K-means Clustering of Fitbit Heart Rate Data

Daniela del Río

Tuesday, December 14, 2021

1 Introduction

We are currently amid the covid-19 pandemic which has taken the lives of a total of 5.3 million people based on data published by John Hopkins (updated December 13, 2021) [1]. Nevertheless, the World Health Organization reports a total of 17.9 million deaths just in 2019 due to cardiovascular diseases [2]. That is, cardiovascular diseases cause at least 3 times more deaths in a single year than the total of covid deaths. If the world stopped because of the covid-19 pandemic, we should be taking at least a moment to address the public health crisis caused by cardiovascular diseases. Another relevant factor besides mortality, is the reversibility of cardiovascular diseases (if not caused by genetic factors). The reversibility nature of these diseases indicates that they are preventable because they are related with our lifestyle, for example, amount of exercise and diet.

Currently there are several companies which sell wearable technology for recording biosignals, like heart frequency. For example, Fitbit is a company owned by Google [3] which creates wearable devices for tracking an individual's variables such as: amount of steps, heart rate, amount of sleep time and oxygen saturation. This company allows their users to download from their website their data collected by their device. This opens the possibility of analyzing this data using machine learning techniques. This allows the extraction of valuable information from these recordings. One of such machine learning techniques is K-means, which will be further discussed below.

K-means is a clustering technique based on the euclidean distance between its elements. The objective of this clustering technique is minimizing the cost function J, and that is achieved by minimizing the distance between the cluster centers $\bar{\mu}_k$ and the datapoints $\bar{x}_n$. The entries r_{nk} are values of 1 if the n-th datapoint is in the k-th cluster and if not, 0. The cost function is defined by:

$$J = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \|\bar{x}_n - \bar{\mu}_k\|^2 \tag{1}$$

The way we minimize the cost function is by iterating between assigning r_{nk} values and recalculating the cluster centers $\bar{\mu}_k$. We start by choosing random cluster centers, then we assign r_{nk} values and calculate the cost function. This is iterated and in each step, the cost function must decrease. The convergence criterion used is that the change of J should be small [4, pg. 348-353].

The author speculates that implementing the Fitbit as a device that helps monitor your health could potentially reduce the incidence of cardiovascular diseases. Thus, the motivation for this work in analyzing the Fitbit data with K-means clustering.

2 Methodology

Opening the Fitbit website from a web browser, the data was downloaded following this sequence: Settings, then Data Export and Export Your Account Archive. The .zip file was uncompressed, then the folder of the account holder was opened, and finally opening the Physical Activity folder. All the files are available as .json files, for example: *heart_rate-2021-08-30.json*.

The notebook was developed in Jupyter and written in Python. It is available in the following Github repository: https://github.com/DanidelRio/Heart. In this link you can also find the recordings of 55 days, nevertheless, the following analysis uses the first 20 recordings unless otherwise specified.

3 Results and Discussion

3.1 Time series

First of all was plotting the heart rate time series. This can be seen for the first recording in figure 1. Notice how the heart frequency oscillates between 50 and 70 until since the beginning of the recording until around 10 am and then it increases, this is probably because of a sleeping or resting period.

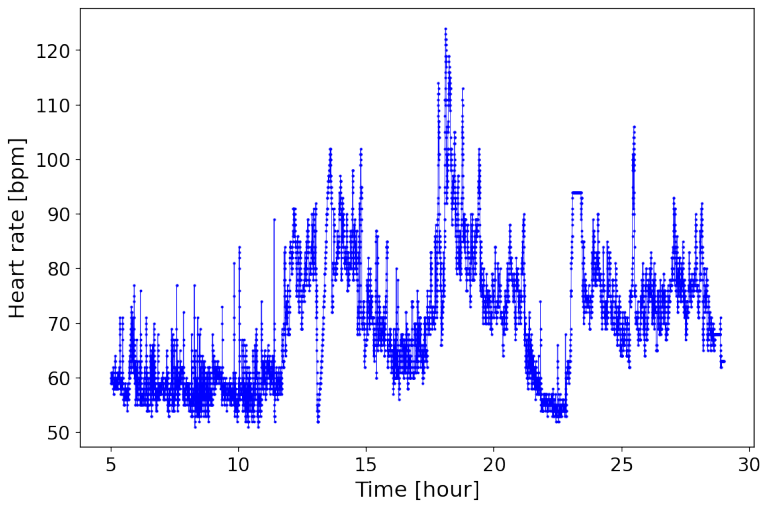


Figure 1: Time series extracted from one day's recording from the Fitbit.

We also concatenated the first 3 consecutive days and plotted each day with a different color. The result is figure 2. Note how there are marked increases and decreases of heart rate throughout the day. The time series we focused on for the rest of the analysis is shown in figure 3 where the first 20 recordings are concatenated.

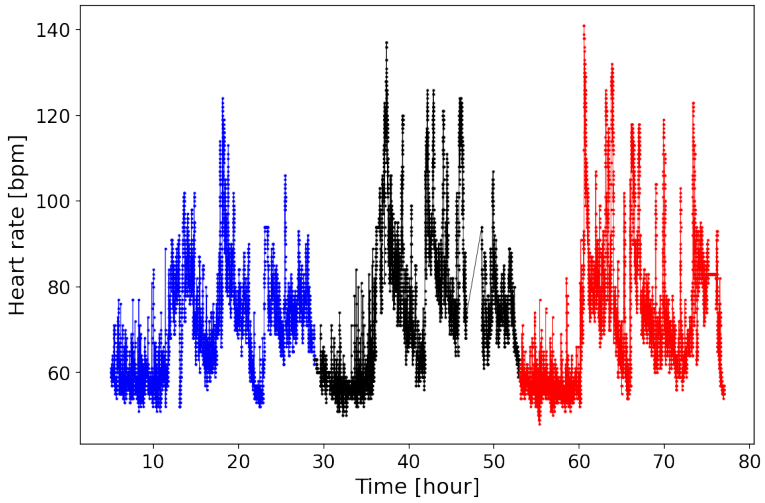


Figure 2: Concatenated time series from 3 days.

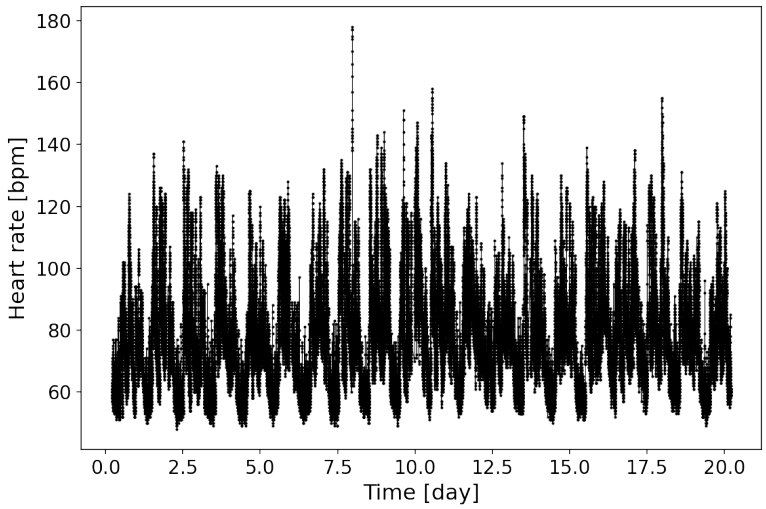


Figure 3: Concatenated time series from 20 days.

3.2 Mean and standard deviation for 5 s and 1 s timebins

The recording was divided in either 5 s or 1 s bins in order to obtain the mean heart rate and standard deviation of this timebin. These two quantities were plotted in figures 4 for the first 3 recordings and 5 for the first 20 recordings. Notice how there is a wider dispersion of points using the 1 s timebin compared to the 5 s timebin. This is expected because we are averaging in a smaller timebin in the 1 s data. Also notice how there are almost no elements with the 5s timebin with a 0 standard deviation.

A relevant remark is that in figures 4 and 5 there is information loss because there is no representation of the density of points in a single location in space. To address this concern, a heatmap was created, see figure 6, which is basically a histogram of figure 5 using the 1s and 5 s timebins. There is also a normalized histogram amplification in figure 7. Notice how most of the points are clustered in a heart rate between 50 and 60 and the standard deviation is between 0 and 2.5. This corresponds to a resting heart frequency, which can be thought of as the period of time when the subject was asleep.

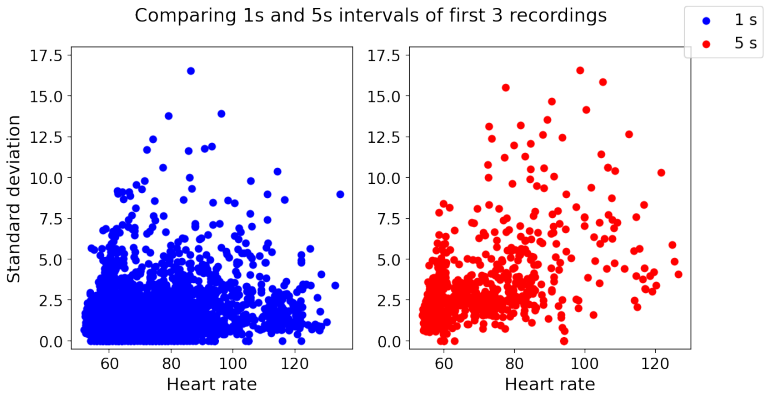


Figure 4: When using only the first 3 recordings, we see a wider dispersion of points using the 1s timebin compared to the 5s timebin.

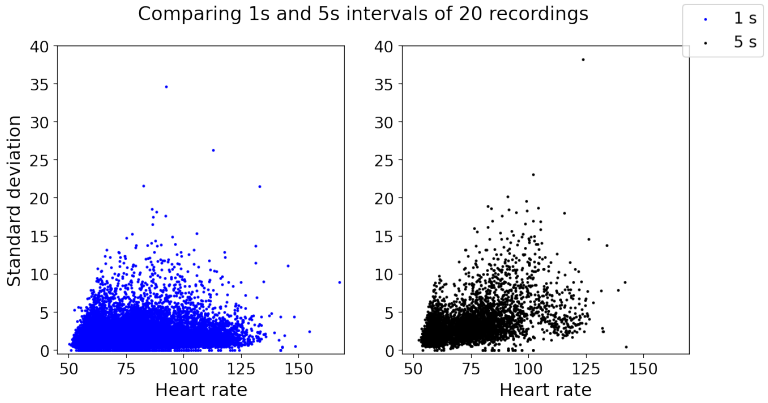


Figure 5: When comparing 20 recordings, we see the same trend as in the first 3 recordings; a wider cloud of points using the 1s timebin compared to the 5 s timebin.

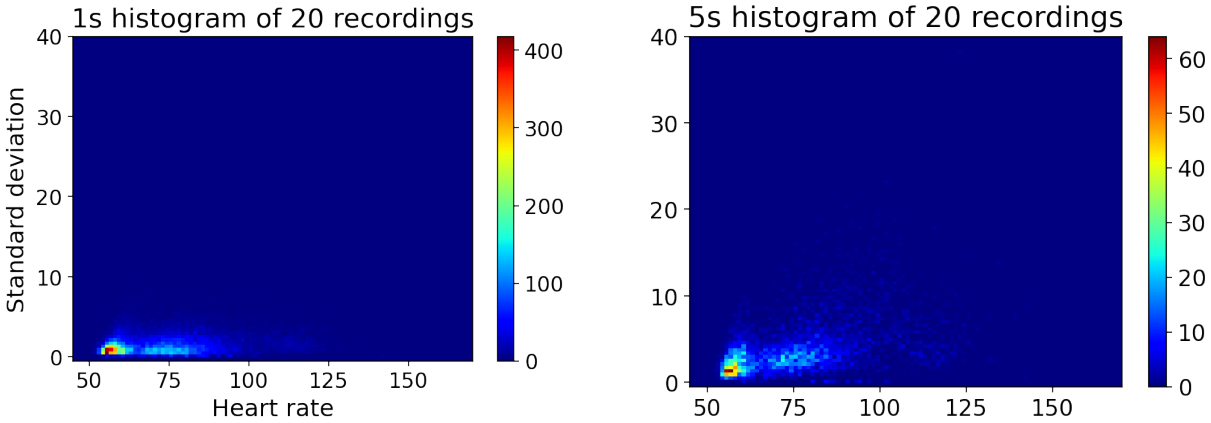


Figure 6: Histogram of the 20 first recordings using 1 s timebins left and 5 s timebins right. The colorbar indicates the number of datapoints. Notice there is a factor of 5 that should be considered in the 5 s histogram to make it comparable to the scale in the 1 s histogram.

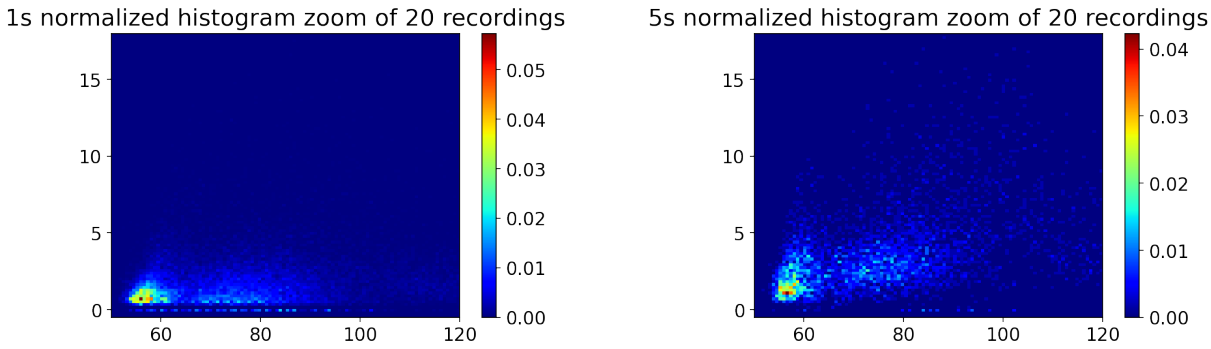


Figure 7: Normalized amplified histogram of the 20 first recordings using 1 s timebins left and 5 s timebins right.

3.3 K means clustering

The K-means clustering algorithm was described in the Introduction. In homework 4, we used this algorithm to minimize the distance of a vector in 1D, specifically voltage. In this case, we have data points $\bar{x}_n$ in 2 dimensions: heart rate and standard deviation. So the functions used in homework 4 were modified for considering these two variables. The K-means clustering algorithm was written in python and was used for obtaining the figures associated with 3 clusters (figure 8) and the cost function (figure 9). For using more than 3 clusters, the `KMeans` function from the sklearn package was used.

Note in figure 8 how the K-means algorithm basically marks vertical lines to separate the clusters, that is, the relevant parameter for clustering is the mean, not the standard deviation. And this can be observed both using the 1 s or the 5 s timebins. The next question one could ask is how many iterations did it take for the algorithm to reach a converging value. This depends on what our converging criteria is, but we can see how the cost function decreases with each iteration until practically having a constant value after 5 iterations in figure 9.

