. The goal is to build hardware and/or software that can determine the amount and type of physical activity performed by an individual by using insights derived from analysing the given dataset.

[1]:
```
import os
import random
from collections import defaultdict
```

[2]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
import statsmodels.api as sm
import tabula
from IPython.display import display
from matplotlib import rcParams
from numpy.fft import rfft
from scipy.stats import ranksums, ttest_ind
from sklearn import cluster, preprocessing
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.metrics import (accuracy_score, classification_report, f1_score,
                             log_loss, v_measure_score)
```

[3]:
```
os.chdir("/home/sahil/Downloads/PAMAP2_Dataset/")  # Setting up working directory
import warnings
```

[46]:
```
pd.set_option("max_columns", None)
pd.set_option("max_rows", None)
```

[5]:
```
warnings.filterwarnings("ignore")
```

2.2 Data Cleaning

For tidying up the data : - The data of various subjects is loaded and given relevant column names for various features. - The data for all subjects are then stacked together to form one table. - 'Orientation' columns are removed because it was mentioned in the data report that it is invalid in this data collection. - The accelerometer of sensitivity 6g is also removed as it's not completely accurate for all activities due to its low sensitivity. - Similarly, the rows with Activity ID "0" are also removed as it does not relate to any specific activity. - The missing values are filled up using the forward fill method.In this method, the blank values are filled with the value occuring just before it. - Added a new feature, 'BMI' or Body Mass Index for the 'subject_detail' table - Additional feature, 'Activity Type' is added to the data which classifies activities into 3 classes, 'Light' activity,'Moderate' activity and 'Intense' activity. 1. Lying,sitting,ironing and standing are labelled as 'light' activities. 2. Vacuum cleaning,descending stairs,normal walking,Nordic walking and cycling are considered as 'Moderate' activities 3. Ascending stairs,running and rope jumping are labelled as 'Intense' activities. This classification makes it easier to perform hypothesis testing between pair of attributes. - Subject 109 is not considered for analysis as it has performed few protocol activities. - Optional activities are ignored since very few subjects have performed them which makes it difficult to model.

Given below are functions to give relevant names to the columns and create a single table containing data for all subjects

[91]:
```
def gen_activity_names():
    # Using this function all the activity names are mapped to their ids
    act_name = {}
    act_name[0] = "transient"
    act_name[1] = "lying"
    act_name[2] = "sitting"
    act_name[3] = "standing"
    act_name[4] = "walking"
    act_name[5] = "running"
    act_name[6] = "cycling"
    act_name[7] = "Nordic_walking"
    act_name[9] = "watching_TV"
    act_name[10] = "computer_work"
    act_name[11] = "car driving"
```

```
    act_name[12] = "ascending_stairs"
    act_name[13] = "descending_stairs"
    act_name[16] = "vacuum_cleaning"
    act_name[17] = "ironing"
    act_name[18] = "folding_laundry"
    act_name[19] = "house_cleaning"
    act_name[20] = "playing_soccer"
    act_name[24] = "rope_jumping"
    return act_name
```

[92]:
```
def generate_three_IMU(name):
    # Adding coordinate suffix for accelerometer
    x = name + "_x"
    y = name + "_y"
    z = name + "_z"
    return [x, y, z]
```

[93]:
```
def generate_four_IMU(name):
    # Adding coordinates suffixes for orientation
    x = name + "_x"
    y = name + "_y"
    z = name + "_z"
    w = name + "_w"
    return [x, y, z, w]
```

[94]:
```
def generate_cols_IMU(name):
    # temperature column names
    temp = name + "_temperature"
    output = [temp]
    # acceleration 16g columns names
    acceleration16 = name + "_3D_acceleration_16"
    acceleration16 = generate_three_IMU(acceleration16)
    output.extend(acceleration16)
    # acceleration 6g column anmes
    acceleration6 = name + "_3D_acceleration_6"
    acceleration6 = generate_three_IMU(acceleration6)
    output.extend(acceleration6)
    # gyroscope column names
    gyroscope = name + "_3D_gyroscope"
    gyroscope = generate_three_IMU(gyroscope)
    output.extend(gyroscope)
    # magnometer column names
    magnometer = name + "_3D_magnetometer"
    magnometer = generate_three_IMU(magnometer)
    output.extend(magnometer)
    # oreintation column names
    oreintation = name + "_4D_orientation"
```

```
    oreintation = generate_four_IMU(oreintation)
    output.extend(oreintation)
    return output
```

[95]:
```
def load_IMU():
    # Function for generating final column names
    output = ["time_stamp", "activity_id", "heart_rate"]
    hand = "hand"
    hand = generate_cols_IMU(hand)
    output.extend(hand)
    chest = "chest"
    chest = generate_cols_IMU(chest)
    output.extend(chest)
    ankle = "ankle"
    ankle = generate_cols_IMU(ankle)
    output.extend(ankle)
    return output
```

[ ]:
```
def load_subjects(
    root1="/home/sahil/Downloads/PAMAP2_Dataset/Protocol/subject",
    root2="/home/sahil/Downloads/PAMAP2_Dataset/Optional/subject",
):
    # This function loads data from subject files and names the columns
    cols = load_IMU()
    output = pd.DataFrame()
    for i in range(101, 110):
        path1 = root1 + str(i) + ".dat"
        subject = pd.DataFrame()
        subject_prot = pd.read_table(path1, header=None, sep="\s+")  # subject␣
↪data from
        # protocol activities
        subject = subject.append(subject_prot)
        subject.columns = cols
        subject = subject.sort_values(
            by="time_stamp"
        )  # Arranging all measurements according to
        # time
        subject["id"] = i
        output = output.append(subject, ignore_index=True)
    return output
```

Feel free to skip the execution of this cell as the output is saved and reloaded in the cells below it.This is done because the eprocessing takes a lot of time and so it was found more appropriate to save the output so re-running the cell is not required

[ ]:
```
data = load_subjects()  # Add your own location for the data here to replicate␣
↪the code
```

```
# for eg data = load_subjects('filepath')
data = data.drop(
    data[data["activity_id"] == 0].index
)  # Removing rows with activity id of 0
act = gen_activity_names()
data["activity_name"] = data.activity_id.apply(lambda x: act[x])
data = data.drop(
    [i for i in data.columns if "orientation" in i], axis=1
)  # Dropping Orientation  columns
cols_6g = [i for i in data.columns if "_6_" in i]  # 6g acceleration data columns
data = data.drop(cols_6g, axis=1)  # dropping 6g acceleration columns
display(data.head())
# Saving transformed data in pickle format becuse it has the fastest read time␣
 ↪compared
# to all other formats
data.to_pickle("activity_data.pkl")  # Saving transformed data for future use
```

[97]:
```
def train_test_split_by_subjects(data):  # splitting by subjects
    subjects = [
        i for i in range(101, 109)
    ]  # Eliminate subject 109  due to less activities
    train_subjects = [101, 103, 104, 105]
    test_subjects = [i for i in subjects if i not in train_subjects]
    train = data[data.id.isin(train_subjects)]  # Generating training data
    test = data[data.id.isin(test_subjects)]  # generating testing data
    return train, test
```

[98]:
```
def split_by_activities(data):
    light = ["lying", "sitting", "standing", "ironing"]
    moderate = [
        "vacuum_cleaning",
        "descending_stairs",
        "normal_walking",
        "nordic_walking",
        "cycling",
    ]
    intense = ["ascending_stairs", "running", "rope_jumping"]
    def split(activity):  #  method for returning activity labels for activities
        if activity in light:
            return "light"
        elif activity in moderate:
            return "moderate"
        else:
            return "intense"
    data["activity_type"] = data.activity_name.apply(lambda x: split(x))
    return data
```

Loading data and doing the train-test split for EDA and Hypothesis testing.

[99]:
```
data = pd.read_pickle("activity_data.pkl")
data = split_by_activities(data)
train, test = train_test_split_by_subjects(
    data
)  # train and test data for EDA and hypothesis testing respectively.
subj_det = tabula.read_pdf(
    "subjectInformation.pdf", pages=1
)  # loading subject detail table from pdf file.
# Eliminating unnecessary columns and fixing the column alignment of the table.
sd = subj_det[0]
new_cols = list(sd.columns)[1:9]
sd = sd[sd.columns[0:8]]
sd.columns = new_cols
subj_det = sd
```

Create clean data for use in modelling

[100]:
```
eliminate = [
    "activity_id",
    "activity_name",
    "time_stamp",
    "id",
]  # Columns not meant to be cleaned
features = [i for i in data.columns if i not in eliminate]
clean_data = data
clean_data[features] = clean_data[features].ffill() # Code for forward fill
display(clean_data.head())
```

```
      time_stamp  activity_id  heart_rate  hand_temperature  \
2928       37.66            1         NaN            30.375
2929       37.67            1         NaN            30.375
2930       37.68            1         NaN            30.375
2931       37.69            1         NaN            30.375
2932       37.70            1       100.0            30.375

      hand_3D_acceleration_16_x  hand_3D_acceleration_16_y  \
2928                    2.21530                    8.27915
2929                    2.29196                    7.67288
2930                    2.29090                    7.14240
2931                    2.21800                    7.14365
2932                    2.30106                    7.25857

      hand_3D_acceleration_16_z  hand_3D_gyroscope_x  hand_3D_gyroscope_y  \
2928                    5.58753            -0.004750             0.037579
2929                    5.74467            -0.171710             0.025479
2930                    5.82342            -0.238241             0.011214
```

```
2931                   5.89930           -0.192912             0.019053
2932                   6.09259           -0.069961            -0.018328

      hand_3D_gyroscope_z  ...  ankle_3D_acceleration_16_z  \
2928            -0.011145  ...                    0.095156
2929            -0.009538  ...                   -0.020804
2930             0.000831  ...                   -0.059173
2931             0.013374  ...                    0.094385
2932             0.004582  ...                    0.095775

      ankle_3D_gyroscope_x  ankle_3D_gyroscope_y  ankle_3D_gyroscope_z  \
2928              0.002908             -0.027714              0.001752
2929              0.020882              0.000945              0.006007
2930             -0.035392             -0.052422             -0.004882
2931             -0.032514             -0.018844              0.026950
2932              0.001351             -0.048878             -0.006328

      ankle_3D_magnetometer_x  ankle_3D_magnetometer_y  \
2928                 -61.1081                 -36.8636
2929                 -60.8916                 -36.3197
2930                 -60.3407                 -35.7842
2931                 -60.7646                 -37.1028
2932                 -60.2040                 -37.1225

      ankle_3D_magnetometer_z   id activity_name activity_type
2928                 -58.3696  101         lying         light
2929                 -58.3656  101         lying         light
2930                 -58.6119  101         lying         light
2931                 -57.8799  101         lying         light
2932                 -57.8847  101         lying         light

[5 rows x 36 columns]
```

After using the Forward Fill method, the first four values of heart rate are still missing. So the first four rows are dropped

[101]:
```
clean_data = clean_data.dropna()
display(clean_data.head())
```

```
      time_stamp  activity_id  heart_rate  hand_temperature  \
2932       37.70            1       100.0            30.375
2933       37.71            1       100.0            30.375
2934       37.72            1       100.0            30.375
2935       37.73            1       100.0            30.375
2936       37.74            1       100.0            30.375

      hand_3D_acceleration_16_x  hand_3D_acceleration_16_y  \
```

```
2932                   2.30106                    7.25857
2933                   2.07165                    7.25965
2934                   2.41148                    7.59780
2935                   2.32815                    7.63431
2936                   2.25096                    7.78598

      hand_3D_acceleration_16_z  hand_3D_gyroscope_x  hand_3D_gyroscope_y  \
2932                   6.09259            -0.069961            -0.018328
2933                   6.01218             0.063895             0.007175
2934                   5.93915             0.190837             0.003116
2935                   5.70686             0.200328            -0.009266
2936                   5.62821             0.204098            -0.068256

      hand_3D_gyroscope_z  ...  ankle_3D_acceleration_16_z  \
2932             0.004582  ...                    0.095775
2933             0.024701  ...                   -0.098161
2934             0.038762  ...                   -0.098862
2935             0.068567  ...                   -0.136998
2936             0.050000  ...                    0.133911

      ankle_3D_gyroscope_x  ankle_3D_gyroscope_y  ankle_3D_gyroscope_z  \
2932              0.001351             -0.048878             -0.006328
2933              0.003793             -0.026906              0.004125
2934              0.036814             -0.032277             -0.006866
2935             -0.010352             -0.016621              0.006548
2936              0.039346              0.020393             -0.011880

      ankle_3D_magnetometer_x  ankle_3D_magnetometer_y  \
2932                 -60.2040                 -37.1225
2933                 -61.3257                 -36.9744
2934                 -61.5520                 -36.9632
2935                 -61.5738                 -36.1724
2936                 -61.7741                 -37.1744

      ankle_3D_magnetometer_z   id  activity_name  activity_type
2932                 -57.8847  101          lying          light
2933                 -57.7501  101          lying          light
2934                 -57.9957  101          lying          light
2935                 -59.3487  101          lying          light
2936                 -58.1199  101          lying          light