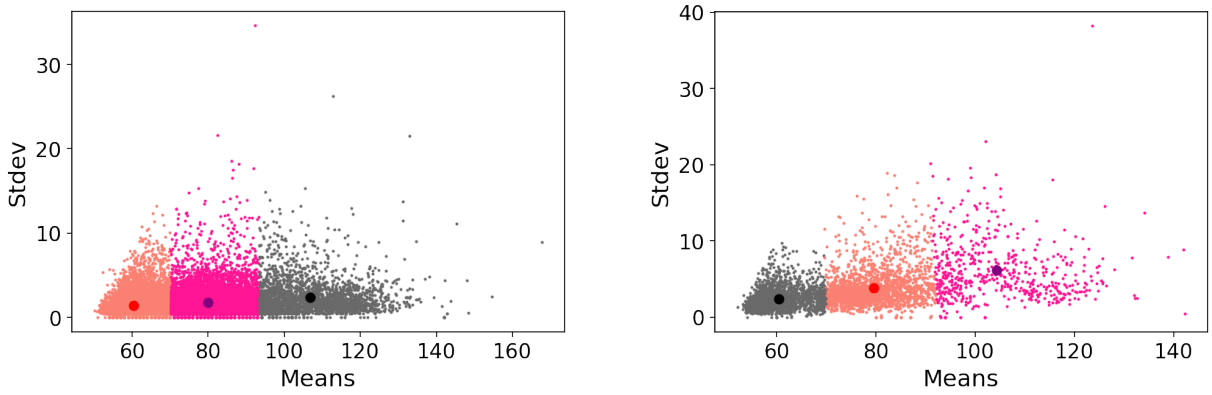


Figure 8: K means clustering of data using 1 s timebins left and 5 s timebins right using 3 clusters.

Now one could ask, what is the optimal number of clusters? This is a tricky question because we want enough clusters to group our data, but if we use too many, it will be harder to analyze even though the cost function will be lower. So we need to find the correct

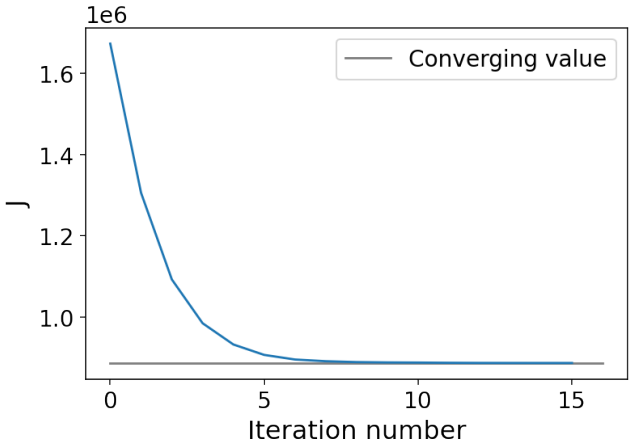


Figure 9: Number of iterations for algorithm to converge using 1 s timebins and 3 clusters.

balance. This balance is obtained when the change of the cost function (that is, the first derivative) changes by a small fraction of its value. In figure 10, one can visually appreciate this change occurs around 4 or 5 clusters, both using 1 s or 5 s timebins. Nevertheless, this can be confirmed in figure 11 where the derivative of the cost function is plotted. There we can see how the elbow of the change of the cost function is small with 4 or 5 clusters. So we clustered the data in these 4 (figure 12) or 5 (figure 13) clusters. Note in these figures how the relevant clustering parameter is the mean heart frequency instead of the standard deviation. This is because the clusters have very well defined vertical separations between them.

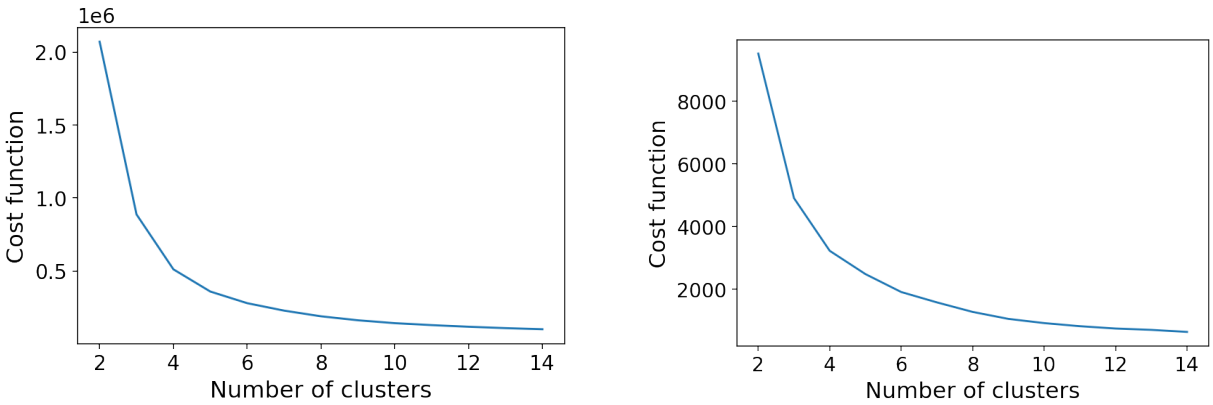


Figure 10: Left cost function for 1 s timebin data and right for the 5 s timebin data.

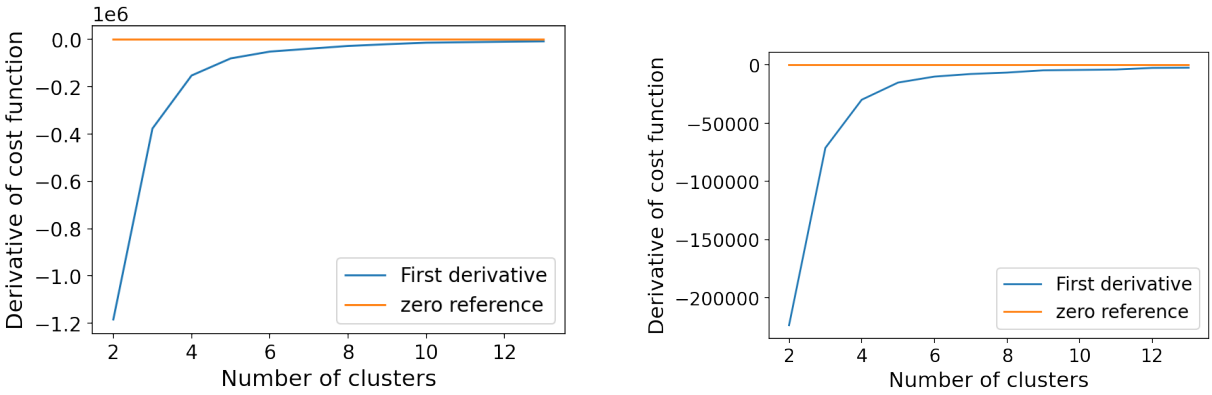


Figure 11: Convergence criteria, left for 1 s and right for 5 s. We plot the first derivative of the cost function. Note how in 4 or 5 clusters, the distance to 0 is small, that is the cost function's change due to number of clusters is close to zero.

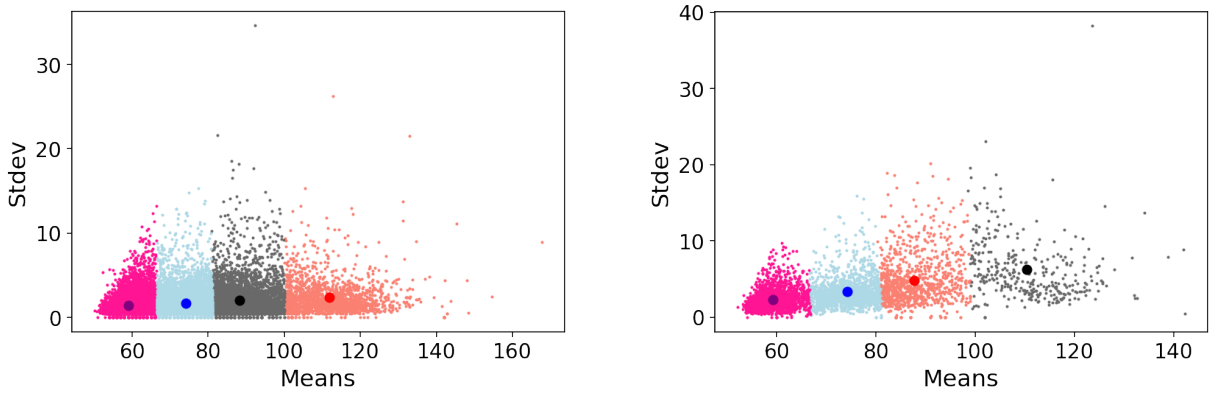


Figure 12: K means clustering of data using 1 s timebins left and 5 s timebins right using 4 clusters.

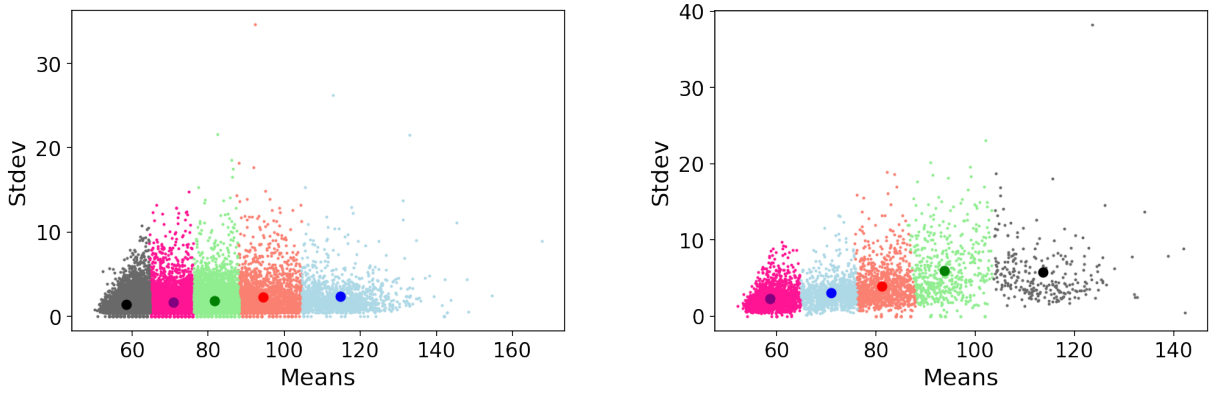


Figure 13: K means clustering of data using 1 s timebins left and 5 s timebins right using 5 clusters.

3.3.1 Clusters based on time-of-day labels

The amount of physical activity we perform during the day changes depending on the time. As the dataset had no associated labels, we created some based on the time of day. The

underlying idea being that physical activity is related with the mean heart frequency. The data was clustered based on this labels, thus obtaining figure 14 and 15. Even though the best value for k found in the previous section was 4 or 5 clusters, in this section there were 6 labels because we wanted to find if there was a distinctive pattern.

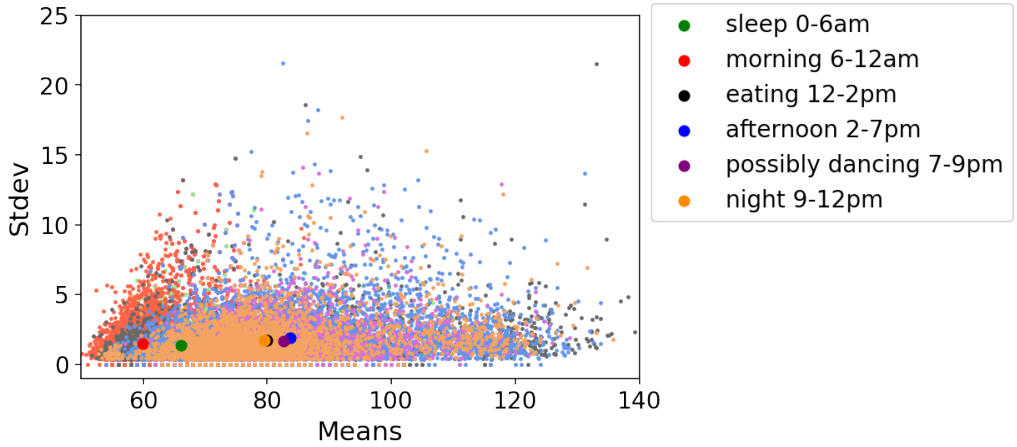


Figure 14: Clustered data based on labels which indicate the time of day using the 1 s timebin.

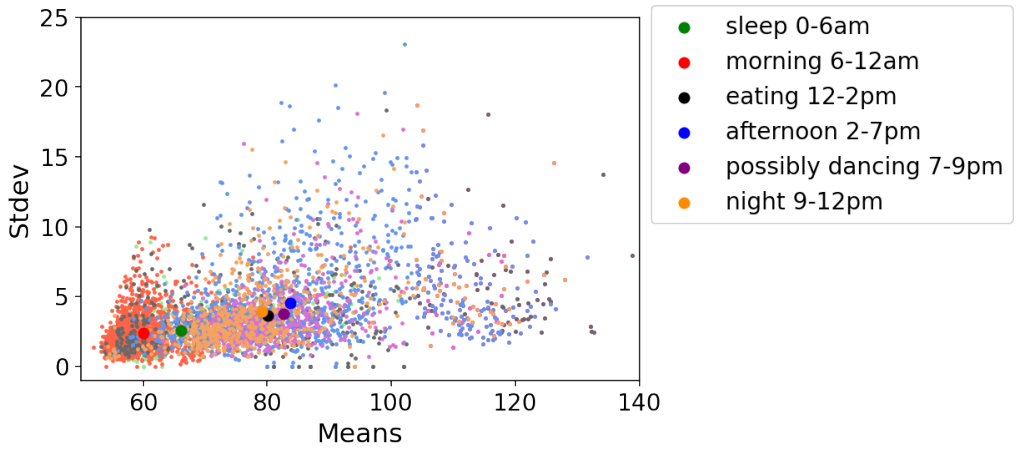


Figure 15: Clustered data based on labels which indicate the time of day using the 5 s timebin.

At least from the previous figures, we cannot say there is a clear pattern associated with the labels. It seems the labels are not separating adequately the data because there are no clearly marked clusters. The only possibly distinctive cluster is the morning cluster. But this is not the best visualizing technique because of overlap, problem we ran into before and thus created the heatmaps.

4 Future work

In the json files downloaded from Fitbit, there is a variable called *confidence*. This is an integer value associated to each data sample ranging from 0 to 3. Nevertheless, in none

of the documents downloaded or in the website was it possible to find an explanation to this variable. So the Fitbit support was contacted and after some email exchange, they replied with an answer. As seen in figure 16, "Confidence scores indicate confidence in the accuracy of the captured data with "3" being the highest and "0" being the lowest." With this confidence score, the heart rate time series could be weighted, giving more relevance or weight to entries with the highest confidence scores and low weight to entries with zeros.

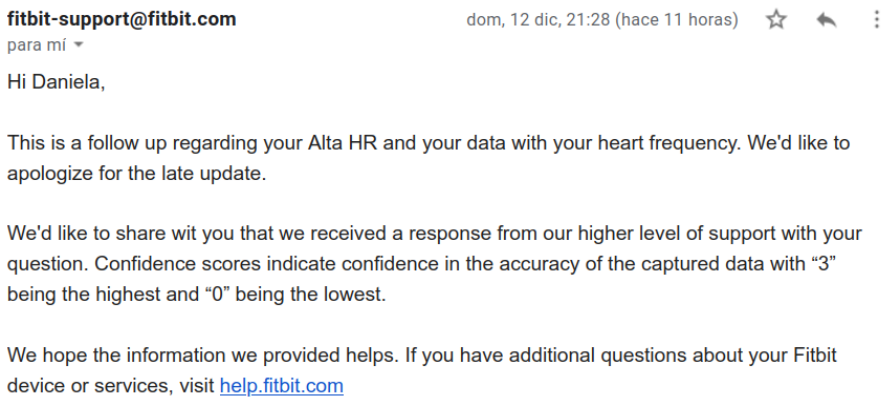


Figure 16: Fitbit's reply to question about confidence scores.

Another aspect we missed is there are 2 files in the Physical Activity folder downloaded from the Fitbit website called: *steps-2021-08-29.json* and *distance-2021-08-29.json*; which contain the number of steps per minute and the distance value per minute per day. This would have been useful because now we could define some intervals of physical activity intensity based on the values of these files. Even though there is no reference to the units of the distance file.

Another idea was to cluster the data using other clustering techniques, such as gaussian mixtures or principal components analysis. Citing Kevin Murphy, "Since K-means is not a proper EM algorithm, it is not maximizing likelihood. Instead, it can be interpreted as a greedy algorithm for approximately minimizing a loss function related to data compression."[4]. As we are using a time series, it is time-dependent, so we could use a Kalman filter or a hidden Markov model [5] to predict future states based on the past history. A potential application of this would be predicting a heart stroke. We did not see it in class, but researchers have even proposed an agent based model for clustering [6]. Using another clustering technique would allow the comparison between it and K-means.

We are only using the first 20 recordings, but this could be extended to using the 55 of them. A detail that was ran into, but not looked further was that there seems to be no mean heart rate in recording 25.

An idea for improving the labels is to label the heart rate data based on the physical activity performed at that time of day. For example:

1. Sleeping.
2. Non vigorous awake activity, like taking a class, or sitting at a desk.
3. Walking.
4. Riding a bike or dancing.

But for keeping this registrar of physical activity, one should be truly committed to do this rigorously.

For a better visualization of the labeled data, we could create a histogram for each label. This could help see the distribution of the datapoints.

5 Conclusions

When comparing 1 s to 5 s timebins, there was a wider dispersion in the 1 s timebins. This was expected because the we are averaging over a wider time period in the 5 s timebin.

We found the optimal number of clusters was 4 or 5 when using timebins of 1 s or 5 s. Nevertheless, other groups have used K-means as well and found the optimal number of clusters is 2 [7].

The most relevant parameter for k-means clustering between mean heart frequency and its standard deviation is heart frequency. This was visualized by the vertical lines which divided one cluster from another.

Finally, it is of utter importance to have an adequate data labeling which must be related to a relevant variable, such as physical activity in this case.

6 References

[1] E. Dong, H. Du, and L. Gardner. *An Interactive Web-Based Dashboard to Track COVID-19 in Real Time*. https://doi.org/10.1016/S1473-3099(20)30120-1. Accessed December 13, 2021. 2020.

[2] World Health Organization. *Cardiovascular diseases (CVDs)*. https://www.who.int/news-room/fact-sheets/detail/cardiovascular-diseases-(cvds). Accessed December 13, 2021. 2017.

[3] A. Williams. *Google Now Owns Fitbit: What It Means For Your Fitness Data Privacy*. https://www.forbes.com/sites/andrewwilliams/2021/01/14/google-now-owns-fitbit-what-it-means-for-your-fitness-data-privacy/?sh=17c9ef4039e1. Accessed December 13, 2021. 2021.

[4] K. P. Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[5] S. Yun et al. "Forecasting of heart rate variability using wrist-worn heart rate monitor based on hidden Markov model". In: *2018 International Conference on Electronics, Information, and Communication (ICEIC)*. IEEE. 2018, pp. 1–2.

[6] M. Bursa and L. Lhotska. "Modified ant colony clustering method in long-term electrocardiogram processing". In: *2007 29th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*. IEEE. 2007, pp. 3249–3252.

[7] W. Materko. "Stratification Fitness Aerobic Based on Heart Rate Variability during Rest by Principal Component Analysis and K-means Clustering." In: *Journal of Exercise Physiology Online* 21.1 (2018).

7 Appendix

The code used for this project is available in https://github.com/DanidelRio/Heart. For completeness, it is added as an appendix in this report.

json_analysis

December 14, 2021

1 K-means clustering of fitbit heart rate data

[1]:
```
import os
import json
import random
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import mixture
from sklearn.cluster import KMeans
from sklearn.utils import shuffle
```

[2]:
```
# commands to create high-resolution figures with large labels
%config InlineBackend.figure_formats = {'png', 'retina'}
plt.rcParams['axes.labelsize'] = 16 # fontsize for figure labels
plt.rcParams['axes.titlesize'] = 18 # fontsize for figure titles
plt.rcParams['font.size'] = 14 # fontsize for figure numbers
plt.rcParams['lines.linewidth'] = 1.4 # line width for plotting
```

[3]:
```
# https://hackanons.com/2020/12/python-extract-data-from-json-file.html
def open_file(file):
#     with open("json_files/"+file) as f:
    with open(file) as f:
        json_data = json.load(f)
    return json_data

# print(json_data)
```

[4]:
```
# First approximation to obtain heart rate data
# Note this does not consider the time axis in a consistent manner.

def extract_hr(json_data):
    heart_rate = np.zeros(len(json_data))

    for i in range(len(json_data)):
        values1 = [json_data[i][k] for k in json_data[i]]
        values2 = [values1[1][k] for k in values1[1]]
```

```
        heart_rate[i] = values2[0]

    return heart_rate
```

For the following, the data was divided in ~5s time bins, the heart rate average was obtained as well as the standard deviation. Then we ploted in 2D and used K-means to cluster the data.

[5]:
```
# Function that separates a string where the separator is

# https://stackoverflow.com/questions/4697006/
 ↪python-split-string-by-list-of-separators
def split(txt, seps):
    default_sep = seps[0]

    # we skip seps[0] because that's the default separator
    for sep in seps[1:]:
        txt = txt.replace(sep, default_sep)
    return [i.strip() for i in txt.split(default_sep)]
```

[6]:
```
# Extract different time formats arrays from json

def splitting_data(json_data):

    dict_keys1 = [k for k in json_data[0]]
    times = [json_data[k][dict_keys1[0]] for k in range(len(json_data))] #␣
 ↪dateTime

    only_time = [split(times[k], ' ')[1] for k in range(len(times))]
    hms = [split(only_time[k], ":") for k in range(len(times))] #hour minute␣
 ↪second

    # The index will represent the second of the day. Total of 86 400 seconds␣
 ↪in one day

    # Converting the time to the number of second of the day
    seconds = [int(hms[k][0])*3600 + int(hms[k][1])*60 + int(hms[k][2]) for k␣
 ↪in range(len(hms))]

    return times, only_time, hms, seconds
```

[7]:
```
# First challenge, divide everything in 5min time bins, that is in 5*60 = 300 s␣
 ↪time bins!
def data_5s(json_data):

    times, only_time, hms, seconds = splitting_data(json_data)
    vecs_sec = []
    vecs_indices = []
```

```
    for i in range(288): # Number of 5 min periods in 24 hours
        vec_sec = []
        vec_indices = []
        banner = False

        for index_i in range(np.argmin(seconds)): # len(indices)

            if i*300 <= seconds[index_i] and seconds[index_i] < (i+1)*300: # 5␣
↪min condition
                vec_sec.append(seconds[index_i])
                vec_indices.append(index_i)
                banner = True

            elif banner == True:
                vecs_sec.append(vec_sec)
                vecs_indices.append(vec_indices)
                break

    return vecs_sec, vecs_indices
```

[8]:
```
# Divide everything in 1min time bins, that is in 60 s time bins!
def data_1s(json_data):

    times, only_time, hms, seconds = splitting_data(json_data)
    vecs_sec = []
    vecs_indices = []

    for i in range(1440): # Number of 1 min periods in 24 hours
        vec_sec = []
        vec_indices = []
        banner = False

        for index_i in range(np.argmin(seconds)): # len(indices)

            if i*60 <= seconds[index_i] and seconds[index_i] < (i+1)*60: #1␣
↪minute condition
                vec_sec.append(seconds[index_i])
                vec_indices.append(index_i)
                banner = True

            elif banner == True:
                vecs_sec.append(vec_sec)
                vecs_indices.append(vec_indices)
                break

    return vecs_sec, vecs_indices
```

[9]:
```
###### Magic number, number of seconds in one day
print(300*288)
print(24*3600)
print(86400/60)
print(288*5)
```

```
86400
86400
1440.0
1440
```

[10]:
```
# Obtain the heart rate mean of these intervals as well as their standard␣
↪deviation

def obtain_data(file_name):
    json_data = open_file(file_name)

    vecs_secs_5s, vecs_indices_5s = data_5s(json_data)
    vecs_secs_1s, vecs_indices_1s = data_1s(json_data)

    # For obtaining the time series
    times, only_time, hms, seconds = splitting_data(json_data) # Seconds has␣
↪the correct x-entries for the heart rate time series
    heart_rate = np.zeros(len(json_data))

    for i in range(len(json_data)):
        values1 = [json_data[i][k] for k in json_data[i]] # Dictionary keys␣
↪from the json files
        values2 = [values1[1][k] for k in values1[1]]
        heart_rate[i] = values2[0]

    # For obtaining the 5s data
    means_5s  = np.zeros(len(vecs_indices_5s))
    stdevs_5s = np.zeros(len(vecs_indices_5s))
    means_1s  = np.zeros(len(vecs_indices_1s))
    stdevs_1s = np.zeros(len(vecs_indices_1s))

    for i in range(len(vecs_indices_1s)):
        means_1s[i] = np.mean(heart_rate[vecs_indices_1s[i]])
        stdevs_1s[i] = np.std(heart_rate[vecs_indices_1s[i]])
        if (i+1)%5 == 0: #Each 5 seconds
            means_5s[round(i/5)-1] = np.mean(heart_rate[vecs_indices_5s[round(i/
↪5)-1]]) # Start offset
            stdevs_5s[round(i/5)-1] = np.std(heart_rate[vecs_indices_5s[round(i/
↪5)-1]])

    times = np.asarray(times)
```

```
    only_time = np.asarray(only_time)
    hms = np.asarray(hms)
    seconds = np.asarray(seconds)
    vecs_secs_5s = np.asarray(vecs_secs_5s)
    vecs_secs_1s = np.asarray(vecs_secs_1s)

    return  times, only_time, hms, seconds, vecs_secs_5s, heart_rate, means_5s,␣
 ↪stdevs_5s, means_1s, stdevs_1s, vecs_secs_1s
```

[11]:
```
# json_data = open_file("json_files/heart_rate-2021-08-31.json")

# vecs_secs_5s, vecs_indices_5s = data_5s(json_data)
# vecs_secs_1s, vecs_indices_1s = data_1s(json_data) # Smething is wrong5
# print(len(vecs_secs_5s))
# print(len(vecs_secs_1s))
```

[12]:
```
# How to extract all the .json files from a folder

Folder = "json_files/" #Folder where the original files are
recordings_path = []

for TheFile in sorted(os.listdir(Folder)):
    TheFileName, TheFileExtension = os.path.splitext(TheFile) # breaks file␣
 ↪name into pieces based on periods

    InputFilePath = Folder + TheFileName + TheFileExtension # Full path to file

    if (TheFileExtension==".json"): # Only interested in .json files
        recordings_path.append(InputFilePath)

# recordings_path  # They are NOW in sequential order
```

[13]:
```
# Object
class Recording:
  def __init__(self, variables):
    # times, only_time, hms, seconds, heart_rate, means, stdevs
    self.times = variables[0]
    self.only_time = variables[1]
    self.hms = variables[2]
    self.seconds = variables[3]
    self.vecs_secs_5s = variables[4]
    self.heart_rate = variables[5]
    self.means_5s = variables[6]
    self.stdevs_5s = variables[7]
    self.means_1s = variables[8]
    self.stdevs_1s = variables[9]
    self.vecs_secs_1s = variables[10]
```

[14]:
```
# json_data = open_file('json_files/heart_rate-2021-08-31.json')

# vecs_secs_5s, vecs_indices_5s = data_5s(json_data)
# vecs_secs_1s, vecs_indices_1s = data_1s(json_data)

# print(len(vecs_indices_5s))
# print(len(vecs_indices_1s))
# print(len(vecs_indices_1s)/len(vecs_indices_5s))
```

[15]:
```
recordings = []

for i in range(0, 21):#len(recordings_path)):
    variables = obtain_data(recordings_path[i])
    recordings.append(Recording(variables))
    print("Finished recording ", i+1)
```

```
Finished recording  1
Finished recording  2
Finished recording  3
Finished recording  4
Finished recording  5
Finished recording  6
Finished recording  7
Finished recording  8
Finished recording  9
Finished recording  10
Finished recording  11
Finished recording  12
Finished recording  13
Finished recording  14
Finished recording  15
Finished recording  16
Finished recording  17
Finished recording  18
Finished recording  19
Finished recording  20
Finished recording  21
```

[16]:
```
# when it reaches midnight, a new day starts, but the plot just starts back at␣
 ↪0 instead of continuing to hour 25.
# Good to know, we are handling data from two days. Not everything in this file␣
 ↪corresponds to the day 8/31/2021

# When does midnight occur for the different recordings
for i in range(len(recordings)):
    print(i, " ", np.argmin(recordings[i].seconds))
```

```
0    7900
1    7375
2    8297
3    8304
4    7690
5    7555
6    7900
7    8328
8    8363
9    7285
10    8155
11    8096
12    8323
13    7217
14    7834
15    7939
16    8227
17    7833
18    8219
19    7887
20    7795
```

[17]:
```
# Problem with recordings[25]. No means data :/
#Algo pasó con heart_rate-2021-09-25.json
# print(np.argmin(recordings[25].seconds))
# print(recordings_path[25])
# print(recordings[25].means)
```

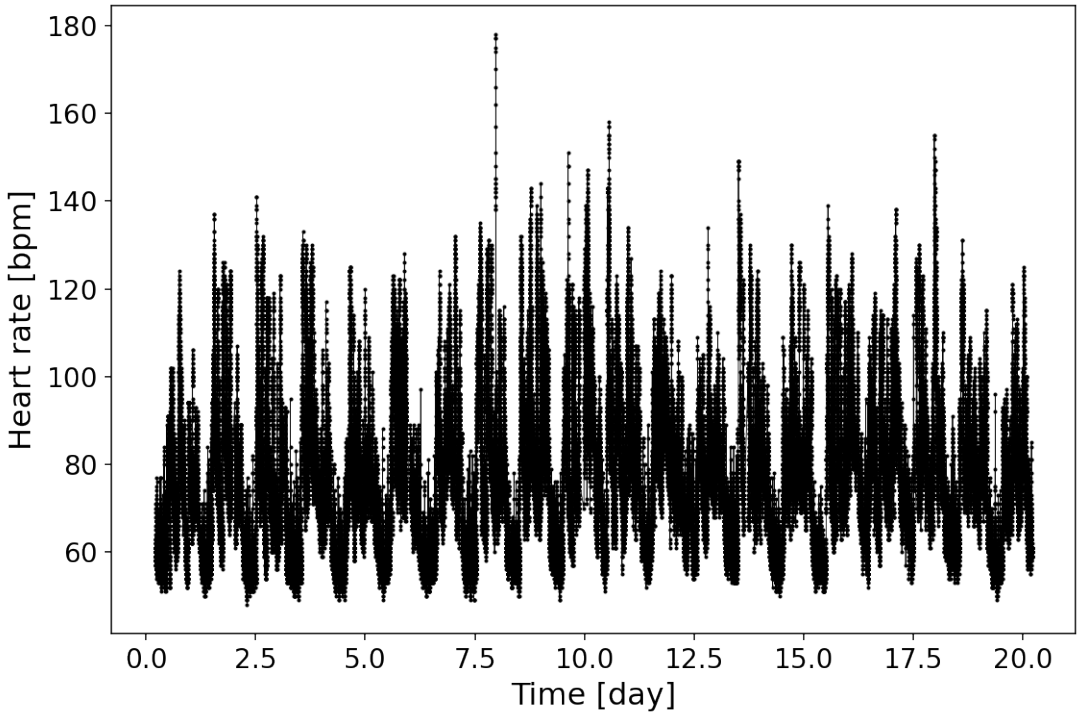
2 Generating a big array by concatenating multiple arrays.