[5 rows x 36 columns]
```

Finally, save the clean data for future use in model prediction

[102]:
```python
clean_data.to_pickle("clean_act_data.pkl")
```

2.3 Exploratory Data Analysis

After labelling the data appropriately, 4 subjects are selected for training set.Subjects 101, 103, 104, 105 are selected for training set adn rest for training set. 4 subjects for testing set such that the training and testing set have approximately equal size. In the training set, we perform Exploratory Data Analysis and come up with potential hypotheses. We then test those hypotheses on the testing set. 50% of data is used for training in this case(Exploratory data analysis) and the rest for testing.

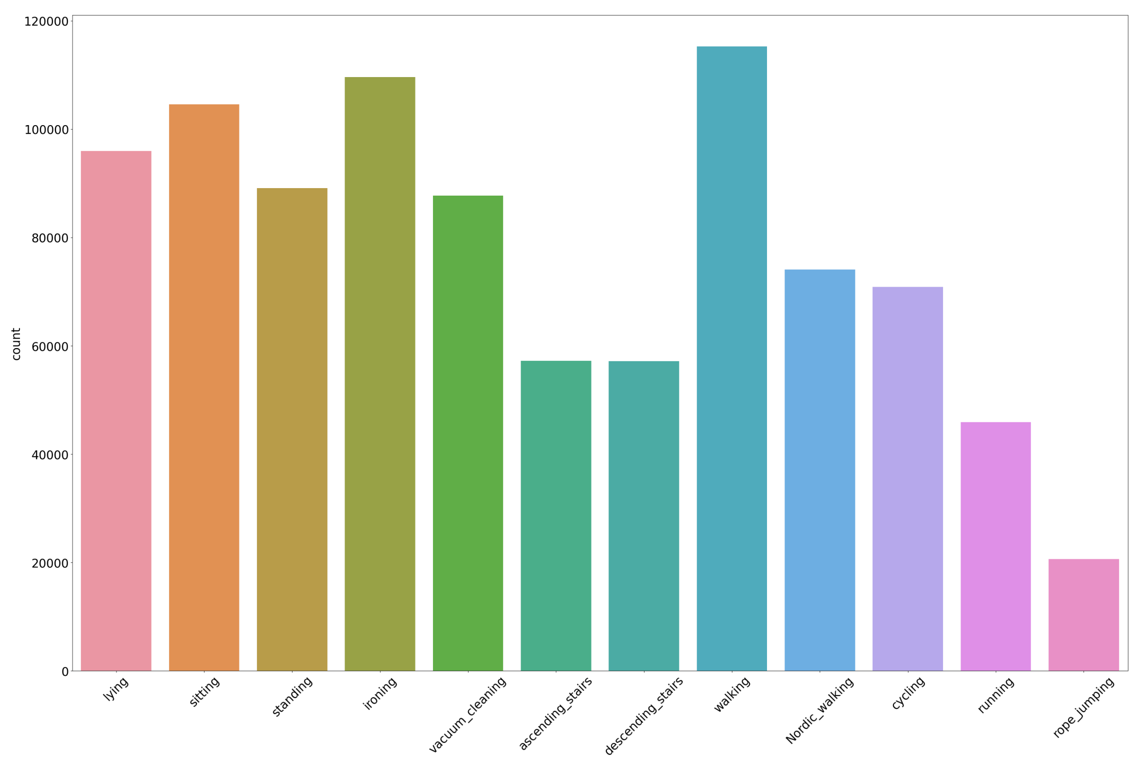
Calculating BMI of the subjects

[103]:
```
height_in_metres = subj_det["Height (cm)"] / 100 # Calculating Height in metres
weight_in_kg = subj_det["Weight (kg)"]
subj_det["BMI"] = weight_in_kg / (height_in_metres) ** 2
```

2.3.1 Data Visualizations

- Bar chart for frequency of activities.

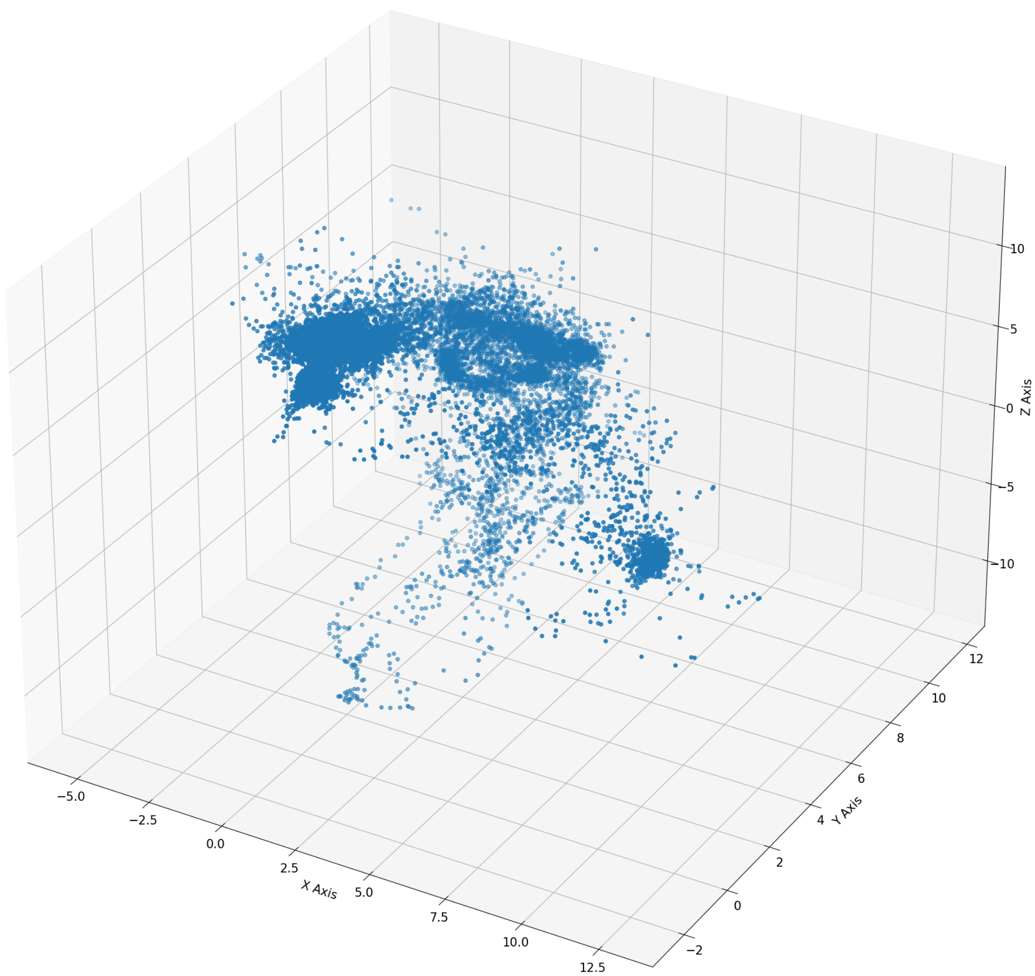
[105]:
```
rcParams["figure.figsize"] = 40, 25 # setting the figure dimensions
rcParams["font.size"] =  25 # Setting font size

ax = sns.countplot(x="activity_name", data=train)
ax.set_xticklabels(ax.get_xticklabels(), rotation=45)  # Rotating Text
plt.show()
```



- 3D scatter plot of chest acceleration coordinates for lying It is expected that vertical chest acceleration will be more while lying due to the movements involved and an attempt is made to check this visually over here.

[107]:
```
plt.clf()
rcParams["font.size"] =  15

train_running = train[train.activity_name == "lying"] # Extracting rows with␣
↪activity labelled as lying
fig = plt.figure()
ax = fig.add_subplot(projection="3d")
x = train_running["chest_3D_acceleration_16_x"]
y = train_running["chest_3D_acceleration_16_y"]
z = train_running["chest_3D_acceleration_16_z"]
ax.scatter(x, y, z)
ax.set_xlabel("X Axis")
ax.set_ylabel("Y Axis")
ax.set_zlabel("Z Axis")
plt.show()
```

```
<Figure size 4000x2500 with 0 Axes>
```



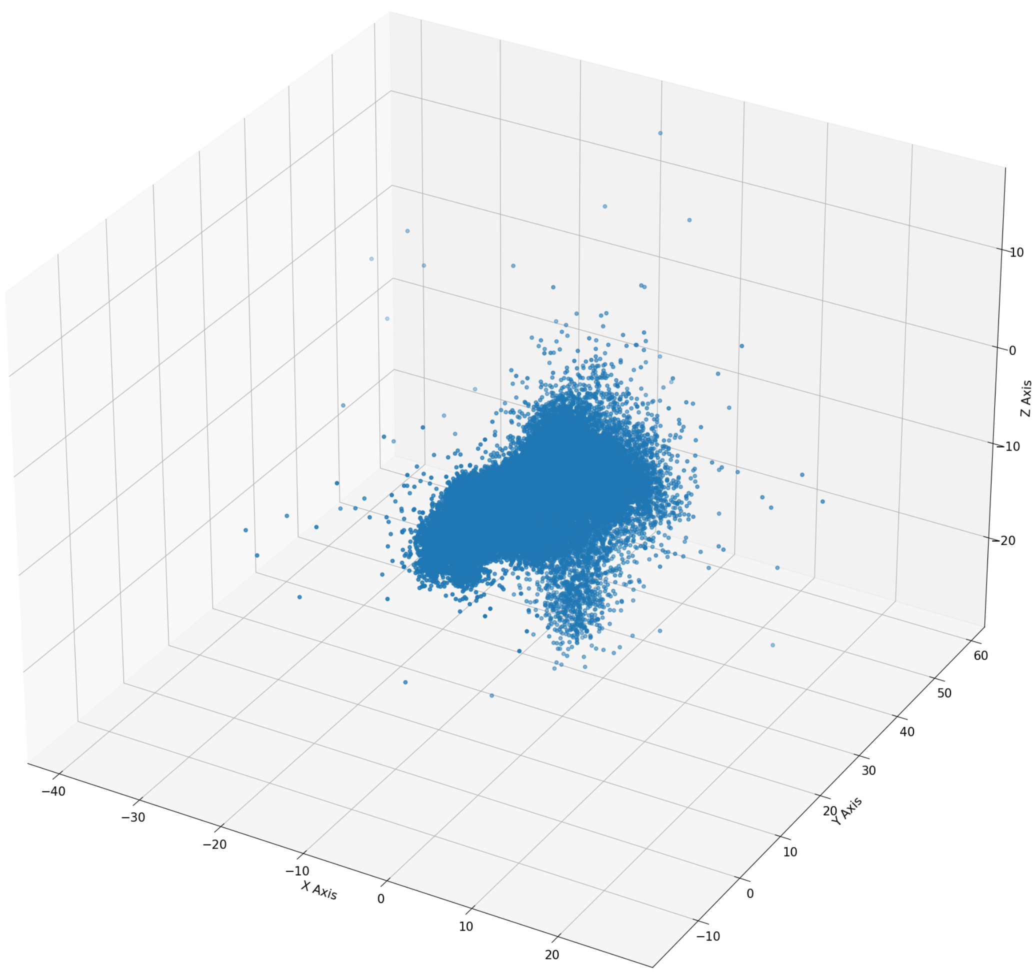
As we see, there seems to be more variance along the z axis(vertical direction) than the x and y axis.

- 3D scatter plot of chest acceleration coordinates for running Since running involves mostly horizontal movements for the chest, we expect most of chest acceleration data to lie on the horizontal x amd y axis.

[108]:
```
plt.clf()
train_running = train[train.activity_name == "running"] # Extracting rows  with␣
↪activity labeleed as running
fig = plt.figure()
ax = fig.add_subplot(projection="3d")
x = train_running["chest_3D_acceleration_16_x"]
y = train_running["chest_3D_acceleration_16_y"]
```

```
z = train_running["chest_3D_acceleration_16_z"]
ax.scatter(x, y, z)
ax.set_xlabel("X Axis")
ax.set_ylabel("Y Axis")
ax.set_zlabel("Z Axis")
plt.show()
```

```
<Figure size 4000x2500 with 0 Axes>
```

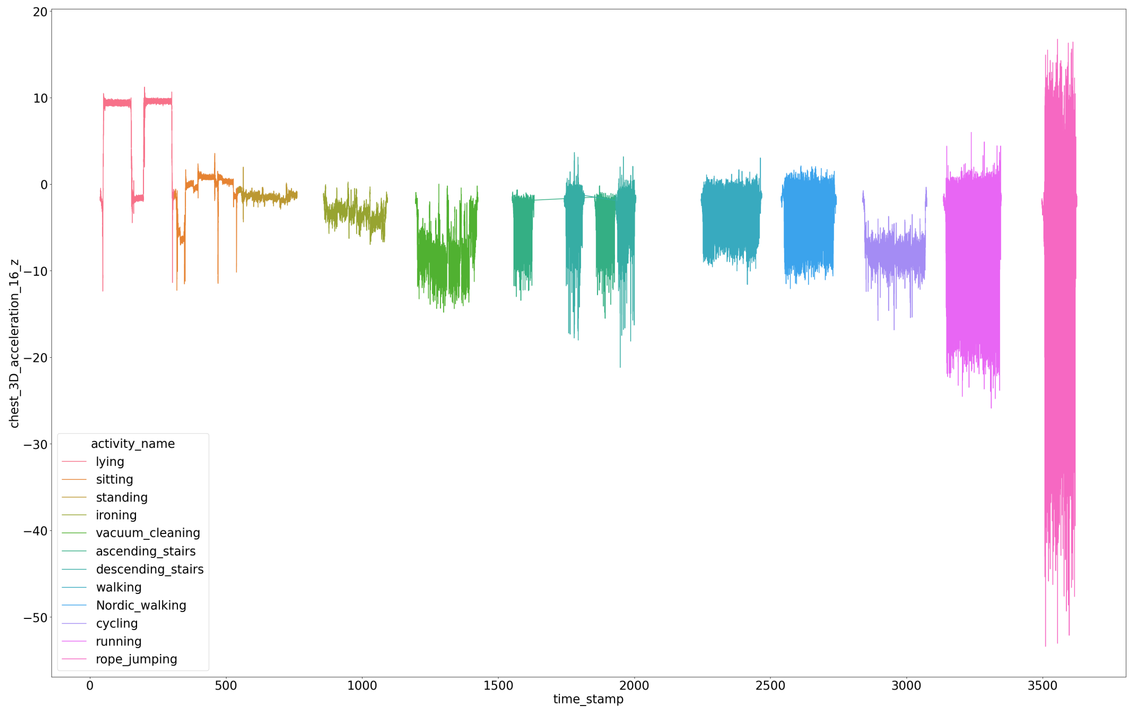


As we expected, for running, most of the points lie along the x and y axis.

- Time series plot of z axis chest acceleration

Subject 101 is considered for this time series plot as it does all protocol activities

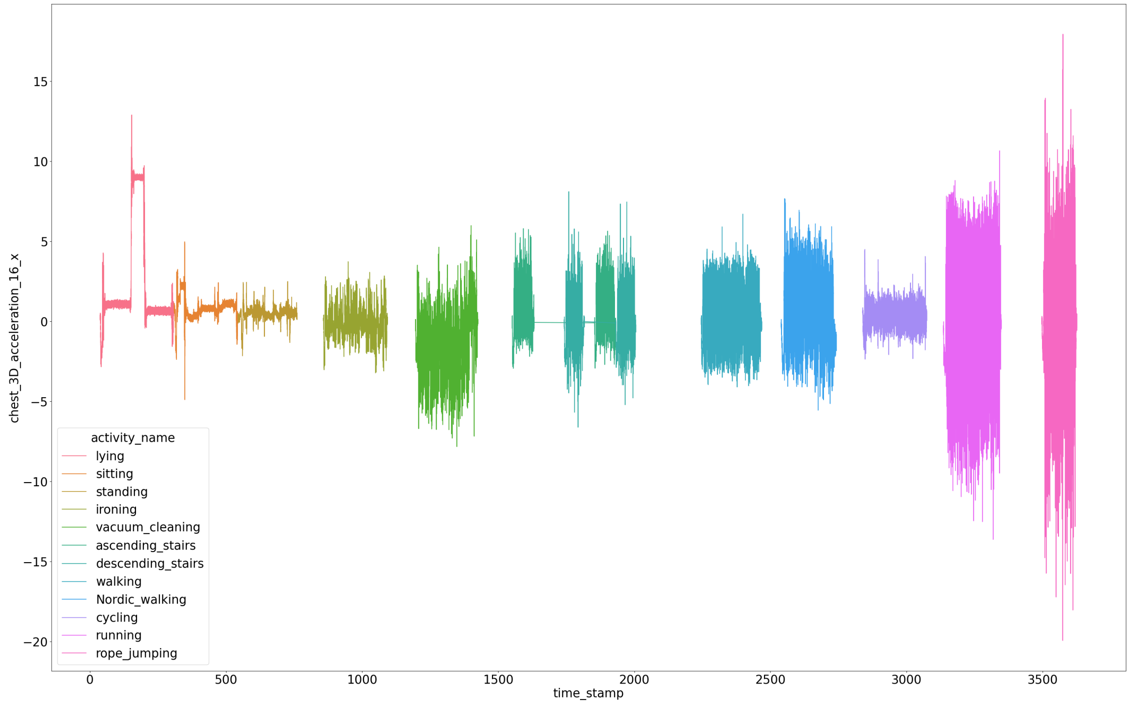
[110]:
```
plt.clf()
rcParams["font.size"] =  25