[18]:
```
# Shift the data from ONE day
def shift(recording):
    # all_seconds should be the same length as all_heart_rate
    recording_i = 0
    secs = recording.seconds

    # Correcting the number of seconds for midnight and restarting the number␣
 ↪of seconds

    minimum_that_day = np.argmin(secs) #When midnight occurs
    shifted_seconds = secs

    # BEcause it actually corresponds to the next day 24*3600 = 86400
    shifted_seconds[minimum_that_day:] = recording.seconds[minimum_that_day:] +␣
 ↪86400
```

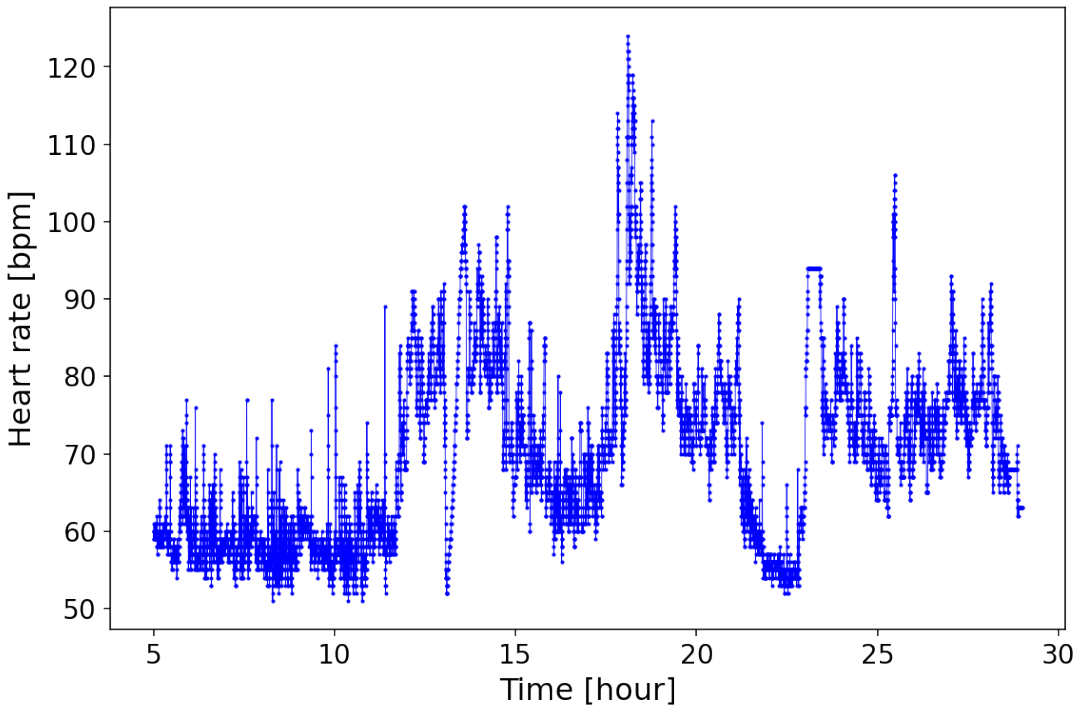
```
    return np.asarray(shifted_seconds)
```

[19]:
```
# Now, we want to concatenate the previous recordings

n = 20
all_seconds = []
all_heart_rate = []
all_heart_rate = np.concatenate([recordings[i].heart_rate for i in range(n)])␣
↪#All data, not the 5 min bin ones

for recording_i in range(n): #Total number of recording for concatenation
    all_seconds.append(shift(recordings[recording_i])+86400*recording_i)

all_seconds = np.asarray(np.concatenate(all_seconds))

all_means_5s = np.concatenate([recordings[i].means_5s for i in range(n)]) # The␣
↪5 min bin ones
all_stdevs_5s = np.concatenate([recordings[i].stdevs_5s for i in range(n)])␣
↪#The 5 min bin ones
all_vecs_secs_5s = np.concatenate([recordings[i].vecs_secs_5s for i in␣
↪range(n)]) # The 5 min bin ones

all_means_1s = np.concatenate([recordings[i].means_1s for i in range(n)]) # The␣
↪1 min bin ones
all_stdevs_1s = np.concatenate([recordings[i].stdevs_1s for i in range(n)])␣
↪#The 1 min bin ones
all_vecs_secs_1s = np.concatenate([recordings[i].vecs_secs_1s for i in␣
↪range(n)])
```

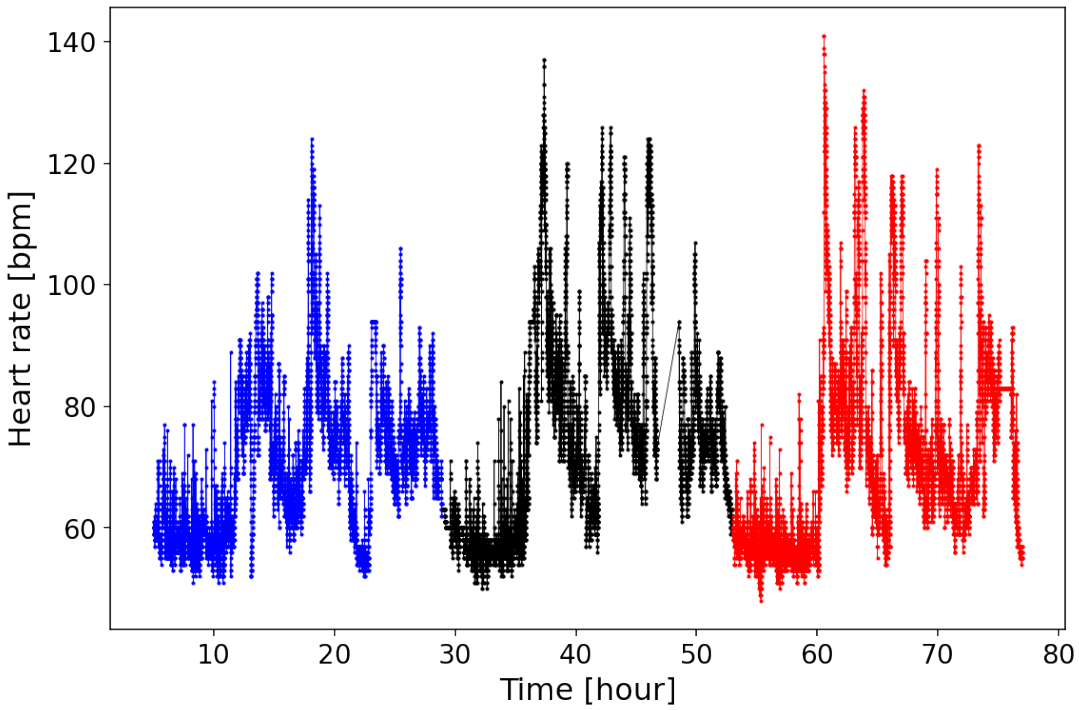
[20]:
```
plt.figure(figsize= (9,6))

plt.plot(all_seconds/86400, all_heart_rate, c="k", marker=".", linewidth=0.4,␣
↪markersize=2)
plt.xlabel("Time [day]")
plt.ylabel("Heart rate [bpm]")
plt.show()
```



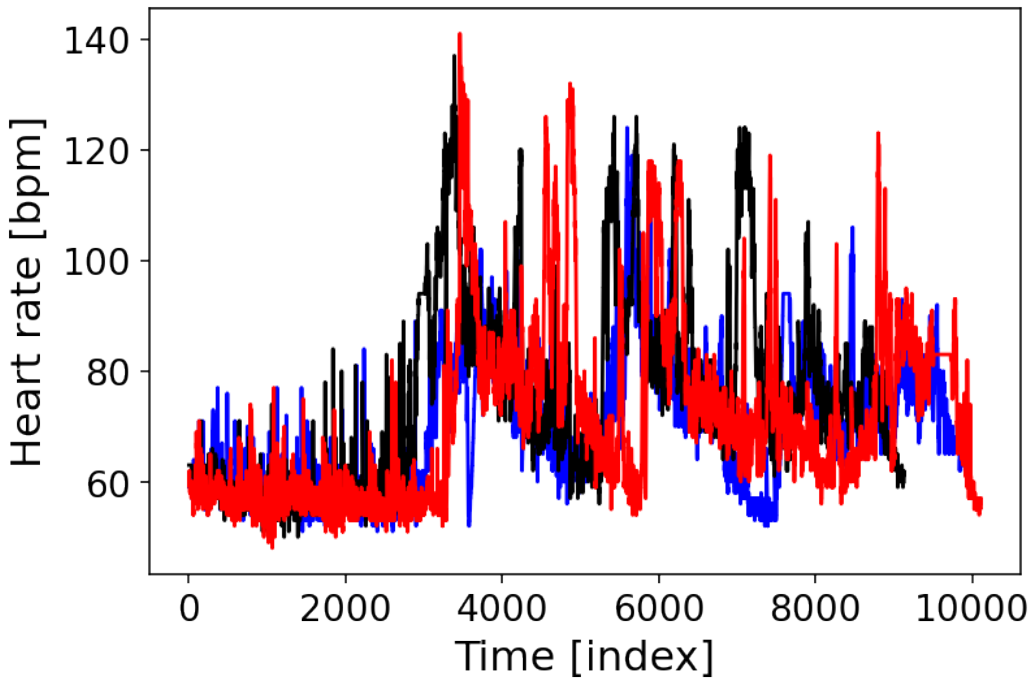
[21]:
```
couleur = ["b", "k", "r"]
plt.figure(figsize= (9,6))

plt.plot((recordings[0].seconds)/3600, recordings[0].heart_rate, c="b",␣
 ↪marker=".", linewidth=0.4, markersize=2)
plt.xlabel("Time [hour]")
plt.ylabel("Heart rate [bpm]")
plt.show()
```



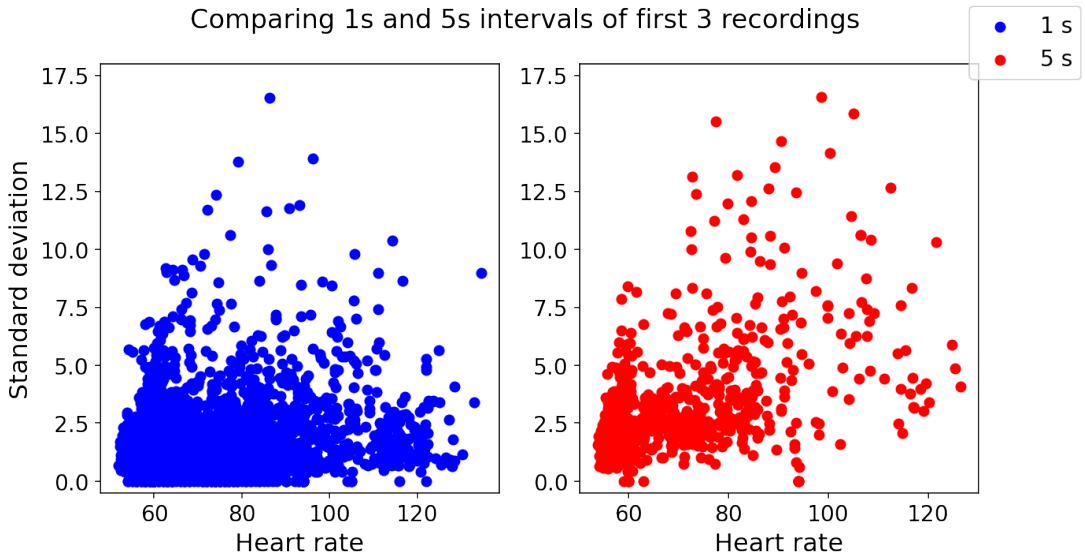
[22]:
```
# Only 3 sequential recordings
couleur = ["b", "k", "r"]
plt.figure(figsize= (9,6))
for i in range(3):
    plt.plot((recordings[i].seconds+86400*i)/3600, recordings[i].heart_rate,␣
 ↪c=couleur[i], marker=".", linewidth=0.4, markersize=2)
plt.xlabel("Time [hour]")
plt.ylabel("Heart rate [bpm]")
plt.show()
```



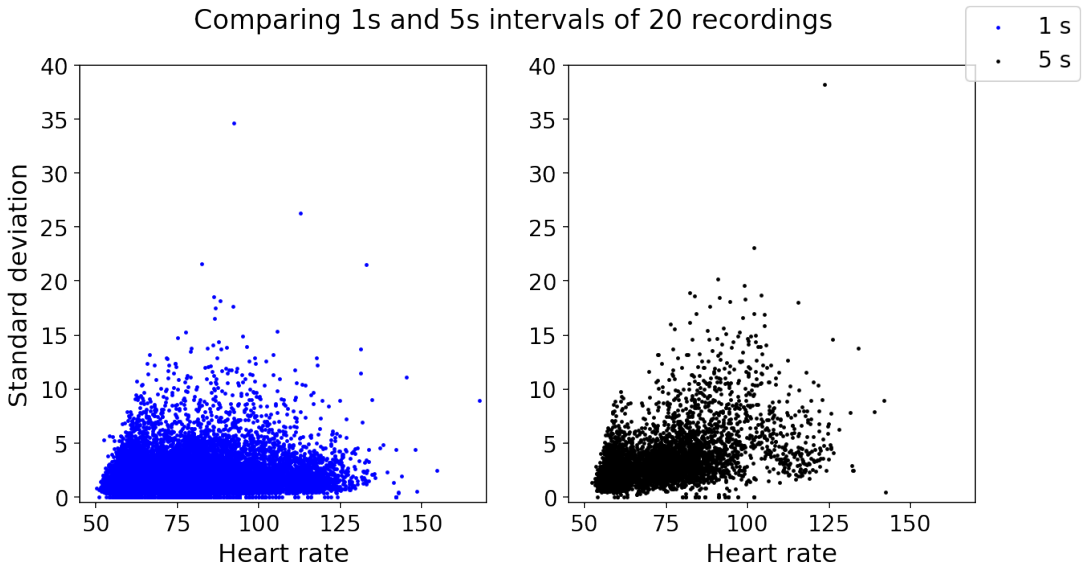
[23]:
```
plt.plot(recordings[0].heart_rate, c="b")
plt.plot(recordings[1].heart_rate, c="k")
plt.plot(recordings[2].heart_rate, c="r")
plt.xlabel("Time [index]")
plt.ylabel("Heart rate [bpm]")
plt.show()
```



[24]:
```
f, axs = plt.subplots(1,2,figsize=(10,5))
f.suptitle('Comparing 1s and 5s intervals of first 3 recordings')

axs[0].scatter(recordings[0].means_1s, recordings[0].stdevs_1s, c = "b", label␣
 ↪= '1 s')
axs[0].scatter(recordings[1].means_1s, recordings[1].stdevs_1s, c="b")
axs[0].scatter(recordings[2].means_1s, recordings[2].stdevs_1s, c="b")
axs[0].set_xlabel("Heart rate")
axs[0].set_ylabel("Standard deviation") # Heart rate variability
axs[0].set_ylim([-0.5, 18])

axs[1].scatter(recordings[0].means_5s, recordings[0].stdevs_5s, c = "r",␣
 ↪label="5 s")
axs[1].scatter(recordings[1].means_5s, recordings[1].stdevs_5s, c="r")
axs[1].scatter(recordings[2].means_5s, recordings[2].stdevs_5s, c="r")
axs[1].set_xlabel("Heart rate")
axs[1].set_ylim([-0.5, 18])

f.legend()
plt.show()
```

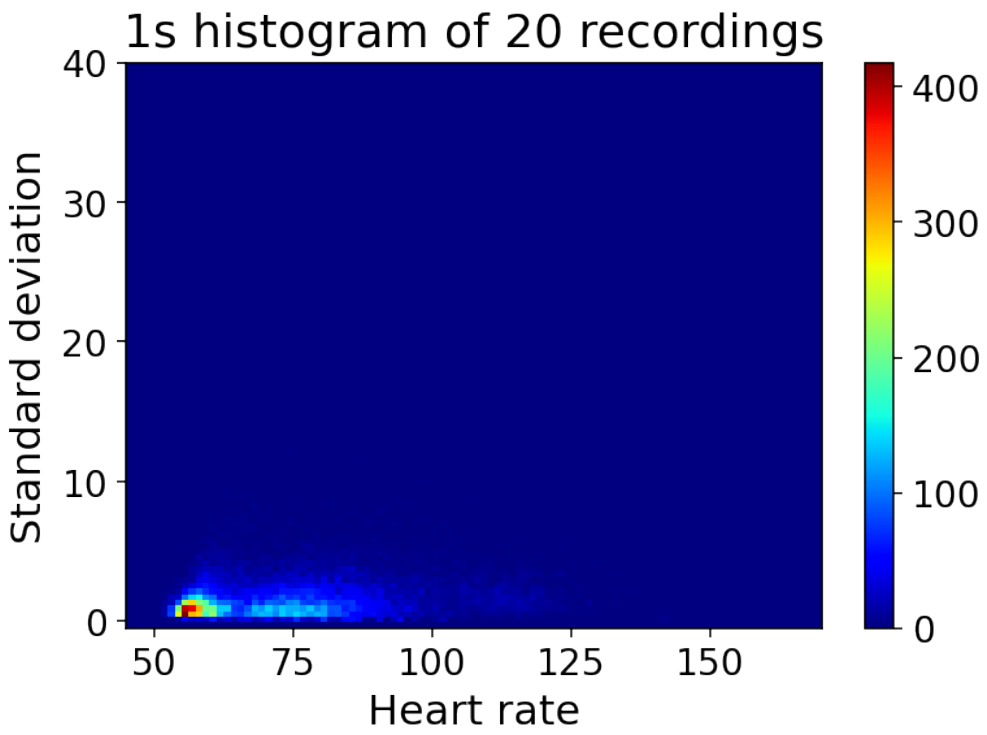


[25]:
```
f, axs = plt.subplots(1,2,figsize=(10,5))
f.suptitle('Comparing 1s and 5s intervals of 20 recordings')

axs[0].scatter(all_means_1s, all_stdevs_1s, s=2, c = "b", label = '1 s')
axs[0].set_xlabel("Heart rate")
axs[0].set_ylabel("Standard deviation") # Heart rate variability
axs[0].set_ylim([-0.5, 40])
axs[0].set_xlim([45, 170])

axs[1].scatter(all_means_5s, all_stdevs_5s, c = "k", s=2, label="5 s")
axs[1].set_xlabel("Heart rate")
axs[1].set_ylim([-0.5, 40])
axs[1].set_xlim([45, 170])


f.legend()
plt.show()
```

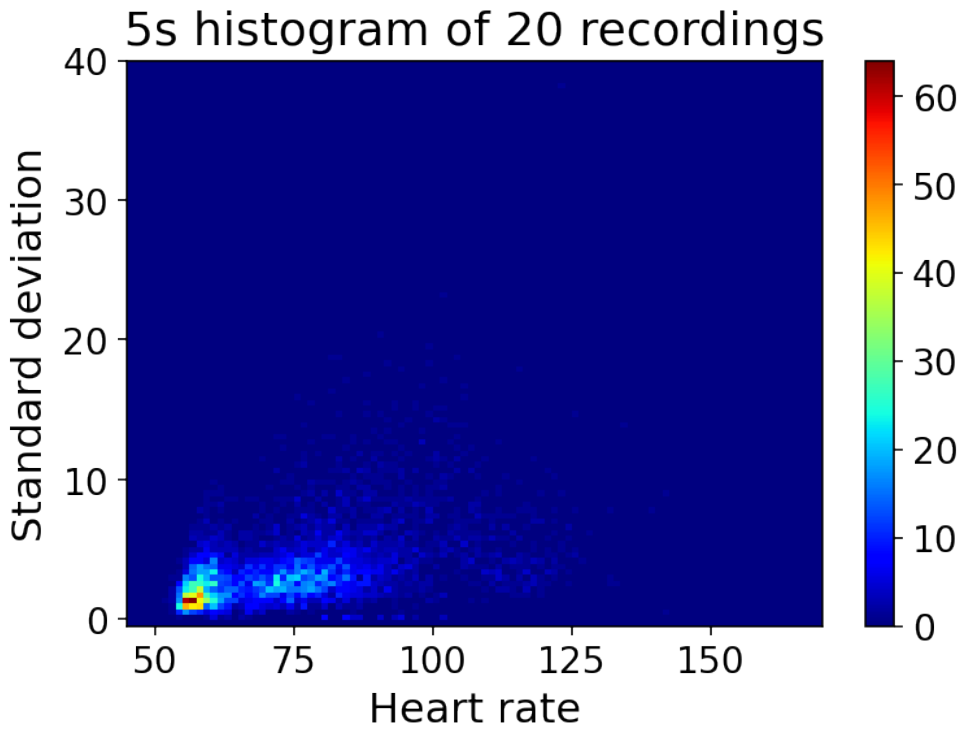


Generating the histogram of the previos data

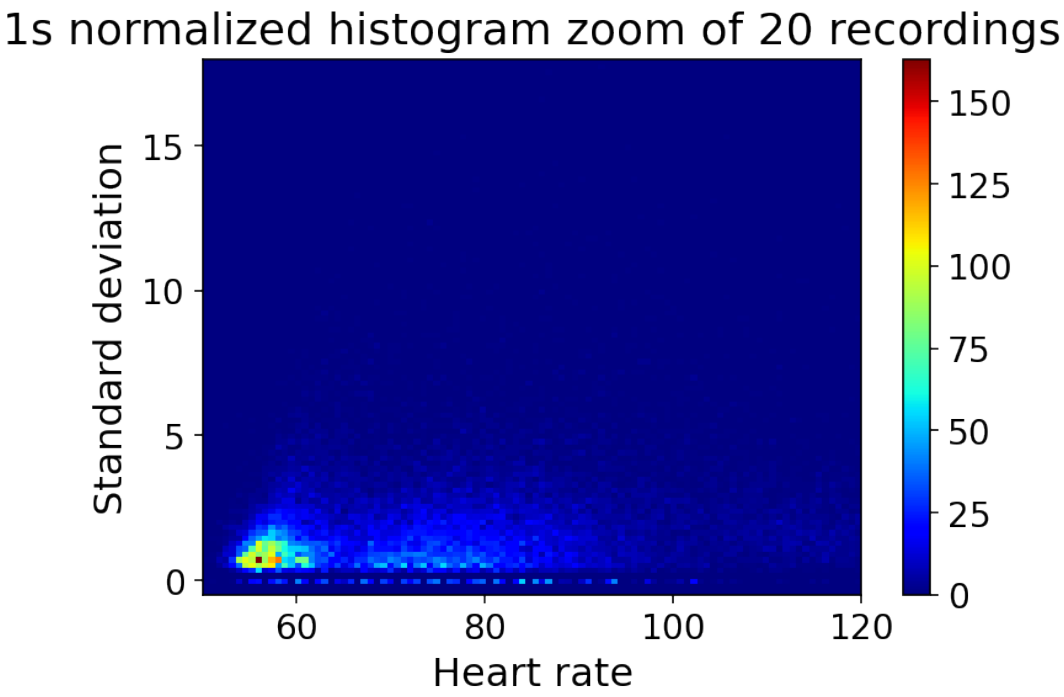
[26]:
```
plt.hist2d(all_means_1s, all_stdevs_1s, bins=(100, 100), range=[[45, 170], [-0.
↪5, 40]], density=False, weights=None, cmin=None, cmax=None, cmap="jet")
plt.title("1s histogram of 20 recordings")
plt.xlabel("Heart rate")
plt.ylabel("Standard deviation") # Heart rate variability

plt.colorbar()
plt.show()
```



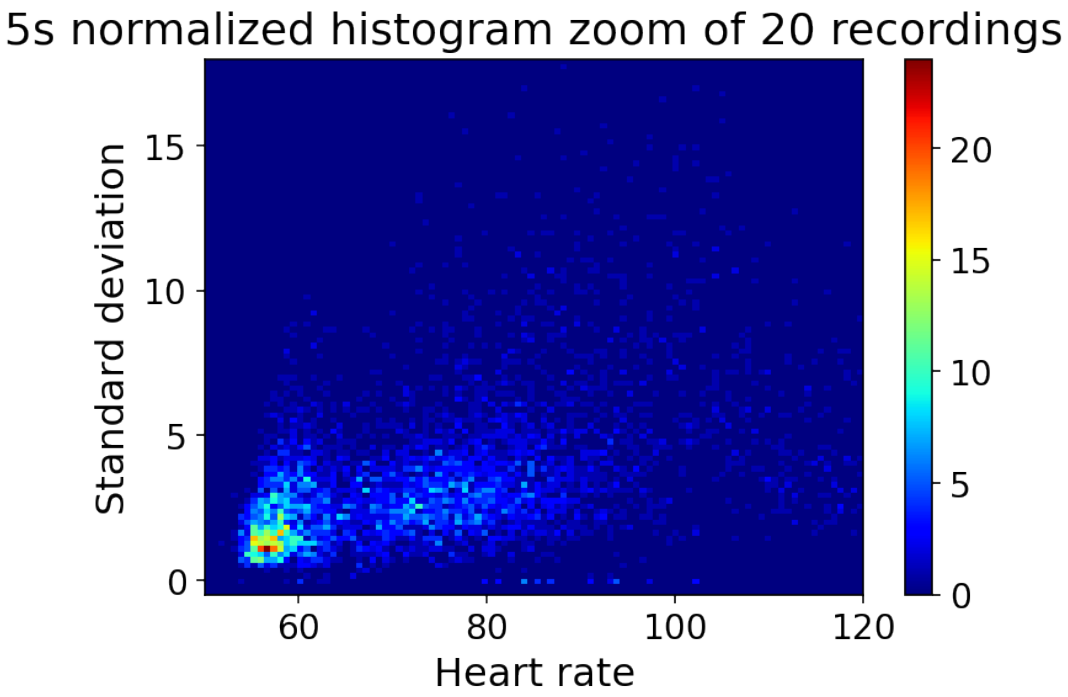
[27]:
```
plt.hist2d(all_means_5s, all_stdevs_5s, bins=(100, 100), range=[[45, 170], [-0.
↪5, 40]], density=False, weights=None, cmin=None, cmax=None, cmap="jet")
plt.title("5s histogram of 20 recordings")
plt.xlabel("Heart rate")
plt.ylabel("Standard deviation") # Heart rate variability
plt.colorbar()
plt.show()
```



[28]:
```
plt.hist2d(all_means_1s, all_stdevs_1s, bins=(100, 100), range=[[50, 120], [-0.
↪5, 18]], density=False, weights=None, cmin=None, cmax=None, cmap="jet")
plt.xlabel("Heart rate")
plt.ylabel("Standard deviation") # Heart rate variability
plt.title("1s normalized histogram zoom of 20 recordings")
plt.colorbar()
plt.show()
```



[29]:
```
plt.hist2d(all_means_5s, all_stdevs_5s, bins=(100, 100), range=[[50, 120], [-0.
↪5, 18]], density=False, weights=None, cmin=None, cmax=None, cmap="jet")
plt.title("5s normalized histogram zoom of 20 recordings")
plt.xlabel("Heart rate")
plt.ylabel("Standard deviation") # Heart rate variability
plt.colorbar()
plt.show()
```



Notice the two previous plots are comparable when multiplying the 5 s histogram by a factor of 5.

Now comes the interesting part, cluster these datapoints!

3 Clustering with k-means

The code in HW4 is designed for minimizing the distance of a single array in 1D (voltage). In this case, you have 2D: heart rate and standard deviation. Modify the functions accordingly so to consider minimizing the distance to these 2 variables.

[30]:
```
def iterative_cost_function(means, stdevs, mk_means, mk_stdevs, rnks): # Cost␣
↪function for 1 cluster
    n = len(means)
    J = 0
    for i in range(n):
        if rnks[i]:
            J += np.sum((means[i]-mk_means)**2 + (stdevs[i]-mk_stdevs)**2)␣
↪#Sums the cost for the 2 dimensions
    return J


def all_clusters_cost_function(means, stdevs, mk_means, mk_stdevs, rnks): #␣
↪Cost function for all clusters
    J = 0
```

```
    for cluster_i in range(len(mk_means)):
        J+= iterative_cost_function(means, stdevs, mk_means[cluster_i],␣
↪mk_stdevs[cluster_i], rnks[cluster_i])
    return J
```

[31]:
```
# Assigning a cluster based on distance between centroids and points:
def distances(means, stdevs, mk_means, mk_stdevs):
    number_of_clusters = len(mk_means) #3
    number_of_datapoints = len(means) # all the datapoints

    rnks_1 = np.zeros(number_of_datapoints, dtype = bool)
    rnks_2 = np.zeros(number_of_datapoints, dtype = bool)
    rnks_3 = np.zeros(number_of_datapoints, dtype = bool)

    for i in range(number_of_datapoints):
        distance = np.zeros(number_of_clusters)

        for cluster_i in range(number_of_clusters):
            # Considering the euclidian distance in 1 variable as well as in␣
↪the other
            distance[cluster_i]= np.sum((means[i]-mk_means[cluster_i])**2 +␣
↪(stdevs[i]-mk_stdevs[cluster_i])**2)

        minimum = np.argmin(distance) # For assigning a cluster to each␣
↪datapoint

        if minimum ==0: # Belongs in cluster 1
            rnks_1[i] = 1
        elif minimum ==1: # Belongs in cluster 2
            rnks_2[i] = 1
        elif minimum ==2: # Belongs in cluster 3
            rnks_3[i] = 1
    return rnks_1, rnks_2, rnks_3
```

[32]:
```
# Updating centroids
def updates_centroids(means, stdevs, rnks_1, rnks_2, rnks_3):
    c1_means = []
    c1_stdevs = []
    c2_means = []
    c2_stdevs = []
    c3_means = []
    c3_stdevs = []

    for i in range(len(means)):

        if rnks_1[i]:
```

```
            c1_means.append(means[i])
            c1_stdevs.append(stdevs[i])

        elif rnks_2[i]:
            c2_means.append(means[i])
            c2_stdevs.append(stdevs[i])

        elif rnks_3[i]:
            c3_means.append(means[i])
            c3_stdevs.append(stdevs[i])

    mks1_mean = np.mean(c1_means, 0)
    mks1_stdevs = np.mean(c1_stdevs, 0)

    mks2_mean = np.mean(c2_means, 0)
    mks2_stdevs = np.mean(c2_stdevs, 0)

    mks3_mean = np.mean(c3_means, 0)
    mks3_stdevs = np.mean(c3_stdevs, 0)

    return mks1_mean, mks1_stdevs, mks2_mean, mks2_stdevs, mks3_mean,␣
↪mks3_stdevs
```