random.seed(4)
train1 = train[train.id == 101]
sns.lineplot(
    x="time_stamp", y="chest_3D_acceleration_16_z", hue="activity_name",␣
 ↪data=train1
) # Generating timeplot and grouping it with activity name
plt.show()
```



It look like the vertical chest acceleration during lying is higher compared to other activities.Also there seems to be a lot of variance for this feature while the subject is running.

- Time series plot of x axis chest acceleration

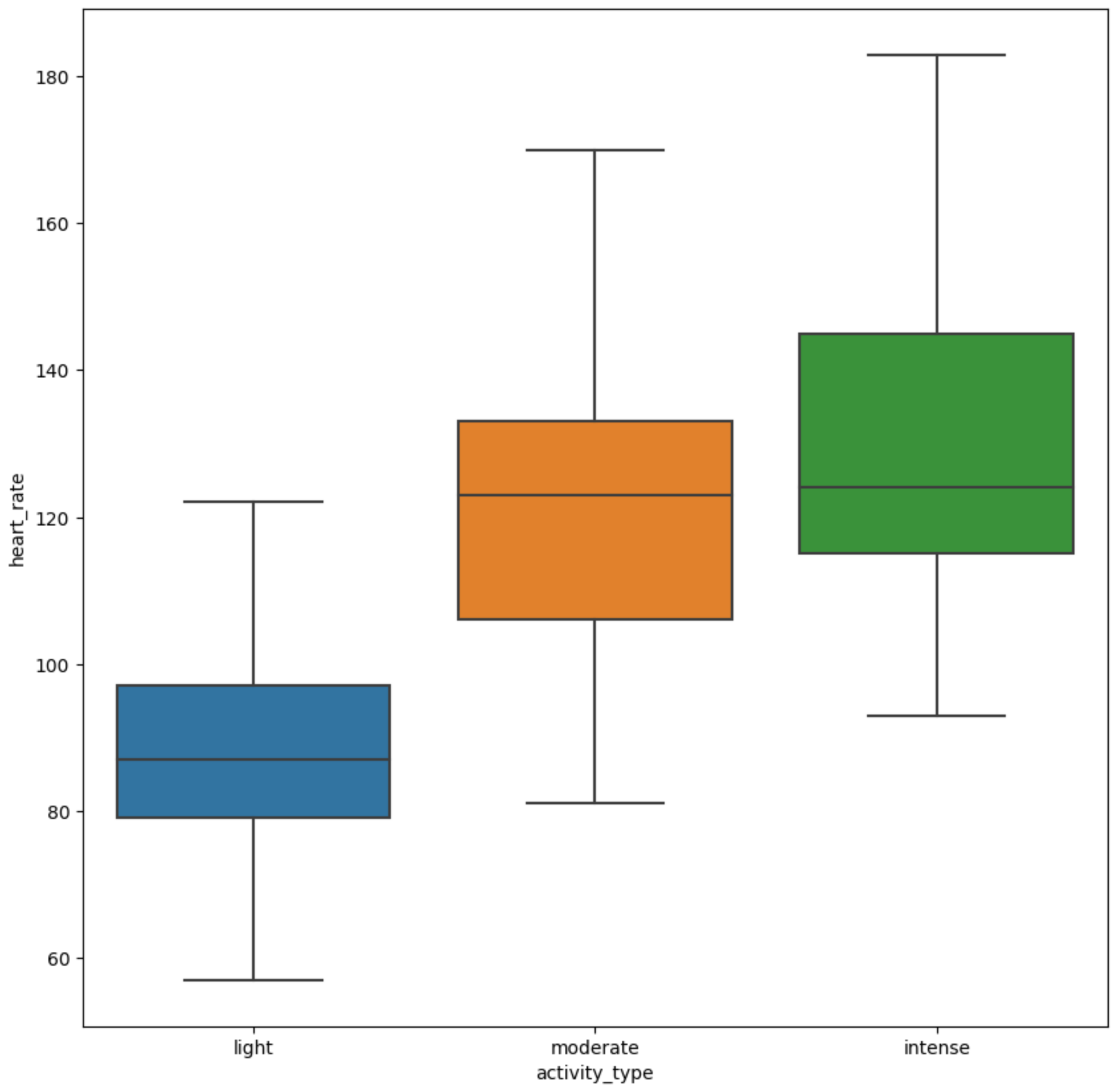
[112]:
```
plt.clf()
random.seed(4)
train1 = train[train.id == 101]
sns.lineplot(
    x="time_stamp", y="chest_3D_acceleration_16_x", hue="activity_name",␣
 ↪data=train1
)
plt.show()
```



As expected the variance is higher for activities that require horizontal movement through space

- Boxplot of heart rate grouped by activity type.

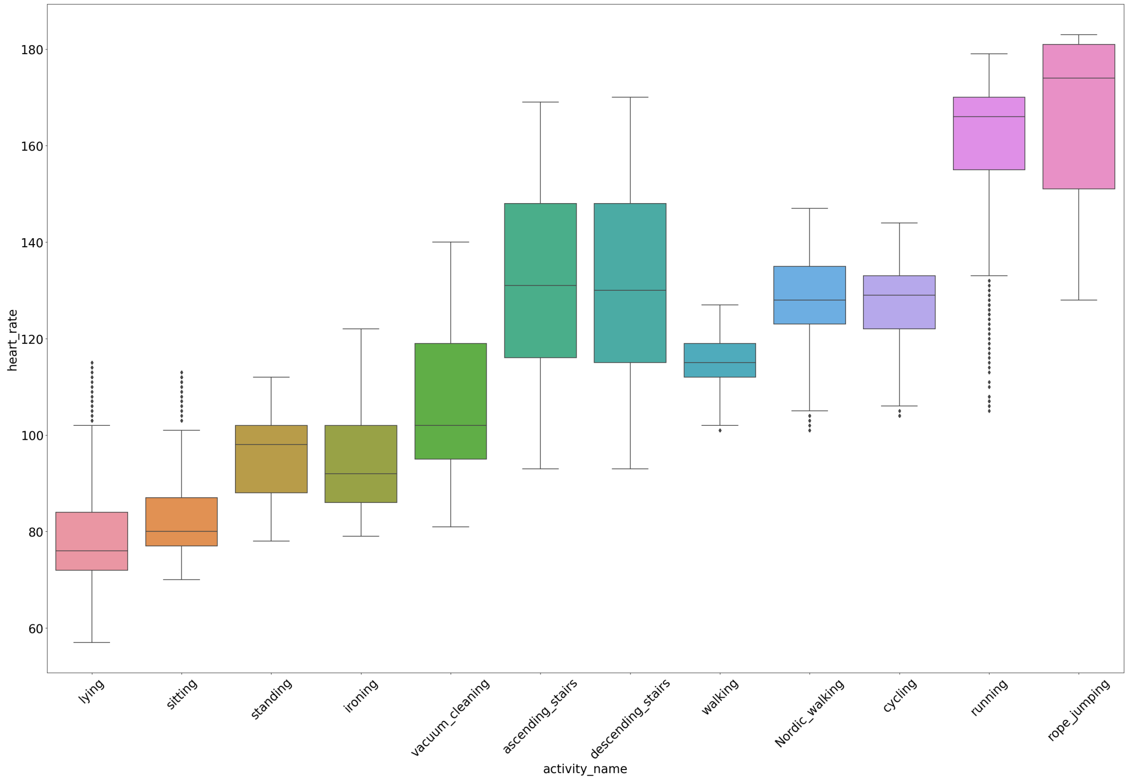
[120]:
```
rcParams["figure.figsize"] = 10, 10
rcParams["font.size"] =  10

ax = sns.boxplot(x="activity_type", y="heart_rate", data=train)
ax.set_xticklabels(ax.get_xticklabels(), rotation=0)  # Rotating Text
plt.show()
```



1. It is observed that moderate and intense activities have higher heart rate than light activities as expected.
2. There doesn't seem to be much seperation between heart rate of moderate and intesne activity.

- Boxplot of heart rate grouped by activity.

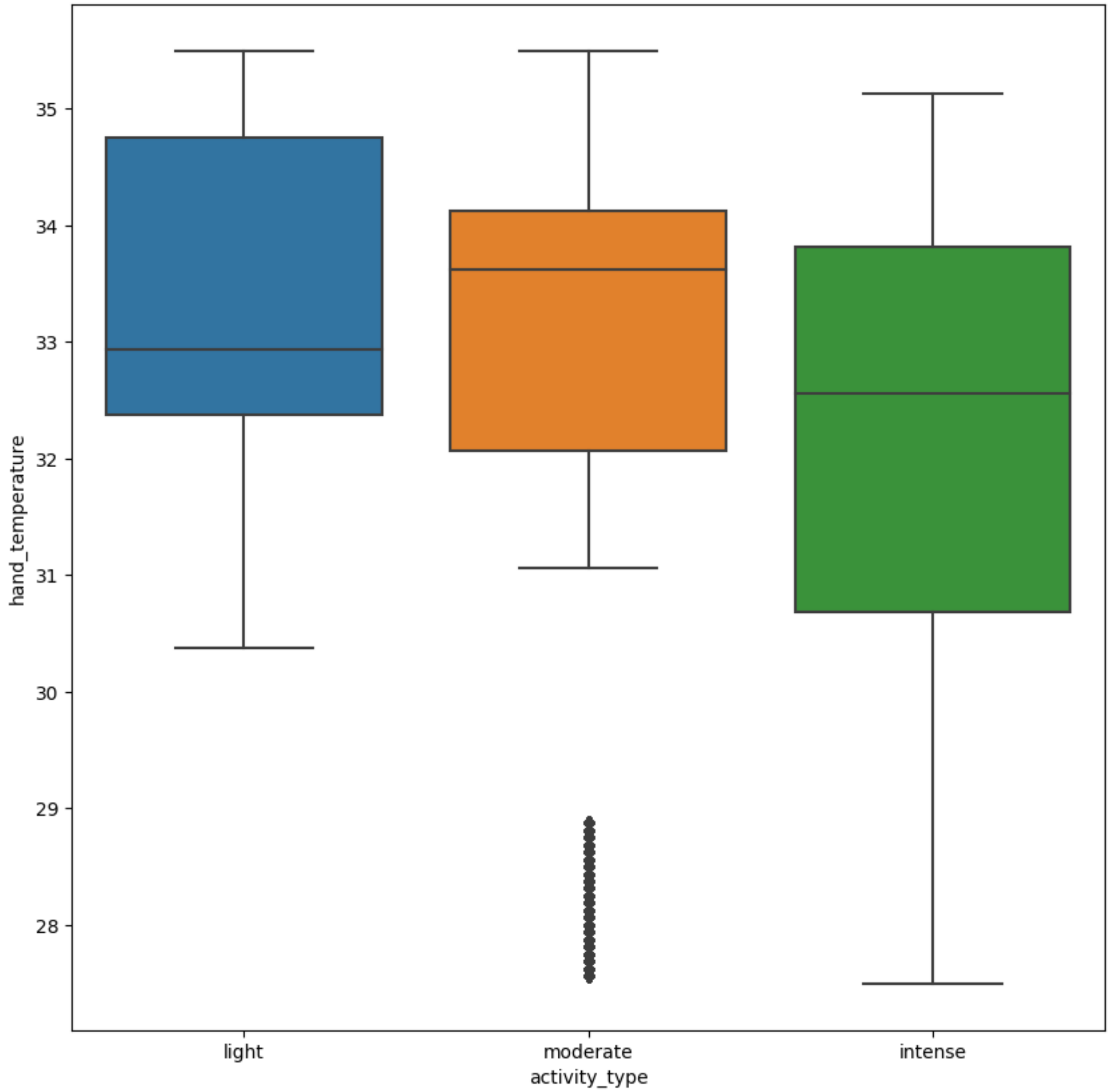
[114]:
```
rcParams["figure.figsize"] = 40, 25
ax = sns.boxplot(x="activity_name", y="heart_rate", data=train)
ax.set_xticklabels(ax.get_xticklabels(), rotation=45)  # Rotating Text
plt.show()
```



1. Most of the activities have a skewed distribution for heart rate.
2. 'Nordic_walking','running' and 'cycling' have a lot of outliers on the lower side.
3. Activities like 'lying','sitting' and 'standing' have a lot of outliers on the upper side.

- Boxplot of hand temperature grouped by activity type.

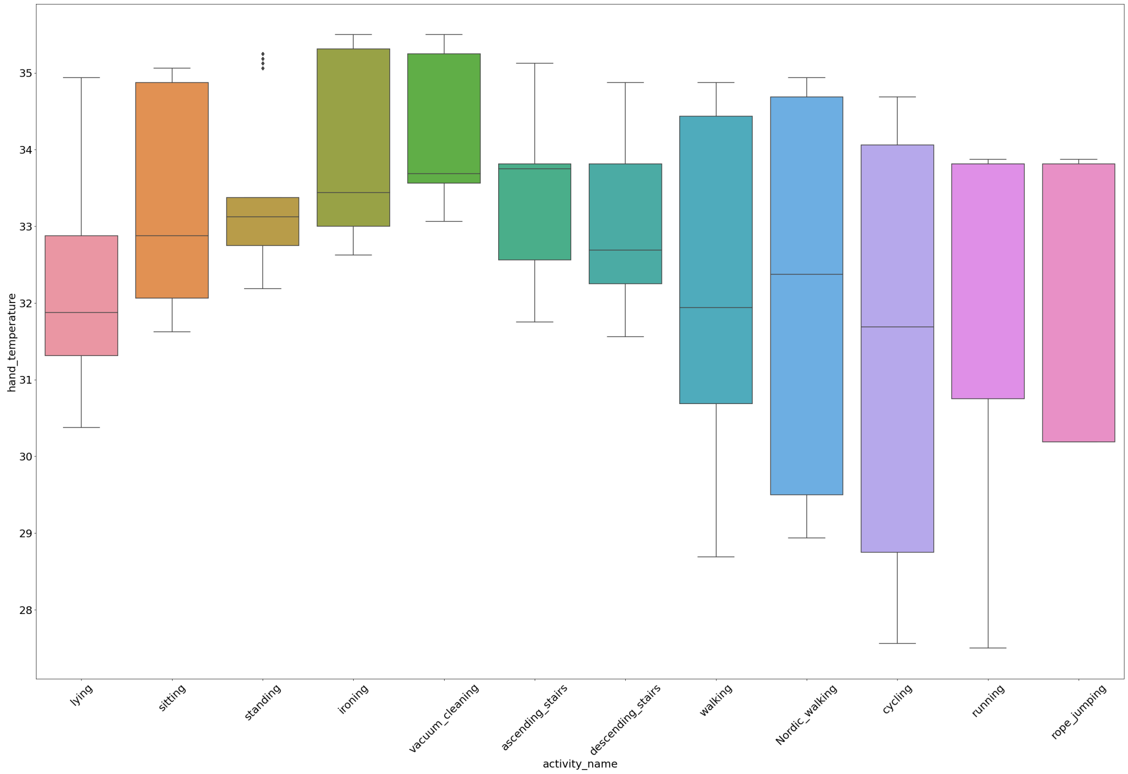
[119]:
```
rcParams["figure.figsize"] = 10, 10
rcParams["font.size"] =  10