[74]:
```
def one_iteration(means, stdevs, mks_mean_init, mks_stdevs_init):
    rnks_1, rnks_2, rnks_3 = distances(means, stdevs, mks_mean_init,␣
↪mks_stdevs_init)
    mks1_mean, mks1_stdevs, mks2_mean, mks2_stdevs, mks3_mean, mks3_stdevs =␣
↪updates_centroids(means, stdevs, rnks_1,
                                                                        ␣
↪                rnks_2, rnks_3)

    mks1_mean, mks1_stdevs, mks2_mean, mks2_stdevs, mks3_mean, mks3_stdevs =␣
↪updates_centroids(means, stdevs, rnks_1,
                                                                        ␣
↪                rnks_2, rnks_3)

    mks_means = [mks1_mean, mks2_mean, mks3_mean]
    mks_stdevs = [mks1_stdevs, mks2_stdevs, mks3_stdevs]
    rnks = [rnks_1, rnks_2, rnks_3]

    J = all_clusters_cost_function(means, stdevs, mks_means, mks_stdevs, rnks)
    return J, rnks, mks_means, mks_stdevs
```

[75]:
```
# Determines what data will run in the following cells, if the 1s one or the 5s.

# means = recordings[0].means
```

```
# stdevs = recordings[0].stdevs

means, stdevs, all_vecs_secs = all_means_1s, all_stdevs_1s, all_vecs_secs_1s
# means, stdevs, all_vecs_secs = all_means_5s, all_stdevs_5s, all_vecs_secs_5s
```

[76]:
```
pool = len(means)
# Initializes with 3 random mean and stdevs values from the dataset
mks_mean_init = [means[round(pool*random.random())], means[round(pool*random.
 ↪random())],
                 means[round(pool*random.random())]]
mks_stdevs_init = [stdevs[round(pool*random.random())],␣
 ↪stdevs[round(pool*random.random())],
                   stdevs[round(pool*random.random())]]

J = []
i = 0
converging = False

J1, rnks_1, mk_means, mk_stdevs = one_iteration(means, stdevs, mks_mean_init,␣
 ↪mks_stdevs_init) # First iteration
J.append(J1)

while converging == False:
# for i in range(10):
    J1, rnks, mk_means, mk_stdevs = one_iteration(means, stdevs, mk_means,␣
 ↪mk_stdevs)
    J.append(J1)

    if np.abs(J[i-1]-J[i])< 0.0001*J[i]: # Finding the convergence value
        print("Converging value occurs in the ", i+1, "th iteration.")
        converging = True
    i +=1
```

```
Converging value occurs in the  18 th iteration.
```

[77]:
```
J
```

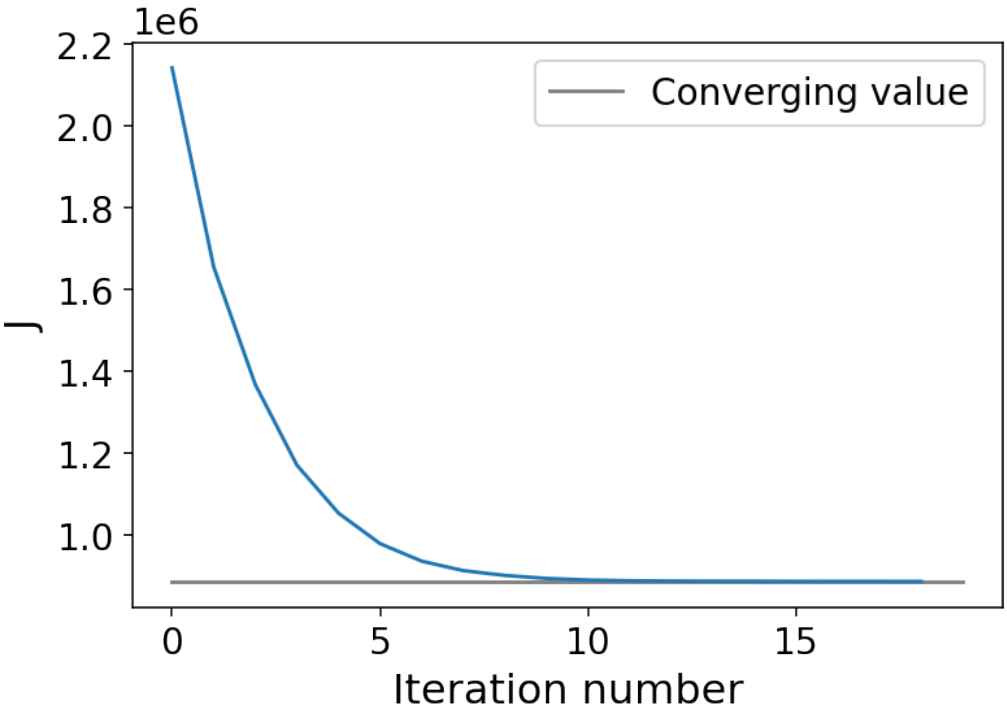
[77]:
```
[2141025.194059588,
 1655044.1950383754,
 1367697.5022342918,
 1170850.174435067,
 1053474.2277432163,
 978938.9626217925,
 936177.7792952323,
 913100.9026411718,
 901392.3754144716,
 894039.1949321056,
```

```
890178.6813390811,
888359.033164205,
887648.5432384766,
887389.333041912,
887189.3697697476,
886959.0058464453,
886868.2548050988,
886820.9778656499,
886803.2363798328]
```

4 IDEA

How many iterations did it take to converge? What convergence criterion are you using?

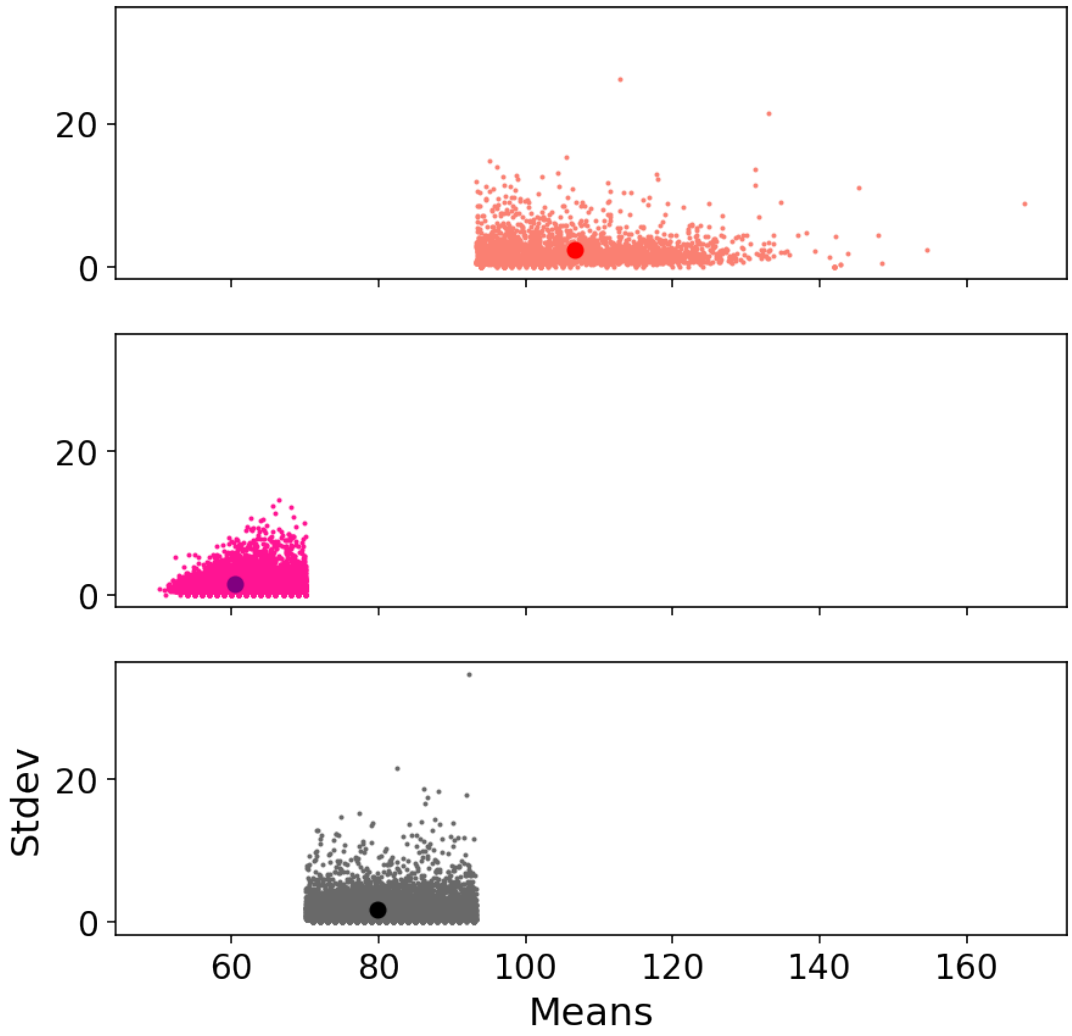
[78]:
```
max_ind = len(J)
plt.plot([0,max_ind], [J[max_ind-1], J[max_ind-1]], label = "Converging value",␣
↪color = "grey")
plt.plot(J)
plt.xlabel("Iteration number")
plt.ylabel("J")
plt.legend()
plt.show()
```



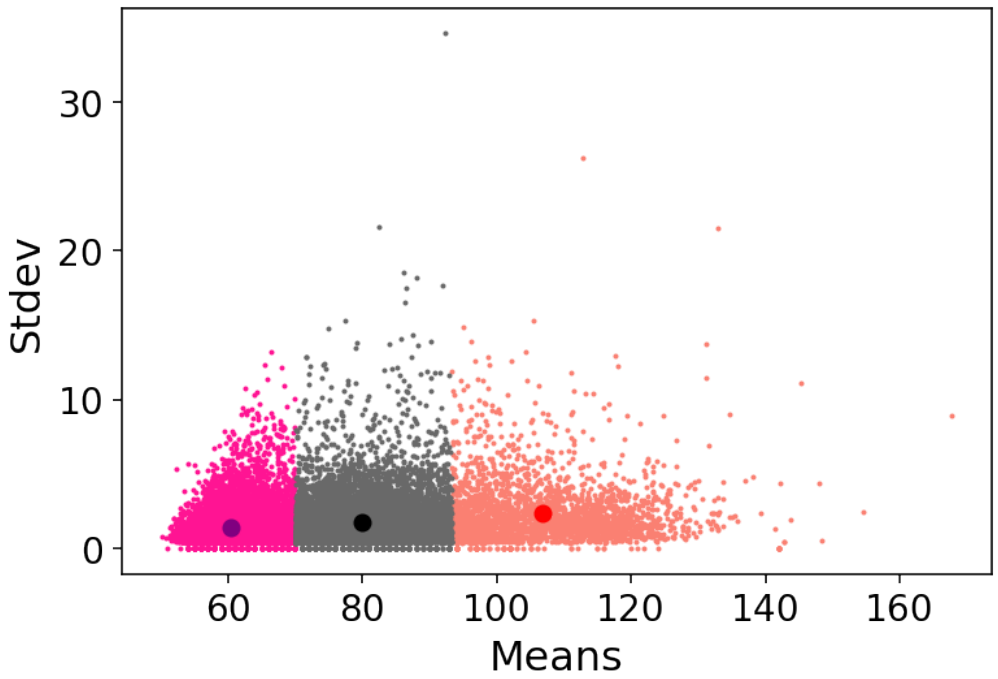
How to make the following cells run faster? Generate vectors.

[79]:
```
aux_means1 = []
aux_stdevs1 = []
aux_means2 = []
aux_stdevs2 = []
aux_means3 = []
aux_stdevs3 = []

for i in range(len(means)):

    if rnks[0][i]: #mk_means, mk_stdevs
        aux_means1.append(means[i]) #, stdevs[i], color="salmon", s=size)
        aux_stdevs1.append(stdevs[i]) #, stdevs[i], color="salmon", s=size)

    elif rnks[1][i]:
#         ax2.scatter(means[i], stdevs[i], color="deeppink", s=size)
        aux_means2.append(means[i]) #, stdevs[i], color="salmon", s=size)
        aux_stdevs2.append(stdevs[i]) #, stdevs[i], color="salmon", s=size)

    elif rnks[2][i]:
#         ax3.scatter(means[i], stdevs[i], color="dimgrey", s=size)
        aux_means3.append(means[i]) #, stdevs[i], color="salmon", s=size)
        aux_stdevs3.append(stdevs[i]) #, stdevs[i], color="salmon", s=size)
```

[80]:
```
fig, (ax1, ax2, ax3) = plt.subplots(3, sharex=True, sharey=True, figsize=(7,7))
size = 1
ax1.scatter(aux_means1, aux_stdevs1, color="salmon", s=size)
ax2.scatter(aux_means2, aux_stdevs2, color="deeppink", s=size)
ax3.scatter(aux_means3, aux_stdevs3, color="dimgrey", s=size)

ax1.scatter(mk_means[0], mk_stdevs[0], c="r")
ax2.scatter(mk_means[1], mk_stdevs[1], c="purple")
ax3.scatter(mk_means[2], mk_stdevs[2], c="k")
plt.xlabel('Means')
plt.ylabel("Stdev")
plt.show()
```



[81]:
```
plt.scatter(aux_means1, aux_stdevs1, color="salmon", s=size)
plt.scatter(aux_means2, aux_stdevs2, color="deeppink", s=size)
plt.scatter(aux_means3, aux_stdevs3, color="dimgrey", s=size)

plt.scatter(mk_means[0], mk_stdevs[0], c="r")
plt.scatter(mk_means[1], mk_stdevs[1], c="purple")
plt.scatter(mk_means[2], mk_stdevs[2], c="k")
plt.xlabel('Means')
plt.ylabel("Stdev")
plt.show()
```



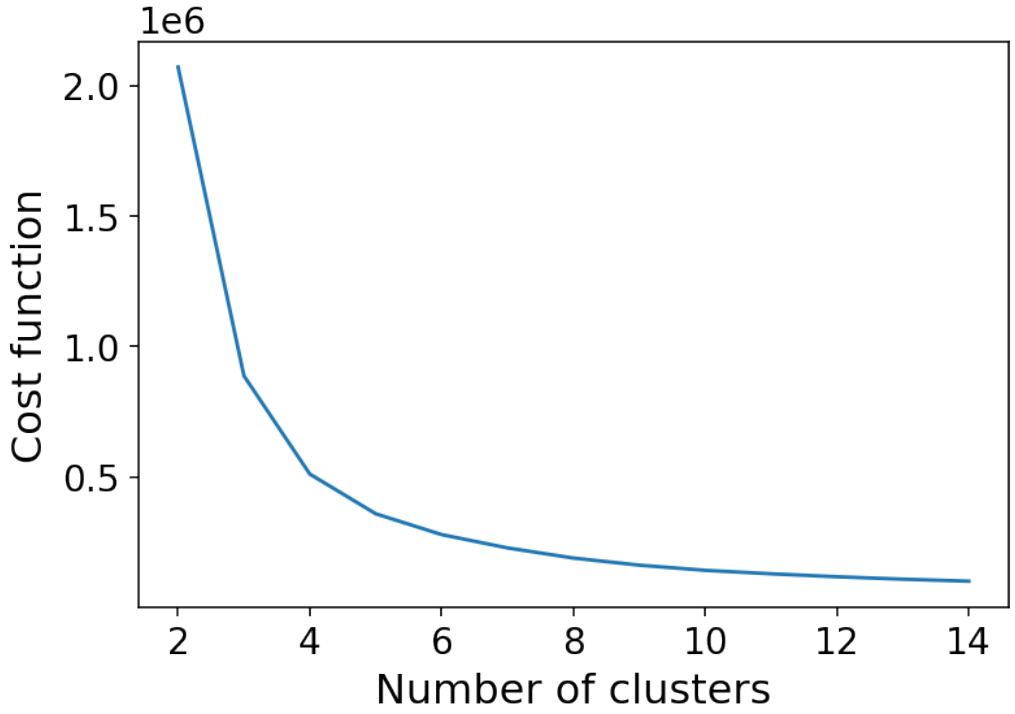
5 What is the optimal number of clusters?

Now, we will use python's KMeans function.

[82]:
```
ndata = [[means[i], stdevs[i]] for i in range(len(means))]
```

[83]:
```
iterations = 15
J = np.zeros(iterations)

for K in range(2,iterations):
    kmeans = KMeans(n_clusters=K, random_state=0, algorithm="full").fit(ndata)
    J[K] = kmeans.inertia_ # This must be the cost function J
```

[84]:
```
plt.plot([i+2 for i in range(13)], J[2:])
plt.xlabel("Number of clusters")
plt.ylabel("Cost function")
plt.show()
```

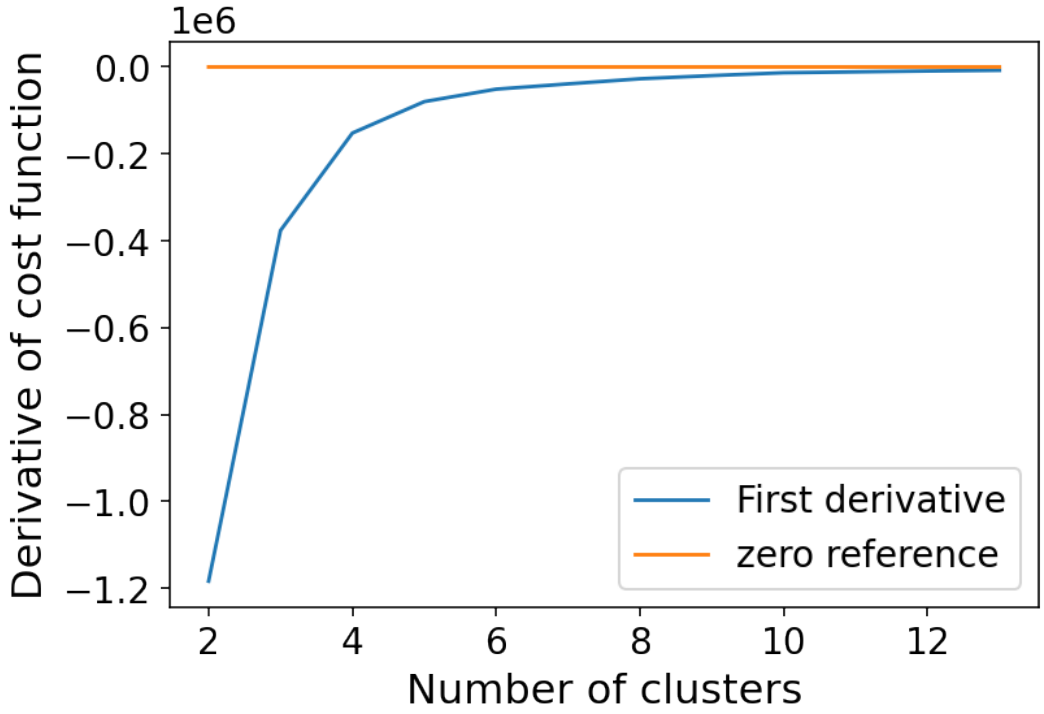


For finding the elbow of the previous curve, we can obtain the first derivative. When this value starts changes by a small amount, then we have found the elbow.

[85]:
```
def firstDerivative(x):
    deriv = np.zeros(len(x)-1)
    for i in range(len(x)-1):
        deriv[i] = x[i+1] - x[i]
    return deriv
```

[86]:
```
deriv1 = firstDerivative(J[2:])
```

[87]:
```
plt.plot([i+2 for i in range(12)], deriv1, label = "First derivative")
plt.plot([2,13], [2,0], label="zero reference")
plt.xlabel("Number of clusters")
plt.ylabel("Derivative of cost function")
plt.legend()
plt.show()
```



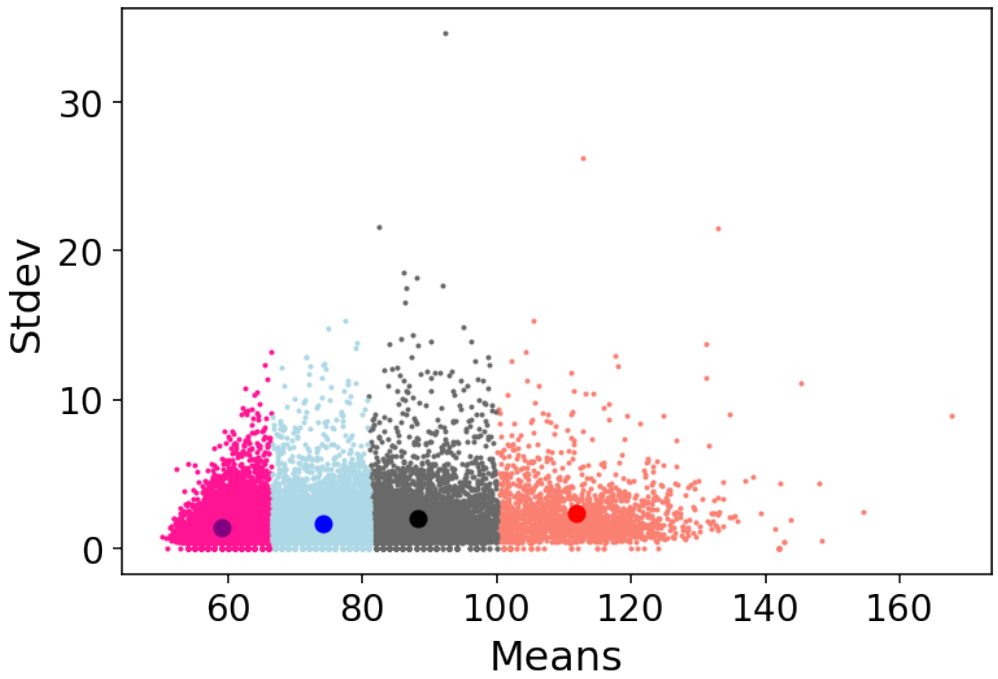
So the elbow of the previous curve is around 4 or 5 clusters. The difference between the derivative value and zero becomes small, which can be interpreted as falling below a certain threshold.

[88]:
```
K = 4
kmeans = KMeans(n_clusters=K, random_state=0, algorithm="full").fit(ndata)
```

[89]:
```
aux_means1 = []
aux_stdevs1 = []
aux_means2 = []
aux_stdevs2 = []
aux_means3 = []
aux_stdevs3 = []
aux_means4 = []
aux_stdevs4 = []

for i in range(len(means)):

    if kmeans.labels_[i] == 0:
        aux_means1.append(means[i]) #, stdevs[i], color="salmon", s=size)
        aux_stdevs1.append(stdevs[i]) #, stdevs[i], color="salmon", s=size)

    elif kmeans.labels_[i] == 1:
#         ax2.scatter(means[i], stdevs[i], color="deeppink", s=size)
```

```
        aux_means2.append(means[i]) #, stdevs[i], color="salmon", s=size)
        aux_stdevs2.append(stdevs[i]) #, stdevs[i], color="salmon", s=size)

    elif kmeans.labels_[i] == 2:
#         ax3.scatter(means[i], stdevs[i], color="dimgrey", s=size)
        aux_means3.append(means[i]) #, stdevs[i], color="salmon", s=size)
        aux_stdevs3.append(stdevs[i]) #, stdevs[i], color="salmon", s=size)

    elif kmeans.labels_[i] == 3:
        aux_means4.append(means[i]) #, stdevs[i], color="salmon", s=size)
        aux_stdevs4.append(stdevs[i]) #, stdevs[i], color="salmon", s=size)

    else:
        print("Something is worng in iteration: ",i)
```

[90]:
```
plt.scatter(aux_means1, aux_stdevs1, color="salmon", s=size)
plt.scatter(aux_means2, aux_stdevs2, color="deeppink", s=size)
plt.scatter(aux_means3, aux_stdevs3, color="dimgrey", s=size)
plt.scatter(aux_means4, aux_stdevs4, color="lightblue", s=size)

plt.scatter(kmeans.cluster_centers_[0,0], kmeans.cluster_centers_[0,1], c="r")
plt.scatter(kmeans.cluster_centers_[1,0], kmeans.cluster_centers_[1,1],␣
 ↪c="purple")
plt.scatter(kmeans.cluster_centers_[2,0], kmeans.cluster_centers_[2,1], c="k")
plt.scatter(kmeans.cluster_centers_[3,0], kmeans.cluster_centers_[3,1], c="b")
plt.xlabel('Means')
plt.ylabel("Stdev")
plt.show()
```

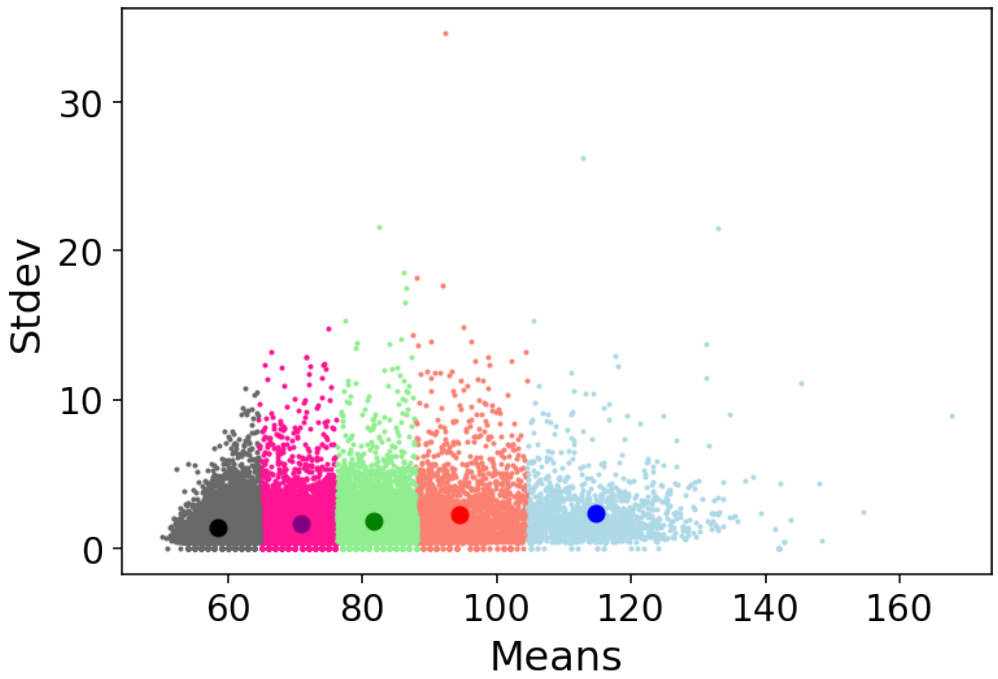


[91]:
```
K = 5
kmeans = KMeans(n_clusters=K, random_state=0, algorithm="full").fit(ndata)
```

[92]:
```
aux_means1 = []
aux_stdevs1 = []
aux_means2 = []
aux_stdevs2 = []
aux_means3 = []
aux_stdevs3 = []
aux_means4 = []
aux_stdevs4 = []
aux_means5 = []
aux_stdevs5 = []

for i in range(len(means)):

    if kmeans.labels_[i] == 0:
        aux_means1.append(means[i]) #, stdevs[i], color="salmon", s=size)
        aux_stdevs1.append(stdevs[i]) #, stdevs[i], color="salmon", s=size)

    elif kmeans.labels_[i] == 1:
#         ax2.scatter(means[i], stdevs[i], color="deeppink", s=size)
        aux_means2.append(means[i]) #, stdevs[i], color="salmon", s=size)
        aux_stdevs2.append(stdevs[i]) #, stdevs[i], color="salmon", s=size)
```

```
    elif kmeans.labels_[i] == 2:
#         ax3.scatter(means[i], stdevs[i], color="dimgrey", s=size)
        aux_means3.append(means[i]) #, stdevs[i], color="salmon", s=size)
        aux_stdevs3.append(stdevs[i]) #, stdevs[i], color="salmon", s=size)

    elif kmeans.labels_[i] == 3:
        aux_means4.append(means[i]) #, stdevs[i], color="salmon", s=size)
        aux_stdevs4.append(stdevs[i]) #, stdevs[i], color="salmon", s=size)

    elif kmeans.labels_[i] == 4:
        aux_means5.append(means[i]) #, stdevs[i], color="salmon", s=size)
        aux_stdevs5.append(stdevs[i]) #, stdevs[i], color="salmon", s=size)

    else:
        print("Something is worng in iteration: ",i)
```

[93]:
```
plt.scatter(aux_means1, aux_stdevs1, color="salmon", s=size)
plt.scatter(aux_means2, aux_stdevs2, color="deeppink", s=size)
plt.scatter(aux_means3, aux_stdevs3, color="dimgrey", s=size)
plt.scatter(aux_means4, aux_stdevs4, color="lightblue", s=size)
plt.scatter(aux_means5, aux_stdevs5, color="lightgreen", s=size)

plt.scatter(kmeans.cluster_centers_[0,0], kmeans.cluster_centers_[0,1], c="r")
plt.scatter(kmeans.cluster_centers_[1,0], kmeans.cluster_centers_[1,1],␣
 ↪c="purple")
plt.scatter(kmeans.cluster_centers_[2,0], kmeans.cluster_centers_[2,1], c="k")
plt.scatter(kmeans.cluster_centers_[3,0], kmeans.cluster_centers_[3,1], c="b")
plt.scatter(kmeans.cluster_centers_[4,0], kmeans.cluster_centers_[4,1],␣
 ↪c="green")
plt.xlabel('Means')
plt.ylabel("Stdev")
plt.show()
```