ax = sns.boxplot(x="activity_type", y="hand_temperature", data=train)
ax.set_xticklabels(ax.get_xticklabels(), rotation=0)
plt.show()
```



1. Hand temperature of moderate activitie have a lot of outliers on the lower side.
2. There doesn't seem to be much difference in temperatures between activities.

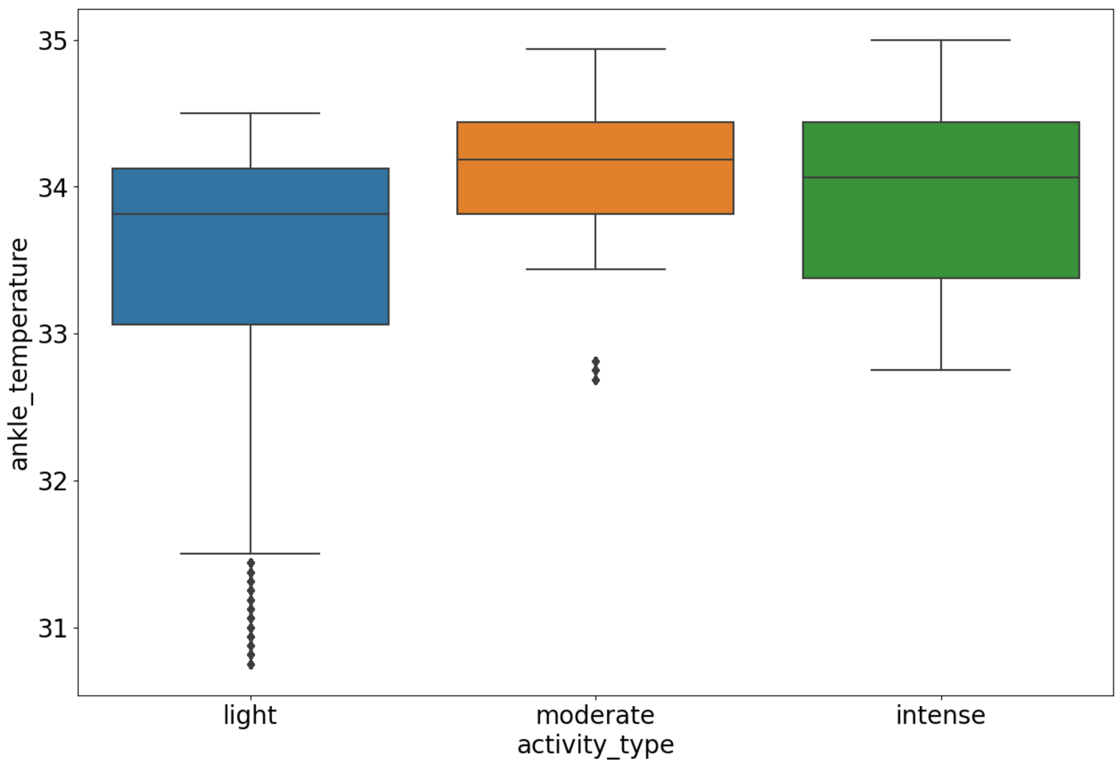
- Boxplot of hand temperature grouped by activity.

[125]:
```
rcParams["figure.figsize"] = 40, 25
rcParams["font.size"] =  22

ax = sns.boxplot(x="activity_name", y="hand_temperature", data=train)
ax.set_xticklabels(ax.get_xticklabels(), rotation=45)  # Rotating Text
plt.show()
```



- Boxplot of ankle temperature grouped by activity_type

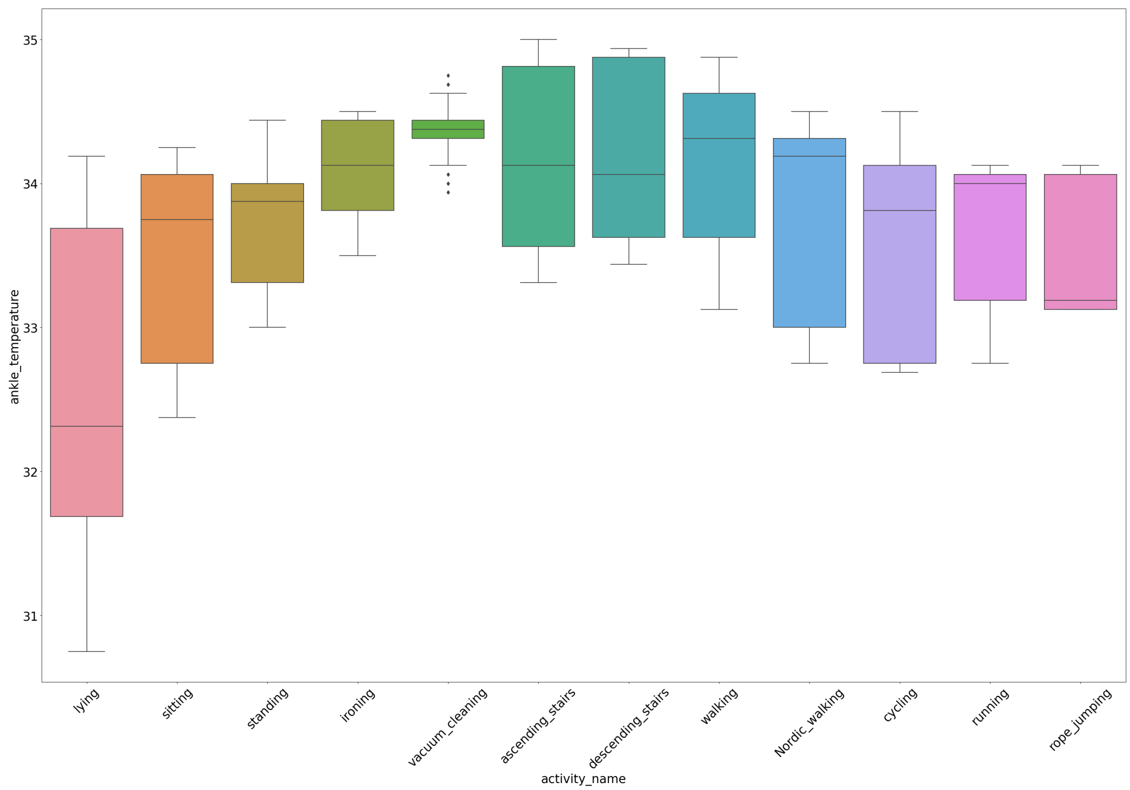
[126]:
```
rcParams["figure.figsize"] = 15, 10
rcParams["font.size"] =  20

ax = sns.boxplot(x="activity_type", y="ankle_temperature", data=train)
ax.set_xticklabels(ax.get_xticklabels(), rotation=0)
plt.show()
```



1. Ankle temperature of light and moderate activitie have outliers on the lower side.
2. There doesn't seem to be much difference in temperatures between activities.

- Boxplot of ankle temperature grouped by activity

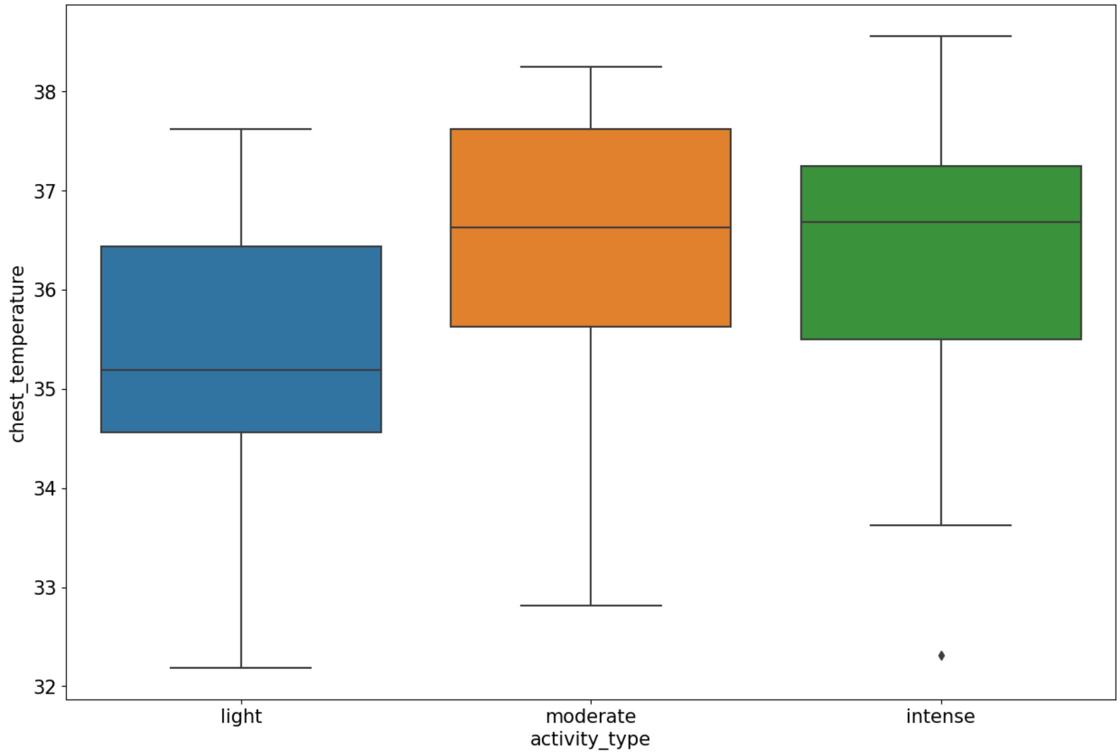
[128]:
```
rcParams["figure.figsize"] = 40, 25
rcParams["font.size"] =  25

ax = sns.boxplot(x="activity_name", y="ankle_temperature", data=train)
ax.set_xticklabels(ax.get_xticklabels(), rotation=45)  # Rotating Text
plt.show()
```



1. Outliers are mostly present in 'vacuum_cleaning' on the lower side.

- Boxplot of chest temperature grouped by activity_type

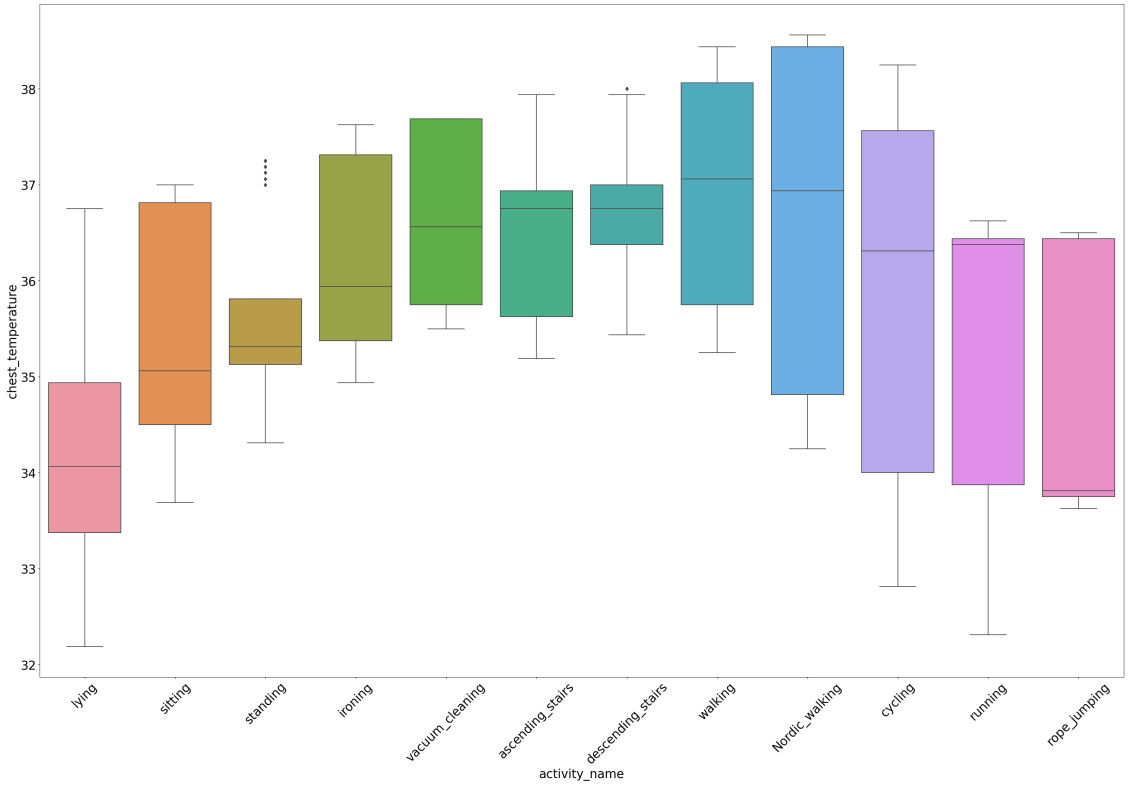
[130]:
```
rcParams["figure.figsize"] = 15, 10
rcParams["font.size"] =  15

ax = sns.boxplot(x="activity_type", y="chest_temperature", data=train)
ax.set_xticklabels(ax.get_xticklabels(), rotation=0)
plt.show()
```



1. For chest temperatures, only the 'intense' activity type has one outlier.
2. For this feature as well, there doesn't seem to be much difference between temperatures.

- Boxplot of chest temperature grouped by activity.

[134]:
```
rcParams["figure.figsize"] = 40, 25
rcParams["font.size"] =  25

ax = sns.boxplot(x="activity_name", y="chest_temperature", data=train)
ax.set_xticklabels(ax.get_xticklabels(), rotation=45)   # Rotating Text
plt.show()
```

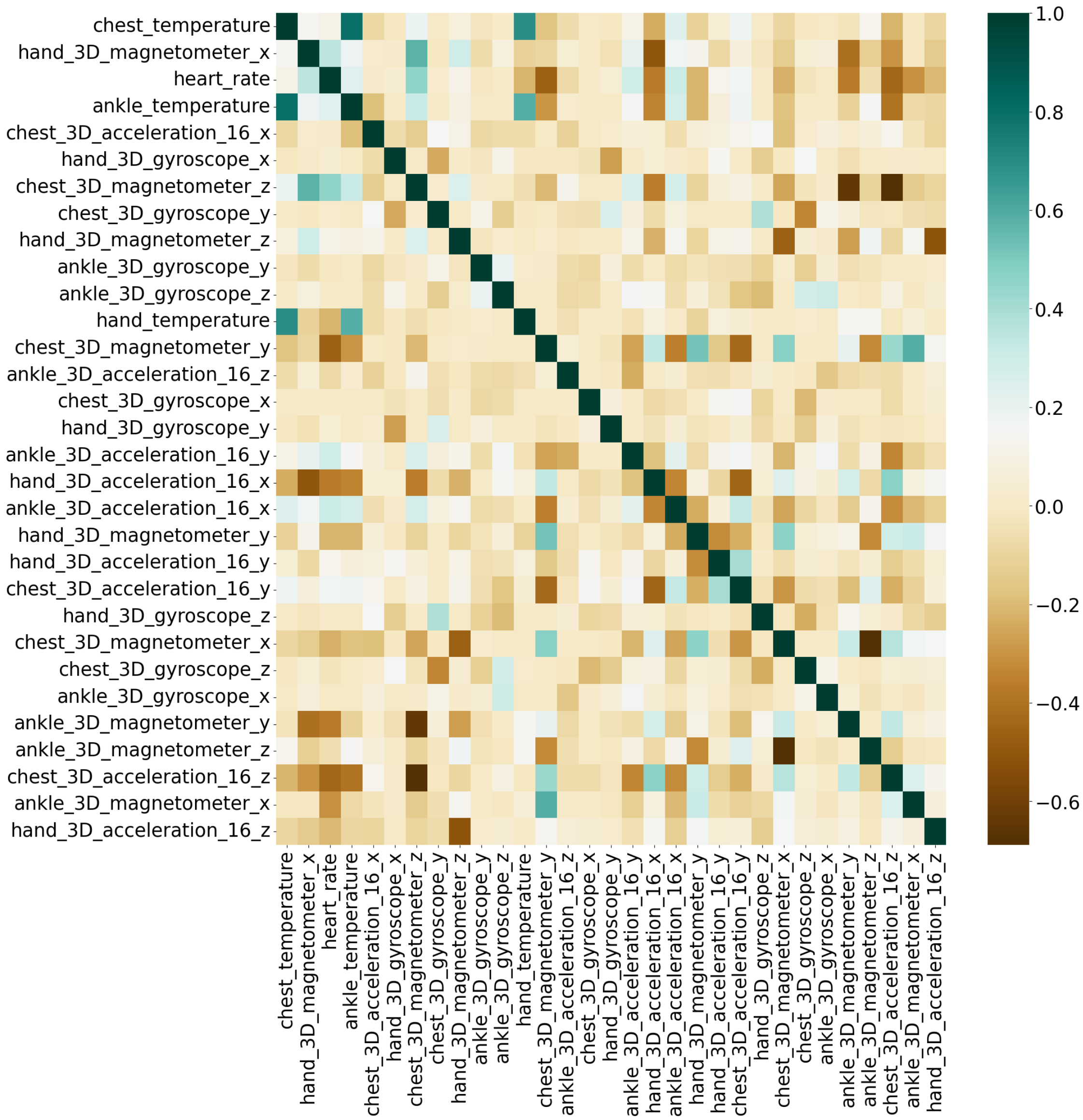


1. Most of the activities seem to have a skewed distribution for chest temperature.

- Correlation map for relevant features

[135]:
```
discard = [
    "activity_id",
    "activity",
    "time_stamp",
    "id",
]  # Columns to exclude from correlation map and descriptive statistics
train_trimmed = train[set(train.columns).difference(set(discard))]
```

[137]:
```
rcParams["figure.figsize"] = 20, 20 # Setting figure dimension
sns.heatmap(train_trimmed.corr(), cmap="BrBG") # Setting colorsheme and giving␣
 ↪correlation matrix as input
plt.show()
```



There seems to be a lot of significant correlations between many features

2.3.2 Descriptive Statistics

Subject Details

[138]:
```
display(subj_det)
```

	Subject ID	Sex	Age (years)	Height (cm)	Weight (kg)	\
0	101	Male	27	182	83	
1	102	Female	25	169	78	
2	103	Male	31	187	92	
3	104	Male	24	194	95	

```
4        105    Male          26          180           73
5        106    Male          26          183           69
6        107    Male          23          173           86
7        108    Male          32          179           87
8        109    Male          31          168           65

   Resting HR (bpm)  Max HR (bpm) Dominant hand        BMI
0                75           193         right  25.057360
1                74           195         right  27.309968
2                68           189         right  26.309017
3                58           196         right  25.241790
4                70           194         right  22.530864
5                60           194         right  20.603780
6                60           197         right  28.734672
7                66           188          left  27.152711
8                54           189         right  23.030045
```

Mean of heart rate and temperatures for each activity

[139]:
```
display(
    train.groupby(by="activity_name")[
        ["heart_rate", "chest_temperature", "hand_temperature",␣
↪"ankle_temperature"]
    ].mean()
)
```

```
                   heart_rate  chest_temperature  hand_temperature  \
activity_name
Nordic_walking     128.934574          36.640204         32.124192
ascending_stairs   132.404398          36.586821         33.381660
cycling            127.117356          35.894279         31.493273
descending_stairs  130.733971          36.701044         33.195439
ironing             94.586717          36.162343         33.845229
lying               78.609097          34.449084         32.508522
rope_jumping       165.084261          34.773034         31.562491
running            158.613734          35.262980         32.372712
sitting             82.242313          35.238913         33.025149
standing            95.112994          35.590371         33.445441
vacuum_cleaning    107.774620          36.530023         34.006465
walking            115.147733          36.964260         32.232594

                   ankle_temperature
activity_name
Nordic_walking             33.778505
ascending_stairs           34.146043
cycling                    33.617997
descending_stairs          34.147815
```

```
ironing                    34.103435
lying                      32.690584
rope_jumping               33.499627
running                    33.646088
sitting                    33.370173
standing                   33.742018
vacuum_cleaning            34.358195
walking                    34.152572
```

Descriptive info of relevant features

[140]:
```
display(train_trimmed.describe())
```

```
       chest_temperature  hand_3D_magnetometer_x    heart_rate  \
count      927296.000000           922733.000000  84798.000000
mean           35.961841               20.935601    109.912769
std             1.398668               25.717188     26.029426
min            32.187500             -103.941000     57.000000
25%            35.062500                2.499940     89.000000
50%            36.000000               22.012300    109.000000
75%            37.000000               40.078900    127.000000
max            38.562500              133.830000    183.000000

       ankle_temperature  chest_3D_acceleration_16_x  hand_3D_gyroscope_x  \
count      924261.000000               927296.000000        922733.000000
mean           33.776342                    0.240625             0.012205
std             0.761948                    1.736878             1.251699
min            30.750000                  -39.203400           -27.804400
25%            33.375000                   -0.580049            -0.387488
50%            34.000000                    0.355426            -0.005547
75%            34.312500                    1.035780             0.338614
max            35.000000                   27.522300            26.415800

       chest_3D_magnetometer_z  chest_3D_gyroscope_y  hand_3D_magnetometer_z  \
count            927296.000000         927296.000000           922733.000000
mean                  3.616971              0.009578              -25.306925
std                  23.695410              0.549806               22.172517
min                 -66.684700             -4.672250             -164.937000
25%                 -11.103325             -0.135082              -40.472500
50%                   1.054870             -0.000901              -24.964300
75%                  20.523525              0.162768              -10.695900
max                  96.358500              4.540310              101.758000

       ankle_3D_gyroscope_y  ...  chest_3D_acceleration_16_y  \
count         924261.000000  ...               927296.000000
mean              -0.029152  ...                    8.137126
std                0.571440  ...                    4.867817
```

```
min              -7.807450  ...                -25.955900
25%              -0.130252  ...                  5.901193
50%              -0.006023  ...                  9.265210
75%               0.085158  ...                  9.768020
max               6.410380  ...                107.825000

       hand_3D_gyroscope_z  chest_3D_magnetometer_x  chest_3D_gyroscope_z  \
count        922733.000000            927296.000000         927296.000000
mean             -0.000027                 4.634805             -0.025351
std               1.550689                18.799139              0.288043
min             -14.264700               -70.062700             -2.642760
25%              -0.354068                -6.381310             -0.127878
50%              -0.004496                 3.193960             -0.016610
75%               0.393650                14.399500              0.077227
max              14.338400                80.473900              2.716240

       ankle_3D_gyroscope_x  ankle_3D_magnetometer_y  ankle_3D_magnetometer_z  \
count         924261.000000            924261.000000            924261.000000
mean               0.007019                -0.243321                16.765524
std                1.022017                23.717646                21.612666
min              -13.385600              -137.908000              -102.716000
25%               -0.201214               -15.123500                 2.088300
50%                0.003077                -0.349413                20.468600
75%                0.095482                17.762500                30.407100
max               13.142500                94.247800               122.521000

       chest_3D_acceleration_16_z  ankle_3D_magnetometer_x  \
count               927296.000000            924261.000000
mean                    -1.164960               -34.292921
std                      4.721856                21.381194
min                    -53.401900              -172.865000
25%                     -3.827573               -45.705600
50%                     -1.233810               -35.768600
75%                      0.835319               -17.365300
max                     17.878100                91.551600

       hand_3D_acceleration_16_z
count              922733.000000
mean                    3.819557
std                     3.675067
min                   -38.907800
25%                     1.721670
50%                     3.721100
75%                     6.407160
max                    76.639600

[8 rows x 31 columns]
```

Variance of each axis of acceleration grouped by activities It is expected that variance along x,y and z axis for acceleration will be different for different activities.An attempt is made to investigate this.

[141]:
```
coordinates = [i for i in train.columns if "acceleration" in i]
display(train.groupby(by="activity_name")[coordinates].var())
```

```
                   hand_3D_acceleration_16_x  hand_3D_acceleration_16_y  \
activity_name
Nordic_walking                     25.629540                  58.398422
ascending_stairs                   23.543054                   7.566316
cycling                            17.791446                  15.649830
descending_stairs                  24.243241                  11.550501
ironing                            10.310193                  10.363109
lying                              15.855200                  13.097997
rope_jumping                       75.582150                  58.570550
running                           110.632062                 212.062379
sitting                            10.279172                  13.464679
standing                           20.283887                   6.405919
vacuum_cleaning                    18.818634                  15.030792
walking                            13.350637                   6.510350

                   hand_3D_acceleration_16_z  chest_3D_acceleration_16_x  \
activity_name
Nordic_walking                     10.603472                    2.575901
ascending_stairs                    5.824274                    2.765901
cycling                             9.939227                    0.715780
descending_stairs                  12.489337                    2.984501
ironing                            13.358504                    2.033213
lying                              13.166171                    4.153232
rope_jumping                       38.432095                    4.649561
running                            20.369313                    8.078898
sitting                            10.087709                    0.247896
standing                            3.793687                    0.679866
vacuum_cleaning                    11.900489                    5.039040
walking                             6.264966                    2.527115

                   chest_3D_acceleration_16_y  chest_3D_acceleration_16_z  \
activity_name
Nordic_walking                      15.242082                    6.148342
ascending_stairs                    10.775441                    5.624425
cycling                              3.711593                    4.622890
descending_stairs                   21.431527                    4.440417
ironing                              0.736907                    4.374898
lying                                6.062105                   12.668459
rope_jumping                       247.259665                   29.357327
running                            102.004097                    9.118545
sitting                              0.804636                    6.960005
```

```
standing                                 0.068863                     1.491701
vacuum_cleaning                         14.608914                    10.007767
walking                                 10.131506                     4.359562

                   ankle_3D_acceleration_16_x  ankle_3D_acceleration_16_y  \
activity_name
Nordic_walking                          33.484057                    92.289911
ascending_stairs                        42.033634                    55.288195
cycling                                 17.858721                    12.164733
descending_stairs                       47.242317                    70.014202
ironing                                  0.174704                     1.796946
lying                                    7.192011                    11.658936
rope_jumping                           118.280760                   234.159169
running                                161.038054                   228.653523
sitting                                  6.838823                     7.252916
standing                                 0.081663                     2.684077
vacuum_cleaning                          1.981244                     7.303831
walking                                 34.600482                    90.401580

                   ankle_3D_acceleration_16_z
activity_name
Nordic_walking                          15.376649
ascending_stairs                        28.013199
cycling                                  3.018095
descending_stairs                       25.787439
ironing                                  1.696242
lying                                   12.639666
rope_jumping                            50.880373
running                                 47.994581
sitting                                  8.381130
standing                                 1.072036
vacuum_cleaning                          4.504396
walking                                 17.312541
```

As we notice the variance is quite different along different axes for different activities

2.4 Hypothesis Testing

Based on the exploratory data analysis carried out, the following hypotheses are tested on the test set: - Heart rate of moderate activities are greater than heart rate of light activities. - Heart rate of intense activities are greater than heart rate of light activities. - Chest acceleration along z axis is greater while lying compared to z axis chest acceleration of other activities.

Based on the EDA we performed, it does not seem that the data is normally distributed. It is for this reason that Wilcoxon rank sum test was used to test the above hypothesis instead of the usual t-test which assumes that the samples follow a normal distribution. We test the above hypothesis using the confidence level of 5%.

2.4.1 Hypothesis 1

H_0(Null) : The heart rate during moderate activities are the same or lower than that of light activities. H_1(Alternate) : The heart rate during moderate activities are likely to be higher during lying compared to light activities.

[145]:
```python
test1 = test[
    test.activity_type == "moderate"
].heart_rate.dropna()  # Heart rate of moderate activities with nan values dropped
test2 = test[
    test.activity_type == "light"
].heart_rate.dropna()  # Heart rate of light activities with nan values dropped
print(ranksums(test1, test2, alternative="greater"))
```

```
RanksumsResult(statistic=188.93129841668087, pvalue=0.0)
```

Since we get a p value of 0 we have to reject the null hypothesis and accept the alternate hypotheses that the moderate intensity activities have higher heart rate than light intensity activities

2.4.2 Hypothesis 2

H_0(Null) : The heart rate during intense activities are the same or lower than that of light activities. H_1(Alternate) : The heart rate during intense activities are likely to be higher during than during lower activities.

[146]:
```python
test1 = test[
    test.activity_type == "intense"
].heart_rate.dropna()  # Heart rate of moderate activities with nan values dropped
test2 = test[
    test.activity_type == "light"
].heart_rate.dropna()  # Heart rate of light activities with nan values dropped
print(ranksums(test1, test2, alternative="greater"))
```

```
RanksumsResult(statistic=225.13542455896652, pvalue=0.0)
```

Since we get a p-value of 0 which is lower than 0.05 we reject the null hypothesis and accept the alternate hypothesis. It implies that intense activities have higher heart rate than light activities.

2.4.3 Hypothesis 3

$H_0(Null) : Thezaxischestaccelerationduringlyingislowerorsameasaccelerationofallotheractivities.H_1(Alternate) : T$

The regression equation is $Y_t = \beta X_t + \alpha$ where α is the intercept. It is tested if $\beta > 0$ A β value of greater than 0 proves that the vertical chest acceleration is more during lying when compared to other activities. Our hypothesis can be restated as $ H_0: $\beta \leq 0$ $$ $H\_1 : \beta > 0$

```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results                            
==============================================================================
Dep. Variable:     chest_3D_acceleration_16_z   R-squared:                       0.475
Model:                            OLS   Adj. R-squared:                  0.475
Method:                 Least Squares   F-statistic:                 9.120e+05
Date:                Thu, 06 Jan 2022   Prob (F-statistic):               0.00
Time:                        16:32:55   Log-Likelihood:            -2.6807e+06
No. Observations:             1006765   AIC:                         5.361e+06
Df Residuals:                 1006763   BIC:                         5.361e+06
Df Model:                           1                                         
Covariance Type:            nonrobust                                         
================================================================================
                   coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------
const           -2.7776      0.004   -764.045      0.000      -2.785      -2.771
lying_or_not    11.2204      0.012    955.000      0.000      11.197      11.243
==============================================================================
Omnibus:                263236.597   Durbin-Watson:                   0.108
Prob(Omnibus):               0.000   Jarque-Bera (JB):          1577756.632
Skew:                       -1.125   Prob(JB):                         0.00
Kurtosis:                    8.705   Cond. No.                         3.43
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
"""
```

We get a t-statistic of 955 and a p-value of 0. It implies that we can safely reject the null hypothesis considering the 5% confidence interval and accept the alternate hypothesis that vertical chest acceleration is indeed more during lying when compared with other activities as there is some significant evidence for this. Since we get a p-value of 0 which is lower than 0.05 we reject the null hypothesis and accept the alternate hypothesis.

2.5 Model Prediction

2.5.1 Data Split

For modelling, the data is split into train ,validation and test set.Train set is is used to calibrate the model, validation set is used to test various models and select the best one and the selected model is finally tested in the test set. The splitting is done by subjects, such that the train set has subjects (101, 103, 104, 105) the validation set has subjects (102, 106) and the test set has subjects (107, 108). These subjects are selected such that the validation and test set has approximately the same size and that length of training size is approximately equal to the combined length of testing and validation set. A model that is trained on one set of subjects, is able to generalize over other subjects, is highely likely to have achieved good generalization.

2.5.2 Computing Features

Model prediction is performed with the aim of determining the type of activity being performed.

For this task, it was decided that it is better to make predictions every 1 second instead of every 0.01 second as the activities are not likely to change that fast.To accomplish this, many additional features were computed using a sliding window approach.

A sliding window length of 256 rows or 2.56 seconds was used with a step size of 100 or 1 second.A window length of 256 was selected since it is a power of 2, which makes it easier to compute the Fast Fourier Transform, an algorithm used in the computation of spectral centroid.Features like mean, median, variance and spectral centroid were then computed for each window and for each feature.For example, if the first sliding window was taken from time t to (t+2.56), then the second sliding window would be taken from time (t+1) to (t+3.56).The original features were retained to compare their usefulness with the newly computed features.

These features that are computed over the rolling window are essentially much more noise resistant, because they rely on taking into account multiple samples instead of just relying on one highly error-prone sample.Mean, median and variance of a feature computed over the sliding window capture essential information about the distribution of the feature in the period covered by the sliding window.

In addition to the above three features, a frequency domain feature called spectral centroid was computed. A frequency-domain feature was considered because it was expected that different activities would have different rhythms for certain features which could be determined by figuring out, for instance, their most dominant frequency when that activity is being performed. Spectral Centroid is computed as,

$$\frac{\sum_{n=0}^{N-1} f(n)x(n)}{\sum_{n=0}^{N-1} x(n)}$$

where $f(n)$ is the frequency value, while $x(n)$ is the absolute value of the Fourier coefficient of that frequency. In this analysis, the spectral centroid gives a value between 0 and 1. The actual frequency can be found out by multiplying this value by 256.

A high value of spectral centroid implies the dominance of high frequency signals in the data and vice versa.

[69]:
```
# Function for copying pandas table into clipboard in markdown format
copy = lambda x:pd.io.clipboards.to_clipboard(x.to_markdown(),excel=False)
```

[6]:
```
clean_data = pd.read_pickle("clean_act_data.pkl")
discard = [
    "activity_id",
    "activity",
    "activity_name",
    "time_stamp",
    "id",
    "activity_type",
]  # Columns to exclude from descriptive stat
```

[7]:
```
def spectral_centroid(signal):
    spectrum = np.abs(np.fft.rfft(signal)) # Computing absolute value of fourier␣
↪coefficient
    normalized_spectrum = spectrum / np.sum(
        spectrum
    )  # similar to  a probability mass function
    normalized_frequencies = np.linspace(0, 1, len(spectrum))
    spectral_centroid = np.sum(normalized_frequencies * normalized_spectrum)
    return spectral_centroid
```

[8]:
```
def sliding_window_feats(data, feats, win_len, step):
    final = []
    i = 0
    for i in range(0, len(data), 100):
        if (i + 256) > len(data):
            break
        temp = data.iloc[i : i + 256]
        temp1 = pd.DataFrame()
        for feat in feats:
            # Computing sliding window features
            temp1[f"{feat}_roll_mean"] = [temp[feat].mean()]
            temp1[f"{feat}_roll_median"] = [temp[feat].median()]
            temp1[f"{feat}_roll_var"] = [temp[feat].var()]
            temp1[f"{feat}_spectral_centroid"] = [spectral_centroid(temp[feat])]
        temp1["time_stamp"] = [list(temp.time_stamp.values)[-1]]
        temp1[feats] = [temp[feats].iloc[-1]]
        temp1["activity_name"] = [temp["activity_name"].iloc[-1]]
        temp1["activity_type"] = [temp["activity_type"].iloc[-1]]
        final.append(temp1)
    final_data = pd.concat(final)
    return final_data
```

[9]:
```
class modelling:
    def __init__(
        self,
        clean_data,
        features,
        train_subjects=[101, 103, 104, 105],
        val_subjects=[102, 106],
        test_subjects=[107, 108],
    ):
        # Initializing variables
        self.clean_data = clean_data
        self.train_subjects = train_subjects
        self.val_subjects = val_subjects
        self.test_subjects = test_subjects
        self.features = features
```

```python
    def split_input_data(self):
        # Splitting input features into train,test and val
        train = self.clean_data[self.clean_data.id.isin(self.train_subjects)]
        val = self.clean_data[self.clean_data.id.isin(self.val_subjects)]
        test = self.clean_data[self.clean_data.id.isin(self.test_subjects)]
        x_train = train[self.features]
        x_val = val[self.features]
        x_test = test[self.features]
        return train, val, test, x_train, x_val, x_test
    def split_one_act(self, activity):
        # Function to give input ouput matrix for one activity
        train, val, test, x_train, x_val, x_test = self.split_input_data()
        one_hot_label = lambda x: 1 if x == activity else 0 # generate dummy␣
↪variable for one activity
        y_train = train.activity_name.apply(lambda x: one_hot_label(x))
        y_val = val.activity_name.apply(lambda x: one_hot_label(x))
        y_test = test.activity_name.apply(lambda x: one_hot_label(x))
        return x_train, x_val, x_test, y_train, y_val, y_test
    def train_test_split_actname(self):
        # Function to give input ouput matrix for all 12 activies

        le = preprocessing.LabelEncoder()
        train, val, test, x_train, x_val, x_test = self.split_input_data()
        y_train = le.fit_transform(train.activity_name)
        y_val = le.fit_transform(val.activity_name)
        y_test = le.fit_transform(test.activity_name)
        return x_train, x_val, x_test, y_train, y_val, y_test, le
```

[151]:
```python
def final_sliding_window(clean_data):
    # Function for generating sliding window
    feats = [i for i in clean_data.columns if i not in discard]
    final = []
    for i in clean_data.id.unique():
        temp = clean_data[clean_data.id == i] #
        temp = sliding_window_feats(temp, feats, 256, 100)
        temp["id"] = [i] * len(temp)
        final.append(temp)
    clean_data_feats = pd.concat(final)
    clean_data_feats.to_pickle("activity_short_data.pkl")
    return clean_data_feats
```

Warning: This cell takes a very long time to run.It is advised to use a debugger to run it line by line to check it.

[ ]:
```python
final_sliding_window(clean_data)
```

```
[17]: clean_data_feats = pd.read_pickle("activity_short_data.pkl") # Load saved clean␣
       ↪data
      features = [i for i in clean_data_feats.columns if i not in discard]
      model = modelling(clean_data_feats, features)
```

```
[18]: (
          x_train,
          x_val,
          x_test,
          y_train,
          y_val,
          y_test,
          le,
      ) = model.train_test_split_actname() # generate input ouput matrix
```

```
[20]: x_train_labels = pd.DataFrame()
      x_train_labels["activity_name"] = le.inverse_transform(y_train)
```

2.5.3 Feature Selection Process

1. First, Clustering is performed for each feature.V-measure is used to determine how good the cluster is.If ncluster is the number of clusters chooses for clustering, then clustering is performed using different values of nclusters and all of them are evaluated based on v-measure.Minimum size choosen is 12 as that is the number of activities and ideally every activity should be associated with one cluster. The 12 activity names are considered as the class labels.
2. Same cluster size is used for clustering each feature.
3. The average of v-measure score for all the feature is taken to determine the final v-measure score.
4. The v-measure score for different cluster sizes give almost the same score, hence the cluster size of 100 is choosen for each feature.
5. The probability of an activity i given a cluster j is computed as

$$p_{ij} = \frac{n(i \cap j)}{n(j)}$$

,where $n(i \cap j)$ is the count of occurence of activity i and j together and $n(j)$ is the count of occurence of cluster j

6. For each feature a precision score is calculated activty i such that

$$P_{ik} = \frac{\sum_{j=1}^{N} p_{ij} C_{ij}}{\sum_{j=1}^{N} C_{ij}}$$

, where C_{ij} is the number of rows of cluster j present in activity i. A higher value of precision for an activity implies that there are many clusters that have samples mostly for this activity.

7. The precision score gives us a good idea of how good a feature will be in predicting a particular activity.

[10]:
```
def precision(df):
    # Function for computing precision
    df.columns = ["activity", "labels"]
    act_precision = dict()
    for act in df.activity.unique():
        num = 0
        denom = 0
        df_act = df[df.activity == act]
        c_lab = df_act.labels.value_counts()
        for lab in df_act.labels.unique():
            clust_prob = len(df[(df.activity == act) & (df.labels == lab)]) /␣
 ↪len(
                df[df.labels == lab]
            )
            num = num + clust_prob * c_lab[lab]
            denom = denom + c_lab[lab]
        act_precision[act] = num / denom
    return act_precision
```

[11]:
```
def best_cluster():
    # Function for determining  best cluster
    v_measure = dict()
    for nclust in range(12, 112, 5):
        clust_vmeasure = []
        for col in x_train.columns:
            clust = cluster.KMeans(init="random", random_state=0,␣
 ↪n_clusters=nclust)
            clust.fit(x_train[[col]])
            x_train_labels[f"{col}_label"] = clust.predict(x_train[[col]])
            clust_vmeasure.append(
                v_measure_score(y_train, x_train_labels[f"{col}_label"])
            )
        v_measure[nclust] = [np.array(clust_vmeasure).mean()]
    nclust_max = max(v_measure, key=v_measure.get)
    print(f"best cluster size : {nclust_max}")
    return v_measure
```

Warning: The cell below takes a very long time to run. A debugger can be used to check it by executing the function line by line.

[ ]:
```
vm = pd.DataFrame(best_cluster())
vm.to_pickle("v_measure.pkl")
```

V measure of different cluster size

[105]:
```
vm = pd.read_pickle("v_measure.pkl")
print("A condensed view of average v-measure of cluster of different sizes")
display(vm[vm.columns[0:9]])
# Not much difference found so using 100 clusters
```

```
A condensed view of average v-measure of cluster of different sizes

         12        17        22        27        32       37        42  \
0  0.217367  0.218729  0.217423  0.215188  0.212477  0.20998  0.207591

         47        52
0  0.205433  0.203584
```

[14]:
```
def activity_precision():
    # Function for computing precision of each activity
    label_act_precision = dict()
    for i in x_train.columns:
        clust = cluster.KMeans(
            init="random", random_state=0, n_clusters=1000
        )
        clust.fit(x_train[[i]])
        x_train_labels[f"{i}_label"] = clust.predict(x_train[[i]])
        label_act_precision[i] = precision(
            x_train_labels[["activity_name", f"{i}_label"]]
        )
    return label_act_precision
```

Warning: The cell below takes a very long time to run. A debugger can be used to check it by executing the function line by line.

[21]:
```
lab_score = pd.DataFrame(activity_precision())
lab_score.to_pickle("precision_score1.pkl")
```

[22]:
```
lab_score = pd.read_pickle("precision_score.pkl")
```

[135]:
```
acts = list(lab_score.T.columns)
for act in acts:
  temp = lab_score[lab_score.index==act]
  print(act)
  print(f"Maximum Precision Score: {temp.max(axis=1)}")
```

```
lying
Maximum Precision Score: lying    0.883875
dtype: float64
sitting
Maximum Precision Score: sitting    0.472997
dtype: float64
standing
```

```
Maximum Precision Score: standing    0.412574
dtype: float64
ironing
Maximum Precision Score: ironing    0.464035
dtype: float64
vacuum_cleaning
Maximum Precision Score: vacuum_cleaning    0.497042
dtype: float64
ascending_stairs
Maximum Precision Score: ascending_stairs    0.431175
dtype: float64
descending_stairs
Maximum Precision Score: descending_stairs    0.384129
dtype: float64
walking
Maximum Precision Score: walking    0.570877
dtype: float64
Nordic_walking
Maximum Precision Score: Nordic_walking    0.482387
dtype: float64
cycling
Maximum Precision Score: cycling    0.528448
dtype: float64
running
Maximum Precision Score: running    0.883534
dtype: float64
rope_jumping
Maximum Precision Score: rope_jumping    0.810501
dtype: float64
```

From above analysis, it is observed that lying has the maximum precision score, implying it is easier to predict. The analysis is taken forward using this information.

[136]:
```
def log_reg(model, split_type, activity_type):
    # Function for acrrying out the Logistic Regression modelling
    if split_type == "one_act":
        x_train, x_val, x_test, y_train, y_val, y_test = model.split_one_act(
            activity_type
        )
    else:
        (
            x_train,
            x_val,
            x_test,
            y_train,
            y_val,
            y_test,
            le,
```

```
        ) = model.train_test_split_actname()
    pca = PCA(n_components=0.99)
    x_train = pca.fit_transform(x_train)
    x_val = pca.transform(x_val)
    x_test = pca.transform(x_test)
    f1 = []
    acc = []
    print(f"Principal Component Feature size: {x_train.shape[1]}")
    for lam in np.arange(0.1, 2, 0.1):
        lr = LogisticRegression(solver="saga", random_state=30, C=1 / lam)
        lr.fit(x_train, y_train)
        f1.append(f1_score(y_val, lr.predict(x_val), average="macro"))
        acc.append(accuracy_score(y_val, lr.predict(x_val)))
    df_lr = pd.DataFrame()
    df_lr["validation_accuracy"] = acc
    df_lr["f1"] = f1
    df_lr["lambda"] = np.arange(0.1, 2, 0.1)
    return df_lr
```

A prediction model is made which determines if the subject is lying or not at some time t. Logistic Regression is used to make the prediction and a set of 19 different regularization parameters(λ) are tested on the validation set to determine the best one.The features are sorted in descending order of precision score and the top 4 features with maximum precision score for lying are selected.Before feeding the features into the Logistic Regression model, PCA transformation is performed such that the resulting principal components explain 99% variance of the original data. PCA is important to perform before Logistic Regression because Logistic Regression works on the assumption that the input features are not correlated with each other.PCA transforms orignal input data into non-correlated principal components which can be used as inputs into the model.It is important to note that the eigenvectors and the eigenvalues are only computed on the training set. The Principle Components of the test set and validation set are computes using the eigenvectors already computed from the training set. If these eigenvalues were computed over the entire dataset, that would mean that the model being trained on the training set is incorporating information from the validation and testing set as well which would jeopardize the prupose of splitting tha data into different sets.

2.5.4 Metrics for Measuring Performance

For determining how good this model is ,we use something called F1-Score. In prediction tasks such as the one which concerns us, where there will be a huge class imbalance. Accuracy is not a good measure for such tasks as simply predicting the class with most number of occurences will give us a very high accuracy score. Therefore we need 2 additional measure for determininbg how good the classification is.

Preicison is computed as

$$\frac{TP}{TP + FP}$$

,

where TP is the number of True Positives and FP is the number of False Positives.

Recall is computed as

$$\frac{TP}{TP + FN}$$

,

where FN is the number of false negatives.

Finally the F score is the harmonic mean of Precision and recall. F1 Score close to one implies that the classification model has a good balance of precision and recall.

For the classification the best 4 features, i.e., the features with the highest precision for the 'lying' and the worst 4 features with lowest precision for the same activity are considered and prediction using both these types of features is performed.

[25]:
```
def one_act_model(act, low_index, up_index, lab_score):
    # Return results for one activity classification modelling
    lab_score = lab_score.T
    best_feats = list(
        lab_score[act].sort_values(ascending=False).index[low_index:up_index]
    )
    model = modelling(clean_data_feats, best_feats)
    df_lr = log_reg(model, "one_act", "lying")
    return df_lr, model, best_feats
```

[121]:
```
df_lr_best_feat, best_model, best_feats = one_act_model("lying", 0, 4, lab_score)
df_lr_worst_feat, worst_model, worst_feats = one_act_model("lying", -4, -1,␣
↪lab_score)
```

```
Feature size: 3
Feature size: 1
```

[70]:
```
copy(df_lr_best_feat)
```

Best Features Classification Performance

	validation_accuracy	f1	lambda
0	0.994152	0.981657	0.1
1	0.994152	0.981657	0.2
2	0.994152	0.981657	0.3
3	0.994152	0.981657	0.4
4	0.994152	0.981657	0.5
5	0.994152	0.981657	0.6
6	0.994152	0.981657	0.7
7	0.994152	0.981657	0.8
8	0.994152	0.981657	0.9
9	0.994152	0.981657	1
10	0.994152	0.981657	1.1
11	0.994152	0.981657	1.2
12	0.994152	0.981657	1.3

	validation_accuracy	f1	lambda
13	0.994152	0.981657	1.4
14	0.994152	0.981657	1.5
15	0.994152	0.981657	1.6
16	0.994152	0.981657	1.7
17	0.994152	0.981657	1.8
18	0.994152	0.981657	1.9

[71]:
```
copy(df_lr_worst_feat)
```

Worst Feature Classification Performance

	validation_accuracy	f1	lambda
0	0.909747	0.47637	0.1
1	0.909747	0.47637	0.2
2	0.909747	0.47637	0.3
3	0.909747	0.47637	0.4
4	0.909747	0.47637	0.5
5	0.909747	0.47637	0.6
6	0.909747	0.47637	0.7
7	0.909747	0.47637	0.8
8	0.909747	0.47637	0.9
9	0.909747	0.47637	1
10	0.909747	0.47637	1.1
11	0.909747	0.47637	1.2
12	0.909747	0.47637	1.3
13	0.909747	0.47637	1.4
14	0.909747	0.47637	1.5
15	0.909747	0.47637	1.6
16	0.909747	0.47637	1.7
17	0.909747	0.47637	1.8
18	0.909747	0.47637	1.9

The F1-Scores and accuracy clearly tells us that the features which has the highest precision for lying outperform the ones with the lowest precision for lying. This shows us that the precision metric is a really good way of determining which features to use for classification problems. Since all λ values give the same results we use just $\lambda = 0.9$ and test this final model on test set.

[50]:
```
lam = 0.9
x_train, x_val, x_test, y_train, y_val, y_test = best_model.
↪split_one_act("lying")
lr = LogisticRegression(solver="saga", random_state=30, C=1 / lam)
lr.fit(x_train, y_train)
y_pred = lr.predict(x_test)
print("Test Set Results")
```

```
print(classification_report(y_test, y_pred))
print(f"Time spent lying (predicted): {list(y_pred).count(1)} seconds")
print(f"Time spent lying (actual): {list(y_test).count(1)} seconds")
```

```
Test Set Results
              precision    recall  f1-score   support

           0       0.99      1.00      1.00      4451
           1       1.00      0.95      0.97       494

    accuracy                           0.99      4945
   macro avg       1.00      0.97      0.98      4945
weighted avg       0.99      0.99      0.99      4945

Time spent lying (predicted): 467 seconds
Time spent lying (actual): 494 seconds
```

[173]:
```
print(f"Best Features for lying: {best_feats}")
print(f"Worst Features for lying: {worst_feats}")
```

```
Best Features for lying: ['ankle_3D_acceleration_16_y_roll_median',
'ankle_3D_acceleration_16_x_roll_mean',
'ankle_3D_acceleration_16_x_roll_median',
'chest_3D_acceleration_16_z_roll_median']
Worst Features for lying: ['heart_rate_roll_var', 'ankle_temperature_roll_var',
'chest_temperature_roll_var']
```

Now an attempt is made to make a model to predict all 12 activities. To select features for this, the top 4 features with the highest precision for each activity is selected.

[54]:
```
feat_score = lab_score.T
best_feats = np.concatenate(
    [
        list(feat_score[act].sort_values(ascending=False).index[0:4])
        for act in feat_score.columns
    ]
)
best_feats = list(set(best_feats))    # Removed duplicates
```

[55]:
```
cluster_pred = modelling(clean_data_feats, best_feats)
(
    x_train,
    x_val,
    x_test,
    y_train,
    y_val,
    y_test,
    le,
```

```
) = cluster_pred.train_test_split_actname()
```

[56]:
```
def determine_ncluster(x_train, y_train):
    # Compute v measure for all cluster sizes for dinal clustering task
    v_measure_feats = defaultdict(list)
    for ncluster in range(12, 100, 5):
        clust = cluster.KMeans(init="random", random_state=0,␣
↪n_clusters=ncluster)
        clust.fit(x_train)
        y_lab = clust.predict(x_train)
        v_measure_feats[ncluster].append(v_measure_score(y_train, y_lab))
        print(f"cluster {ncluster} done")
    return v_measure_feats
```

Warning: The cell below takes a very long time to run. A debugger can be used to check it by executing the function line by line.

[58]:
```
v_measure_feats = determine_ncluster(x_train, y_train)
pd.DataFrame(v_measure_feats).to_pickle("multiple_feature_vmeasure.pkl")
```

```
cluster 12 done
cluster 17 done
cluster 22 done
cluster 27 done
cluster 32 done
cluster 37 done
cluster 42 done
cluster 47 done
cluster 52 done
cluster 57 done
cluster 62 done
cluster 67 done
cluster 72 done
cluster 77 done
cluster 82 done
cluster 87 done
cluster 92 done
cluster 97 done
```

K-Means clustering is performed over the n dimensional data obtained from selecting high precision features and these clusters are used to compute the probability of the activities given a particular cluster.

Yet again, to determine the optimal number of clusters, V-measure score is used and 57 is determined to be the optimal number of clusters

The probability is computed using the following formula

$$p_{ij} = \frac{n(i \cap j)}{n(j)}$$

,where $n(i \cap j)$ is the count of occurence of activity i and j together and $n(j)$ is the count of occurence of cluster j. p_{ij} is the probability of activity i given cluster j.

[62]:
```
v_measure = pd.read_pickle("multiple_feature_vmeasure.pkl")
ncluster = v_measure.idxmax(axis=1).values[0]
print(f"Optimal No. of Clusters:{ncluster}")
clf = cluster.KMeans(init="random", n_clusters=ncluster, random_state=0)
clf.fit(x_train, y_train)
x_train_labels = x_train.copy()
x_train_labels["labels"] = clf.predict(x_train)
x_train_labels["activity_name"] = le.inverse_transform(y_train)
xc = pd.DataFrame(
    index=x_train_labels.activity_name.unique(),
    columns=x_train_labels.labels.unique(),
)
for i in range(ncluster):
    temp = x_train_labels[x_train_labels.labels == i]
    for j in x_train_labels.activity_name.unique():
        clust_prob = len(temp[temp.activity_name == j]) / len(temp)
        xc.loc[j, i] = clust_prob # Computing probability of activity given a␣
 ↪cluster
print("Probability of activity given a cluster label:")
xc = xc.astype("float")
```

```
Optimal No. of Clusters:52
Probability of activity given a cluster label:
```

[96]:
```
xc.index.name = 'Activity Type'
xc.columns.name = 'Cluster Label'
print("A condesed view of the probability table \n")
print("(Probability of a  ceratin activty given a cluster label)")
xc[xc.columns[0:4]]
```

```
A condesed view of the probability table

(Probability of a  ceratin activty given a cluster label)
```

[96]:

Cluster Label	37	18	50	51
Activity Type				
lying	0.193548	0.0625	0.119266	0.042857
sitting	0.161290	0.6250	0.009174	0.000000
standing	0.564516	0.0000	0.000000	0.000000
ironing	0.000000	0.0000	0.009174	0.000000
vacuum_cleaning	0.000000	0.3125	0.477064	0.728571
ascending_stairs	0.032258	0.0000	0.000000	0.085714
descending_stairs	0.000000	0.0000	0.091743	0.128571
walking	0.048387	0.0000	0.110092	0.000000
Nordic_walking	0.000000	0.0000	0.110092	0.000000

```
cycling            0.000000  0.0000  0.055046  0.014286
running            0.000000  0.0000  0.000000  0.000000
rope_jumping       0.000000  0.0000  0.018349  0.000000
```

[93]:
```
xc
```

[93]:
```
Cluster Label            37      18        50        51   0        47  \
Activity Type
lying              0.193548  0.0625  0.119266  0.042857  1.0  0.013378
sitting            0.161290  0.6250  0.009174  0.000000  0.0  0.973244
standing           0.564516  0.0000  0.000000  0.000000  0.0  0.003344
ironing            0.000000  0.0000  0.009174  0.000000  0.0  0.000000
vacuum_cleaning    0.000000  0.3125  0.477064  0.728571  0.0  0.010033
ascending_stairs   0.032258  0.0000  0.000000  0.085714  0.0  0.000000
descending_stairs  0.000000  0.0000  0.091743  0.128571  0.0  0.000000
walking            0.048387  0.0000  0.110092  0.000000  0.0  0.000000
Nordic_walking     0.000000  0.0000  0.110092  0.000000  0.0  0.000000
cycling            0.000000  0.0000  0.055046  0.014286  0.0  0.000000
running            0.000000  0.0000  0.000000  0.000000  0.0  0.000000
rope_jumping       0.000000  0.0000  0.018349  0.000000  0.0  0.000000

Cluster Label            32   7        40        36        14        21  \
Activity Type
lying              0.789474  1.0  0.000000  0.000000  0.012308  0.000000
sitting            0.000000  0.0  0.589520  0.012448  0.024615  0.846154
standing           0.000000  0.0  0.384279  0.000000  0.030769  0.000000
ironing            0.000000  0.0  0.000000  0.966805  0.916923  0.000000
vacuum_cleaning    0.210526  0.0  0.000000  0.000000  0.015385  0.000000
ascending_stairs   0.000000  0.0  0.000000  0.000000  0.000000  0.076923
descending_stairs  0.000000  0.0  0.000000  0.020747  0.000000  0.061538
walking            0.000000  0.0  0.004367  0.000000  0.000000  0.015385
Nordic_walking     0.000000  0.0  0.000000  0.000000  0.000000  0.000000
cycling            0.000000  0.0  0.000000  0.000000  0.000000  0.000000
running            0.000000  0.0  0.021834  0.000000  0.000000  0.000000
rope_jumping       0.000000  0.0  0.000000  0.000000  0.000000  0.000000

Cluster Label            11        8         34        23        3         27  \
Activity Type
lying              0.000000  0.000000  0.006645  0.127753  0.000000  0.000000
sitting            0.000000  0.000000  0.000000  0.030837  0.070111  0.077982
standing           0.979167  0.000000  0.063123  0.044053  0.380074  0.600917
ironing            0.000000  0.007752  0.853821  0.198238  0.007380  0.036697
vacuum_cleaning    0.000000  0.038760  0.073090  0.594714  0.298893  0.275229
ascending_stairs   0.000000  0.054264  0.000000  0.000000  0.007380  0.009174
descending_stairs  0.020833  0.085271  0.000000  0.004405  0.029520  0.000000
walking            0.000000  0.000000  0.000000  0.000000  0.092251  0.000000
Nordic_walking     0.000000  0.077519  0.003322  0.000000  0.007380  0.000000
```

```
cycling             0.000000  0.713178  0.000000  0.000000  0.088561  0.000000
running             0.000000  0.000000  0.000000  0.000000  0.018450  0.000000
rope_jumping        0.000000  0.023256  0.000000  0.000000  0.000000  0.000000

Cluster Label             45        12        48        15         4        41  \
Activity Type
lying               0.000000  0.000000  0.000000  0.005714  0.000000  0.000000
sitting             0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
standing            0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
ironing             0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
vacuum_cleaning     0.872727  0.902778  0.312883  0.148571  0.087302  0.483871
ascending_stairs    0.000000  0.069444  0.000000  0.051429  0.214286  0.306452
descending_stairs   0.009091  0.013889  0.000000  0.005714  0.309524  0.145161
walking             0.009091  0.000000  0.042945  0.017143  0.015873  0.000000
Nordic_walking      0.000000  0.000000  0.049080  0.000000  0.063492  0.000000
cycling             0.109091  0.013889  0.595092  0.765714  0.238095  0.064516
running             0.000000  0.000000  0.000000  0.000000  0.063492  0.000000
rope_jumping        0.000000  0.000000  0.000000  0.005714  0.007937  0.000000

Cluster Label             13        33         6        10        16        30  \
Activity Type
lying               0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
sitting             0.007782  0.000000  0.000000  0.000000  0.000000  0.000000
standing            0.704280  0.000000  0.000000  0.000000  0.000000  0.000000
ironing             0.000000  0.000000  0.882353  0.000000  0.000000  0.000000
vacuum_cleaning     0.151751  0.073529  0.066176  0.005587  0.006536  0.000000
ascending_stairs    0.116732  0.040441  0.003676  0.251397  0.562092  0.521368
descending_stairs   0.019455  0.058824  0.018382  0.078212  0.294118  0.410256
walking             0.000000  0.018382  0.000000  0.491620  0.117647  0.068376
Nordic_walking      0.000000  0.000000  0.018382  0.145251  0.000000  0.000000
cycling             0.000000  0.790441  0.011029  0.027933  0.019608  0.000000
running             0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
rope_jumping        0.000000  0.018382  0.000000  0.000000  0.000000  0.000000

Cluster Label             31        26        49         9        46        44  \
Activity Type
lying               0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
sitting             0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
standing            0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
ironing             0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
vacuum_cleaning     0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
ascending_stairs    0.052960  0.236181  0.042254  0.216216  0.150407  0.468085
descending_stairs   0.060748  0.040201  0.323944  0.729730  0.077236  0.489362
walking             0.878505  0.185930  0.626761  0.027027  0.012195  0.010638
Nordic_walking      0.007788  0.361809  0.007042  0.009009  0.613821  0.021277
cycling             0.000000  0.080402  0.000000  0.000000  0.146341  0.000000
running             0.000000  0.045226  0.000000  0.000000  0.000000  0.000000
```

```
rope_jumping       0.000000  0.050251  0.000000  0.018018  0.000000  0.010638

Cluster Label             2        39     5        42   25   38   43  \
Activity Type
lying              0.000000  0.000000  0.00  0.000000  0.0  0.0  0.0
sitting            0.000000  0.008621  0.00  0.018519  0.0  0.0  0.0
standing           0.000000  0.000000  0.00  0.000000  0.0  0.0  0.0
ironing            0.000000  0.000000  0.00  0.012346  0.0  0.0  0.0
vacuum_cleaning    0.000000  0.000000  0.00  0.000000  0.0  0.0  0.0
ascending_stairs   0.068127  0.000000  0.00  0.000000  0.0  0.0  0.0
descending_stairs  0.051095  0.000000  0.00  0.000000  0.0  0.0  0.0
walking            0.671533  0.000000  0.00  0.024691  0.0  0.0  0.0
Nordic_walking     0.189781  0.982759  0.94  0.932099  0.0  0.0  0.0
cycling            0.019465  0.000000  0.04  0.000000  0.0  0.0  0.0
running            0.000000  0.008621  0.01  0.006173  1.0  1.0  1.0
rope_jumping       0.000000  0.000000  0.01  0.006173  0.0  0.0  0.0

Cluster Label            22   24   29   28         1        20        19   17  \
Activity Type
lying              0.000000  0.0  0.0  1.0  0.000000  0.000000  0.000000  1.0
sitting            0.000000  0.0  0.0  0.0  0.497984  0.000000  0.973214  0.0
standing           0.000000  0.0  0.0  0.0  0.439516  0.000000  0.000000  0.0
ironing            0.000000  0.0  0.0  0.0  0.016129  0.000000  0.000000  0.0
vacuum_cleaning    0.000000  0.0  0.0  0.0  0.046371  0.000000  0.026786  0.0
ascending_stairs   0.000000  0.0  0.0  0.0  0.000000  0.243478  0.000000  0.0
descending_stairs  0.000000  0.0  0.0  0.0  0.000000  0.608696  0.000000  0.0
walking            0.000000  0.0  0.0  0.0  0.000000  0.000000  0.000000  0.0
Nordic_walking     0.000000  0.0  0.0  0.0  0.000000  0.000000  0.000000  0.0
cycling            0.000000  0.0  0.0  0.0  0.000000  0.147826  0.000000  0.0
running            0.242424  0.0  0.0  0.0  0.000000  0.000000  0.000000  0.0
rope_jumping       0.757576  1.0  1.0  0.0  0.000000  0.000000  0.000000  0.0

Cluster Label       35
Activity Type
lying              0.0
sitting            0.0
standing           0.0
ironing            0.0
vacuum_cleaning    0.0
ascending_stairs   0.0
descending_stairs  0.0
walking            0.0
Nordic_walking     0.0
cycling            0.0
running            1.0
rope_jumping       0.0
```

[78]:
```
def accuracy(x, y):
    x_labels = pd.DataFrame(x).copy()
    x_labels["activity_name"] = le.inverse_transform(y)
    x_labels["labels"] = clf.predict(x)
    x_labels["predicted_activity"] = x_labels.labels.apply(
        lambda x: xc[[x]].idxmax().values[0]
    )
    print(
        len(x_labels[x_labels.activity_name == x_labels.predicted_activity])
        / len(x_labels)
    )
    return x_labels
```

[76]:
```
print("Validation accuracy for Clustering:")
xval_lab=accuracy(x_val,y_val)
```

```
Validation accuracy for Clustering:
0.7189083820662768
```

Logistic Regressionmodel is aldo trained using the 48 features to see if this performs better.

[74]:
```
df_lr = log_reg(cluster_pred, "normal", "")
copy(df_lr)
```

```
Feature size: 9
Validation accuracy for LR:
```

Validation Accuracy For Logistic Regression Multiple Activity Classification

	validation_accuracy	f1	lambda
0	0.654386	0.603246	0.1
1	0.654386	0.603246	0.2
2	0.654386	0.603246	0.3
3	0.654386	0.603246	0.4
4	0.654386	0.603246	0.5
5	0.654386	0.603246	0.6
6	0.654386	0.603246	0.7
7	0.654386	0.603246	0.8
8	0.654386	0.603246	0.9
9	0.654386	0.603246	1
10	0.654386	0.603246	1.1
11	0.654386	0.603246	1.2
12	0.654386	0.603246	1.3
13	0.654386	0.603246	1.4
14	0.654386	0.603246	1.5
15	0.654386	0.603246	1.6
16	0.654386	0.603246	1.7
17	0.654386	0.603246	1.8

	validation_accuracy	f1	lambda
18	0.654386	0.603246	1.9

Since the validation accuracy of our Logistic Regression model is lesser than that of the clustering model, Clustering is choosen as the final model which will be evaluated on the test set.

The results of the final model on the testing set are printed below

[79]:
```
print("Testing Accuracy")
clust_test = accuracy(x_test, y_test)
clust_test["id"] = clean_data_feats[clean_data_feats.id.isin([107, 108])].id
for subj in [107, 108]:
    subj_df = clust_test[clust_test.id == subj]
    act_freq_predicted = subj_df.predicted_activity.value_counts()
    act_freq_actual = subj_df.activity_name.value_counts()
    print(f"For subject {subj}")
    for i in subj_df.activity_name.unique():
        print(f"Time spent {i} (predicted) : {act_freq_predicted[i]} seconds")
        print(f"Time spent {i} (actual) : {act_freq_actual[i]} seconds")
```

```
Testing Accuracy
0.6151668351870576
For subject 107
Time spent lying (predicted) : 245 seconds
Time spent lying (actual) : 254 seconds
Time spent sitting (predicted) : 385 seconds
Time spent sitting (actual) : 123 seconds
Time spent standing (predicted) : 129 seconds
Time spent standing (actual) : 257 seconds
Time spent ironing (predicted) : 334 seconds
Time spent ironing (actual) : 295 seconds
Time spent vacuum_cleaning (predicted) : 128 seconds
Time spent vacuum_cleaning (actual) : 216 seconds
Time spent ascending_stairs (predicted) : 142 seconds
Time spent ascending_stairs (actual) : 176 seconds
Time spent descending_stairs (predicted) : 121 seconds
Time spent descending_stairs (actual) : 117 seconds
Time spent walking (predicted) : 361 seconds
Time spent walking (actual) : 337 seconds
Time spent Nordic_walking (predicted) : 312 seconds
Time spent Nordic_walking (actual) : 287 seconds
Time spent cycling (predicted) : 137 seconds
Time spent cycling (actual) : 227 seconds
Time spent running (predicted) : 31 seconds
Time spent running (actual) : 37 seconds
For subject 108
Time spent lying (predicted) : 235 seconds
```

```
Time spent lying (actual) : 240 seconds
Time spent sitting (predicted) : 259 seconds
Time spent sitting (actual) : 229 seconds
Time spent standing (predicted) : 259 seconds
Time spent standing (actual) : 251 seconds
Time spent ironing (predicted) : 21 seconds
Time spent ironing (actual) : 330 seconds
Time spent vacuum_cleaning (predicted) : 546 seconds
Time spent vacuum_cleaning (actual) : 243 seconds
Time spent ascending_stairs (predicted) : 59 seconds
Time spent ascending_stairs (actual) : 117 seconds
Time spent descending_stairs (predicted) : 178 seconds
Time spent descending_stairs (actual) : 97 seconds
Time spent walking (predicted) : 377 seconds
Time spent walking (actual) : 315 seconds
Time spent cycling (predicted) : 170 seconds
Time spent cycling (actual) : 255 seconds
Time spent Nordic_walking (predicted) : 302 seconds
Time spent Nordic_walking (actual) : 289 seconds
Time spent running (predicted) : 145 seconds
Time spent running (actual) : 165 seconds
Time spent rope_jumping (predicted) : 68 seconds
Time spent rope_jumping (actual) : 88 seconds
```

[80]:
```python
print("Features used: ")
print(best_feats)
```

```
Features used:
['chest_3D_magnetometer_y_roll_median', 'chest_temperature',
'chest_3D_acceleration_16_z_roll_median',
'ankle_3D_magnetometer_z_spectral_centroid',
'chest_3D_acceleration_16_y_roll_var', 'ankle_3D_magnetometer_x_roll_mean',
'hand_3D_acceleration_16_y_roll_var', 'chest_3D_gyroscope_y_roll_mean',
'chest_3D_acceleration_16_z_roll_var', 'ankle_3D_acceleration_16_x_roll_mean',
'ankle_3D_gyroscope_y_roll_mean', 'ankle_3D_gyroscope_z_roll_var',
'ankle_3D_acceleration_16_y_roll_median',
'chest_3D_acceleration_16_z_roll_mean',
'hand_3D_acceleration_16_x_spectral_centroid',
'ankle_3D_magnetometer_z_roll_mean', 'hand_temperature',
'hand_temperature_roll_mean', 'heart_rate_roll_median',
'ankle_3D_magnetometer_x', 'ankle_3D_magnetometer_z',
'ankle_3D_acceleration_16_x_roll_median', 'chest_3D_gyroscope_x_roll_var',
'hand_3D_acceleration_16_x_roll_mean', 'chest_3D_magnetometer_y_roll_mean',
'ankle_3D_magnetometer_z_roll_median', 'ankle_3D_magnetometer_x_roll_median',
'ankle_3D_acceleration_16_x_roll_var', 'chest_3D_gyroscope_y_roll_median',
'ankle_3D_gyroscope_x_roll_mean', 'hand_3D_acceleration_16_z_spectral_centroid']
```

3 Conclusion

1. It is relatively easy to predict if a subject is lying or not.
2. Time domain features like mean,median and variance and frequency domain features like spectral centroid computed over a sliding window are useful features for performing classification.
3. Metrics such as precision score can be very useful to select features to feed into classifier for the task of activity prediction.

4 References

1. Parkka, J., Ermes, M., Korpipaa, P., Mantyjarvi, J., Peltola, J. and Korhonen, I., 2006. Activity classification using realistic data from wearable sensors. IEEE Transactions on information technology in bio
2. Ermes, M., Parkka, J. and Cluitmans, L., 2008, August. Advancing from offline to online activity recognition with wearable sensors. In 2008 30th annual international conference of the ieee engineering in medicine and biology society (pp. 4451-4454). IEEE.
3. Huynh, T. and Schiele, B., 2005, October. Analyzing features for activity recognition. In Proceedings of the 2005 joint conference on Smart objects and ambient intelligence: innovative context-aware services: usages and technologies (pp. 159-163).


```
[ ]:
```