[94]:
```
# Labeling the data based on the time of day
def activity_label_per_day(vecs_secs):
    activity_labels = []#np.zeros(len(vecs_secs))

    for i in range(len(vecs_secs)): # for every second in the day
        mean_time_5s_int = round(np.mean(vecs_secs[i]))

        if mean_time_5s_int >=86400: # The index corresponds to the next day
            mean_time_5s_int = mean_time_5s_int - 86400

        if 0 <= mean_time_5s_int and mean_time_5s_int < 21600: # 60*60*6␣
↪between midnight and 6 am
            activity_labels.append(0) # "sleep_0-6"
        elif 21600 <= mean_time_5s_int and mean_time_5s_int <= 43200: # 60*60*6␣
↪between midnight and 6 am
            activity_labels.append(1) # "morning_6-12"
        elif 43200 <= mean_time_5s_int and mean_time_5s_int < 50400:
            activity_labels.append(2) # "eating_12-2pm"
        elif 50400 <= mean_time_5s_int and mean_time_5s_int < 68400:
            activity_labels.append(3) # "afternoon_2-7pm"
        elif 68400 <= mean_time_5s_int and mean_time_5s_int < 75600:
            activity_labels.append(4) # "possibly_dancing_7-9pm"
        elif 75600 <= mean_time_5s_int and mean_time_5s_int < 86400:
            activity_labels.append(5) # "night_9-12pm"
```

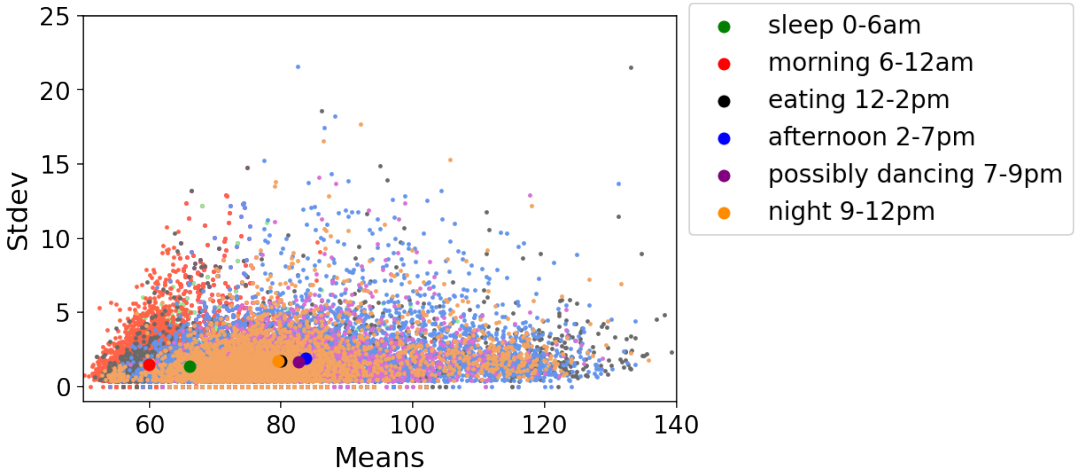
```
        else:
            print("There is something wrong with index: ",i)

    return activity_labels
```

[95]:
```
activity_labels = activity_label_per_day(all_vecs_secs)
```

[96]:
```
# Obtain the mean of the labeled activity clusters
values_c0_means = []
values_c0_stdevs = []
values_c1_means = []
values_c1_stdevs = []
values_c2_means = []
values_c2_stdevs = []
values_c3_means = []
values_c3_stdevs = []
values_c4_means = []
values_c4_stdevs = []
values_c5_means = []
values_c5_stdevs = []

for i in range(len(means)):
    if activity_labels[i] == 0:
        values_c0_means.append(means[i])
        values_c0_stdevs.append(stdevs[i])

    elif activity_labels[i] == 1:
        values_c1_means.append(means[i])
        values_c1_stdevs.append(stdevs[i])

    elif activity_labels[i] == 2:
        values_c2_means.append(means[i])
        values_c2_stdevs.append(stdevs[i])

    elif activity_labels[i] == 3:
        values_c3_means.append(means[i])
        values_c3_stdevs.append(stdevs[i])

    elif activity_labels[i] == 4:
        values_c4_means.append(means[i])
        values_c4_stdevs.append(stdevs[i])

    elif activity_labels[i] == 5:
        values_c5_means.append(means[i])
        values_c5_stdevs.append(stdevs[i])

    else:
```

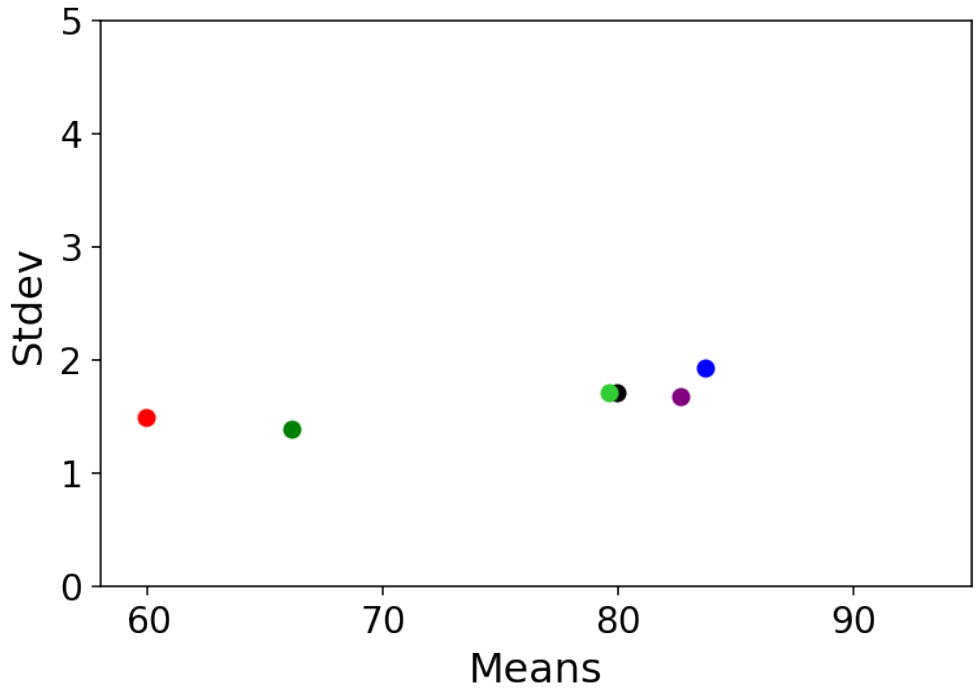
```
        print("Something is wrong in iteration: ",i)
```

[97]:
```
size = 2

plt.scatter(aux_means2, aux_stdevs2, color="deeppink", s=size)

plt.scatter(values_c0_means, values_c0_stdevs, c="lightgreen", s=2)
plt.scatter(values_c1_means, values_c1_stdevs, c="tomato", s=2)
plt.scatter(values_c2_means, values_c2_stdevs, c="dimgrey", s=2)
plt.scatter(values_c3_means, values_c3_stdevs, c="cornflowerblue", s=2)
plt.scatter(values_c4_means, values_c4_stdevs, c="orchid", s=2)
plt.scatter(values_c5_means, values_c5_stdevs, c="sandybrown", s=2)

plt.scatter(np.mean(values_c0_means), np.mean(values_c0_stdevs), c="green",␣
 ↪label = "sleep 0-6am")
plt.scatter(np.mean(values_c1_means), np.mean(values_c1_stdevs), c="r", label =␣
 ↪"morning 6-12am")
plt.scatter(np.mean(values_c2_means), np.mean(values_c2_stdevs), c="k", label =␣
 ↪"eating 12-2pm")
plt.scatter(np.mean(values_c3_means), np.mean(values_c3_stdevs), c="b",␣
 ↪label="afternoon 2-7pm")
plt.scatter(np.mean(values_c4_means), np.mean(values_c4_stdevs), c="purple",␣
 ↪label = "possibly dancing 7-9pm")
plt.scatter(np.mean(values_c5_means), np.mean(values_c5_stdevs),␣
 ↪c="darkorange", label = "night 9-12pm")
plt.xlabel('Means')
plt.ylabel("Stdev")
plt.xlim([50, 140])
plt.ylim([-1, 25])
plt.legend(bbox_to_anchor=(1,0.4))
plt.show()
```



[98]:
```
plt.scatter(np.mean(values_c0_means), np.mean(values_c0_stdevs), c="green")
plt.scatter(np.mean(values_c1_means), np.mean(values_c1_stdevs), c="r")
plt.scatter(np.mean(values_c2_means), np.mean(values_c2_stdevs), c="k")
plt.scatter(np.mean(values_c3_means), np.mean(values_c3_stdevs), c="b")
plt.scatter(np.mean(values_c4_means), np.mean(values_c4_stdevs), c="purple")
plt.scatter(np.mean(values_c5_means), np.mean(values_c5_stdevs), c="limegreen")
plt.xlim(58,95)
plt.ylim(0,5)
plt.xlabel('Means')
plt.ylabel("Stdev")
```

[98]:
```
Text(0, 0.5, 'Stdev')
```



[ ]:

[ ]:

6 IDEA

Does the optimal number of clusters change if we modify the time window considered to 1 min?

7 ARGUE. Why the labels do not work in this case. Different physical activity data under the same label. Side by side comparison of 5 or 6 clusters and this labeling.

8 ARGUE The optimal number of clusters related with some physiological value.

Amount of exercise. - Red one is when sleeping. - Next one some activity when awake that is not vigorous, like taking a class, being in a desk. - Next one could be like walking or such - Next one riding a bike, dancing?

I am curious for the amount of stdev in some of the recordings, though.

Maybe the stadard deviation associated to each data cluster can be associated too with the amounbt of heart rate variability or healthiness of a person? I do not know, some of the dots with high standard deviation must be due to noise or artifacts. But there must be some intrinsic variability of the heart which must be related to health.

color="green", s=size, label="sleep_0-6" color="red", s=size, label = "morning_6-12" color="black", s=size, label = "eating_12-2pm" color="blue", s=size, label = "afternoon_2-7pm" color="purple", s=size, label = "possibly_dancing_7-9pm" color="limegreen", s=size, label = "night_9-12pm"

[ ]:

9 IDEA

What if we now use gaussian mixtures to cluster?

You have several recordings,so you could divide it in the training and test data. Wait, no. Divide the data from the same recording? Use recordings from multiple days? Well, everything is just one big continuos recording. But yes, you could use complete days as training or test.

Use covariance_type='full'

10 IDEA

Use principal components analysis to cluster.

11 THE CODE SHOULD RUN UNTIL HERE.

[99]:
```
training = Peaks[:5000,:]
# means[i], stdevs
classification = gmix.fit(training).predict(training) +1 # You are assinging to␣
↪each component one class from 1 to 10
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
```

```
/tmp/ipykernel_10596/1457066691.py in <module>
----> 1 training = Peaks[:5000,:]
      2 # means[i], stdevs
      3 classification = gmix.fit(training).predict(training) +1 # You are␣
 ↪assinging to each component one class from 1 to 10

NameError: name 'Peaks' is not defined
```

[ ]:
```
# Code from homework assignment HW4
gmix = mixture.GaussianMixture(n_components=10, covariance_type='full')
gmix.fit(Peaks)
np.shape(Peaks)

training = Peaks[:5000,:]
classification = gmix.fit(training).predict(training) +1 # You are assinging to␣
 ↪each component one class from 1 to 10
```

[ ]:
```
# Code from HW4 part a
# Plot with cluster labels legend
PP = pd.DataFrame(np.array(training))
PP["Cluster"] = classification.tolist()
g = sns.PairGrid(PP, hue = "Cluster")
g = g.map_lower(plt.hexbin, gridsize=50, mincnt=1, cmap='jet_r',bins='log')

for i, j in zip(*np.triu_indices_from(g.axes, 0)):
    g.axes[i, j].set_visible(False)
g.add_legend()
plt.show()
```

[ ]:
```
# Code from HW4 part b
training = Peaks[:5000,:]
test = Peaks[5000:10000,:]

ks = [i+8 for i in range(13)]
scores = np.zeros(len(ks))
classifications = []

for i in range(len(ks)):
    gmix = mixture.GaussianMixture(n_components=ks[i], covariance_type='full')
    classification = gmix.fit(training).predict(test) +1 # You are assinging to␣
 ↪each component one class
    classifications.append(classification)
    scores[i] = gmix.fit(training).score(test) # Likelihood of the test data

# K value that maximizes the likelihood
best_k_value = np.argmax(scores)+8
print("Number of clusters (K) that maximizes the likelihood: ",best_k_value)
```

[ ]:
```
plt.plot(ks, scores)
plt.xlabel("K")
plt.ylabel("Likelihood")
plt.title("One trial")
plt.show()
```

[ ]:
```
# Plot with cluster labels legend
PP = pd.DataFrame(np.array(test))
PP["Cluster"] = classifications[best_k_value-8].tolist()
g = sns.PairGrid(PP, hue = "Cluster")
g = g.map_lower(plt.hexbin, gridsize=50, mincnt=1, cmap='jet',bins='log')

for i, j in zip(*np.triu_indices_from(g.axes, 0)):
    g.axes[i, j].set_visible(False)
g.add_legend()
plt.show()
```

[ ]:
```
# Suffling the data and obtain an average likelihood
ks = [i+8 for i in range(13)]
scores = np.zeros(len(ks))
repetitions = 20

for i in range(len(ks)):
    for same_k_i in range(repetitions):
        # In each iteration, data will be shuffled again
        shuffled_data = shuffle(Peaks)
        training = shuffled_data[:5000,:]
        test = shuffled_data[5000:10000,:]

        gmix = mixture.GaussianMixture(n_components=ks[i],␣
 ↪covariance_type='full')
        scores[i] += gmix.fit(training).score(test) # Likelihood of the test␣
 ↪data

scores = scores/repetitions

best_averaged_k_value = np.round(np.argmax(scores)+8)
print("K value that maximizes the likelihood:",best_averaged_k_value)
```

[ ]:
```
plt.plot(ks, scores)
plt.xlabel("K")
plt.ylabel("Likelihood")
plt.title("Average trend")
plt.show()
```

[ ]:
```

```

[ ]:

[ ]:

[ ]:

[ ]:

12 Review and modify based on Della's comments in this NB

Comments on HW5 (PCA): Q1. a. (-4) sklearn PCA input data X should be of shape (n_samples, n_features). pca.explained_variance_ is the eigenvalue, not pca.singular_values_. You did it right in b and c, I am confused why you did (a) differently :0 Q2. e. (-3) For all channel data, you shouldn't truncate eigenvectors_160 while computing projections. This is why you are computing total_var_backprojection_160 wrong Q3. a. (-1) The assumed noise would draw the projected points closer to their mean b. (-5) You result didn't look right. While estimating new psi, you need to use np.diag(np.diag()) to get the diagonal of the matrix. Further, if you run the FA multiple times with different random psi initializations, you will find FA gives multiple local optimums.

[ ]:

[ ]:

[ ]:

Della's comments for further reference

Comments on HW4: - Q1. (-3) Recall that you are using argmax(x[5:25]), you need to add 5 back to the argmax index when extracted peak values - Q4. (-5) You had the same mistake as Q1 so the peaks you were using are wrong. - Q5. b. (-2) Distance should be computed as the euclidean distance rather than L1 norm c. (-5) The datapoints at the boundary of clusters should have minimum euclidean distance to all 2/3/4 centers

Comments about HW3:

Q1. Actually I would prefer to call the 'lambda' you computed in each trial as firing rate since you haven't done any averaging across trials yet. (-1) To avoid error of log(lambda=0), replace lambda with small value like 1e-5. Setting 'log_term = -230*x_d' doesn't seem to be a good way to go. Q2. Nice figures and labeling. Don't literally need to include movement data. 2b, (-3) Lack of description and explaination on your results. Q3. (-5) Since you are randomly droping neurons, you need to randomly drop them by 20 times. Then estimate the mean and variations of decoding accuracy. You'll find your curve (decoding accuracy vs. number of dropped neurons) much smoother.

[ ]: