Data Science – Feature extraction from Time Series data

Yoran de Weert (s1693859)
Selina Zwerver (s1690388)

December 3, 2020

Exercise 4.1

a) The training data set contains 7352 rows and 561 columns. The testing data set contains 2947 rows and 561 columns. The rows correspond to individuals doing an activity. The columns represent the values for a certain feature.

b) To determine the statistics, the test and training data were combined such that the total data set contains 10299 rows. Table 1 shows the mean, median and the standard deviation for six different features.

Feature	**Mean [g]**	**Median [g]**	**Std [g]**
tBodyAcc-mean()-X	0.274	0.277	0.068
tBodyAcc-mean()-Y	-0.018	-0.017	0.037
tBodyAcc-mean()-Z	-0.109	-0.109	0.053
tBodyAcc-std()-X	-0.608	-0.943	0.439
tBodyAcc-std()-Y	-0.510	-0.835	0.500
tBodyAcc-std()-Z	-0.613	-0.851	0.404

Table 1: Mean, median and standard deviation (std) for certain features.

Exercise 4.2

a) The training data set contains 7352 rows and 1 column. The testing data set contains 2947 rows and 1 column. The column represents the index for the activity label. The column of the 'Y_train.txt' (respectively 'Y_test.txt') corresponds with the activity performed in the row of 'X_train.txt' ('X_test.txt').

b) The number of data points for each user activity is shown in Figure 1. It is important to have a balanced data set in machine learning and classification purposes to ensure that the model performs its task successfully. If the amount of data points from one user activity outnumber the other, the data is skewed in favour of this activity. As a consequence, this activity will dominate the others, and the model is trained to be more focused on this activity. From the six activities in the given data set, the activities 'Walking downstairs' and 'Walking upstairs' are less represented in the data set, although there is still a significant amount of data points to not be heavily dominated by other activities.

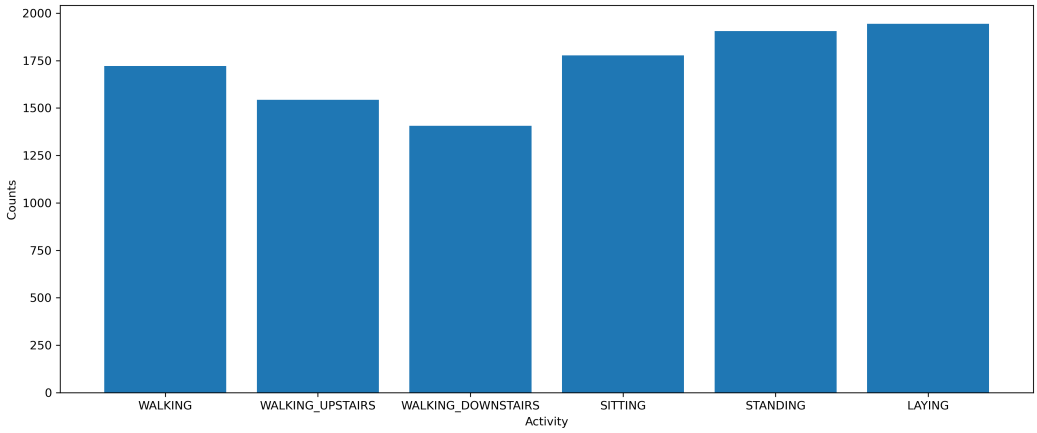

Figure 1: Bar chart of the number of data points for each user activity.

Exercise 4.3

Figure 2 shows the distribution of the feature tBodyAcc - correlation()-X,Y. The distributions appear to be normal for the 6 activities, but the means for the activities differ. The activities laying, standing and walking have a mean around 0.0, while the other activities have a lower mean. The distributions with a positive skewness belong the activities laying and standing. The distribution of walking upstairs has a slightly negative skewness. The activity of walking upstairs has two peaks and indicates a bimodal distribution. This may also be true for the activity laying, since a small peak can be observed where the tBodyAcc-correlation is 1.0.

Figure 3 shows the distribution of the feature tGravityAcc-sma(). This appears to be normally distributed for all the six activities. The activity laying seems to have 2 modes around -0.5 and 0.4. It is is not symmetric nor skewed. The activity sitting has 3 modes with means around -0.5, 0.3 and 1.0. This distribution is positively skewed and is not symmetric. The standing activity is unimodal, and not symmetric. Since there are more data points on the right side of the mean, it is positively skewed. The walking activity is bimodal with means around -0.2 and 0.5. The skewness is slightly positive and the distribution is not symmetric. The walking downstairs activity is as well bimodal with means around -0.6 and -0.3. It is almost symmetric, although the distribution is slightly positively skewed. Walking upstairs is multimodal with 3 modes. It is neither symmetric nor skewed.

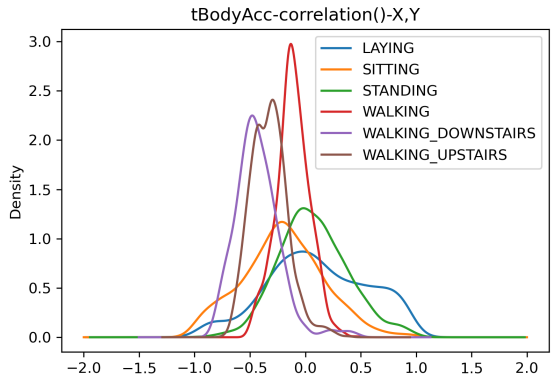

Figure 2: Distributions of tBodyAcc-correlation()-X,Y for the 6 activities

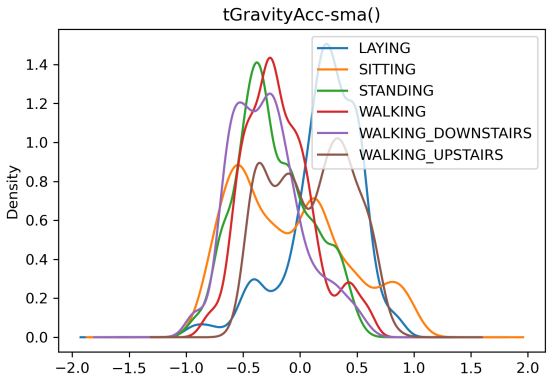

Figure 3: Distributions of tGravityAcc-sma() for the 6 activities

Figure 4 shows the distribution of the feature tGravityAcc-arCoeff(),Y,1. Again, the distributions seem to be normal. The activities walking upstairs, walking downstairs and standing are symmetric and hence not skewed. The laying activity is negatively skewed and unimodal. The sitting activity is bimodal and is not symmetric. The activity walking is normally distributed and might even be bimodal. The distribution seems symmetric as well.

Figure 5 shows the distribution of the feature tBodyGyroJerkMag-min(). This seems, for all activities, to be normally distributed with relatively small standard deviations compared to the mean and causes the bell-shaped area of the distribution to be narrow and high. At the right-side of the bell-shape, there is a small peak and might indicate a slight positive skewness or even a second mode in the distribution. Due to the overlaying lines, it is not possible to observe which activities have these small peaks.

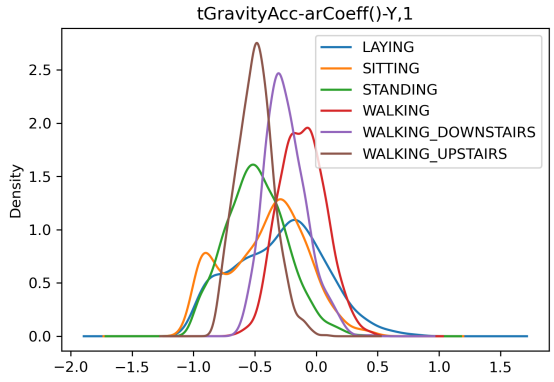

Figure 4: Distributions of tGravityAcc-arCoeff()-Y,1 for the 6 activities

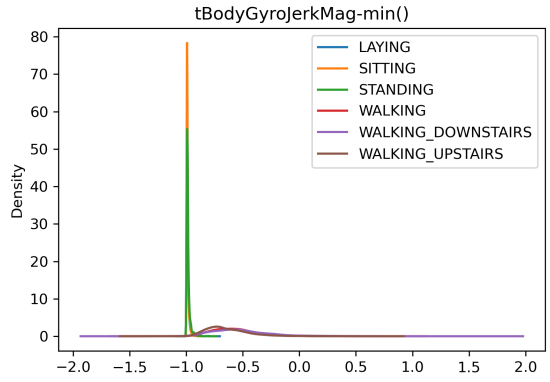

Figure 5: Distributions of tBodyGyroJerkMag-min() for the 6 activities

Figure 6 shows the distribution of the feature tBodyGyroJerkMag-iqr(). This figure is similar to the distributions of the feature tBodyGyroJerkMag-min() in Figure 5. One difference is that the small

peaks at the right side of the bell-shape are higher and again indicates a small positive skewness for some activities.

Figure 7 shows the distribution of the feature fBodyAcc-max()-Y. Again, this feature seem to have a bimodal distribution for most of the activities with the only exceptions being sitting and standing. The curve of the latter two activities is bell-shaped and indicates a normal distribution. Also, the distribution seems symmetric. The bimodal distribution of the remaining features consists of two combined normal distributions. The first of the two modes is similar to the curves of the sitting and standing activities. Regarding the second mode of the bimodal distributions, the skewness is positive for walking downstairs and walking upstairs, negative for the walking. The only curve that is not clearly visible belongs to the laying activity. Some small parts are visible and hint towards a unimodal, normal distribution.

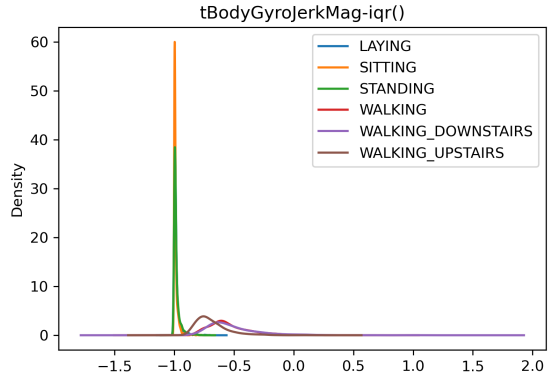

Figure 6: Distributions of tBodyGyroJerkMag-iqr() for the 6 activities

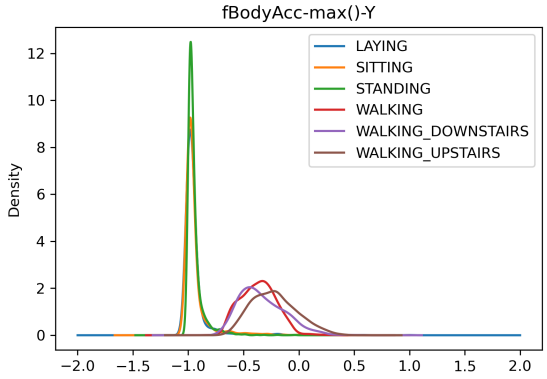

Figure 7: Distributions of fBodyAcc-max()-Y for the 6 activities

Figure 8 and Figure 9 seem to have similar characteristics and represent the features fBodyAcc-bandsEnergy()-49,56. and fBodyAccjerk-energy()-Z, respectively. There is only one peak visible around -1.0 and indicates a normal distribution with a small standard deviation. All curves seem symmetric and unimodal.

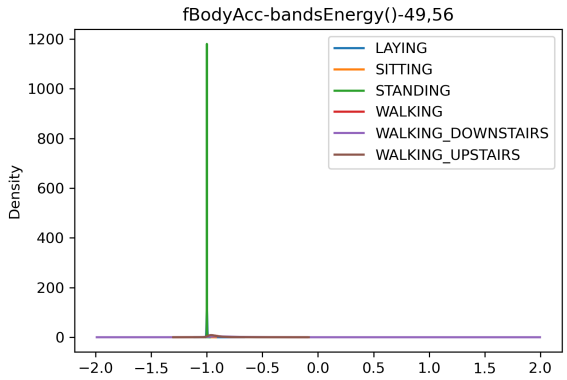

Figure 8: Distribution of fBodyAcc-bandsEnergy()-49,56 for the 6 activities

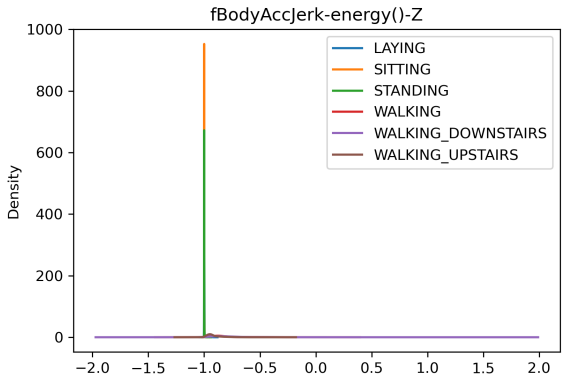

Figure 9: Distribution of fBodyAccjerk-energy()-Z for the 6 activities

Figure 10 shows the distribution of the feature fBodyBodyAccJerkMag-meanFreq(). This feature is normally distributed for the activities walking, walking downstairs and upstairs. It is symmetric and not skewed. The feature is also normally distributed for the remaining activities, but is negatively skewed. All distributions are unimodal.

Figure 11 shows the distribution of the feature fBodyBodyGyroJerkMag - meanFreq(). The distributions of the feature are similar to the distributions of the feature fBodyBodyAccJerkMag-meanFreq() for all activities. The feature is symmetric and normally distributed for the activities walking, walking upstairs and downstairs. For the other activities, the feature is normally distributed as well, but negatively skewed.

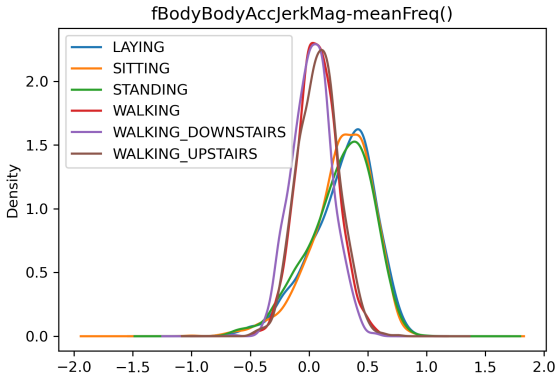

Figure 10: Distribution of fBodyBodyAccJerkMag-meanFreq()

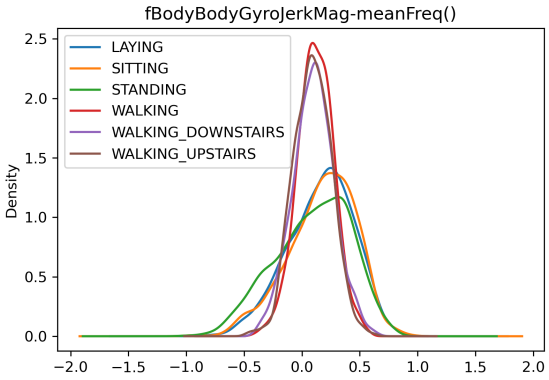

Figure 11: Distribution of fBodyBodyGyroJerkMag - meanFreq()

Activities can be discriminated based on the chosen features. As can be seen in the figures, some features behave similar for each activity, for example fBodyAcc-bandsEnergy()-49,56. If, by any chance, only features are selected that behave similar, discrimination between activities is not possible and it might wrongly be concluded that all activities behave similar. On the other hand, there

are also features that clearly show differences as the feature fBodyBodyAccJerkMag-meanFreq(). The corresponding figure, Figure 10, shows a clear difference between the walking activities and the other three activities. These differences might be explained by the fact that walking has other body movements than the laying, sitting and standing activities.

Code exercise 4.1, 4.2 and 4.3

```
## Assignment 4.1, 4.2 and 4.3

import pandas as pd
import matplotlib.pyplot as plt
import random

print("[Exercise 4.1]")

# Declare file names
file_activity_labels = "UCI HAR Dataset/activity_labels.txt"
file_features = "UCI HAR Dataset/features.txt"
file_testX = "UCI HAR Dataset/test/X_test.txt"
file_testY = "UCI HAR Dataset/test/y_test.txt"
file_trainX = "UCI HAR Dataset/train/X_train.txt"
file_trainY = "UCI HAR Dataset/train/y_train.txt"

# Import data
activity_labels = pd.read_csv(file_activity_labels, delimiter=" ", header=None, names=['id', 'activity'])
features = pd.read_csv(file_features, delimiter=" ", header=None, names=['id', 'feature'])
test_X = pd.read_csv(file_testX, delimiter=" ", header=None, skipinitialspace=True)
test_Y = pd.read_csv(file_testY, delimiter=" ", header=None, skipinitialspace=True)
train_X = pd.read_csv(file_trainX, delimiter=" ", header=None, skipinitialspace=True)
train_Y = pd.read_csv(file_trainY, delimiter=" ", header=None, skipinitialspace=True)

# Add activity name column to labels
train_Y['label'] = train_Y[0].transform(lambda c: activity_labels['activity'][c - 1])
test_Y['label'] = test_Y[0].transform(lambda c: activity_labels['activity'][c - 1])

# Add labels to the measurements
train_X['label'] = train_Y['label']
test_X['label'] = test_Y['label']

# Combine training and test data
X = pd.concat([train_X, test_X], ignore_index=True)
```

```
35 Y = pd.concat([train_Y, test_Y], ignore_index=True)
36
37 ### 4.1a: dimensionality of the dataset
38 print("[4.1a: dimensionality]")
39 print("Training set :", train_X.shape)  # (7352, 561)
40 print("    Test set :", test_X.shape)  # (2947, 561)
41 print()
42
43 ### 4.1b: statistics
44 print("[4.1b: statistics]")
45 nFeatures = 6  # amount of features
46 # Calculate statistics for some features (columns)
47 for i in range(0, nFeatures):
48     print("Feature :", features['feature'][i])
49     print("   Mean : %0.3f" % X[i].mean())
50     print(" Median : %0.3f" % X[i].median())
51     print(" Stddev : %0.3f" % X[i].std())
52     print()
53
54 ### Exercise 4.2
55 print("[Exercise 4.2]")
56
57 ### 4.2a: dimensions
58 print("[4.2a: dimensionality]")
59 print("Training set :", train_Y.shape)
60 print("    Test set :", test_Y.shape)
61 print()
62
63 ### 4.2b: bar chart
64 print("[4.2b: bar chart]")
65 activity_counts = Y[0].value_counts(normalize=False).sort_index()  # count
       occurrence
66
67 # Make plot
68 plt.figure(figsize=(15, 6))
69 plt.xlabel("Activity")
70 plt.ylabel("Counts")
71 plt.bar(activity_labels['activity'], activity_counts)
72 plt.savefig("figures/4_2b.png", dpi=300)
73 print()
74
75 ### 4.3
76 print("[Exercise 4.3]")
77 grouped_X = train_X.groupby('label')
78 random.seed(9001)  # set random number seed to always generate same
      numbers
79 plt.figure(figsize=(6, 4))
80
81 for i in range(0, 10):
```

```
82      feature_id = random.randint(0, features.shape[0])  # choose random
            feature
83      grouped_X[feature_id].plot.kde()  # plot feature kde
84      plt.title(features['feature'].values[feature_id])
85      plt.legend(grouped_X.groups.keys())
86      plt.savefig("figures/4.3_%i.png" % feature_id, dpi=300)
```

Exercise 4.4

a) The training set contains 7352 rows and 128 columns and the test set contains 2947 rows and 128 columns. Each row therefore represents the measurements made in one window (sliding window of 2.56 sec and 50% overlap at a sample frequency of 50Hz result in 128 readings per window) for a certain activity.

b) Since each window (row) has 50% overlap with the next, the final half of the measurements of a certain row correspond to the first half of the measurements of the next. The overlap is visually explained in Figure 12.

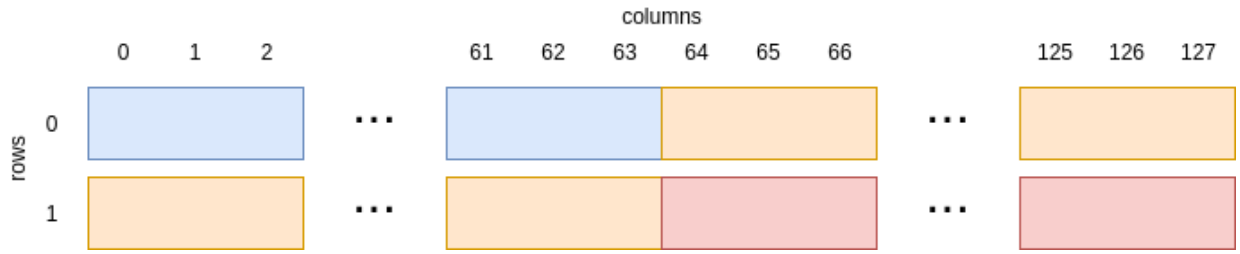

Figure 12: The 50% of overlap in the data explained. Each row consists of 128 columns in which the last 64 columns correspond to the first 64 columns of the next row (orange blocks).

Due to the overlap, some data points correspond to multiple activities. As a consequence, the training is less accurate since the difference between the activities is diminished. A solution is to either remove the first or second half of columns of each row. In this case it is chosen to remove the second half. With this remaining data, the original signal can be obtained by concatenating those data points.

Exercise 4.5

a) Features that are recreated from the raw signal are the mean, standard deviation and kurtosis.

b) The distributions of the mean, standard deviation and kurtosis are displayed in Figure 13, Figure 14 and Figure 15, respectively. At first sight, the distribution families of the activities per time domain feature seem similar. The means of the activities seem to be normally distributed, standard deviations seem to be bimodal and the kurtosis as well normally distributed, but lightly positively skewed. There are some differences in the densities.

The only differences that can be observed in the figures are differences the standard deviation.

As can be seen in Figure 13, the standing activity has a very low standard deviation (the very high peak implies a high density and hence low standard deviation) compared to the walking activity, where the peak is much lower and relatively a wider bell-shaped curve. However, this characteristic is not sufficient to conclude that activities can be discriminated between these selected time domain features.

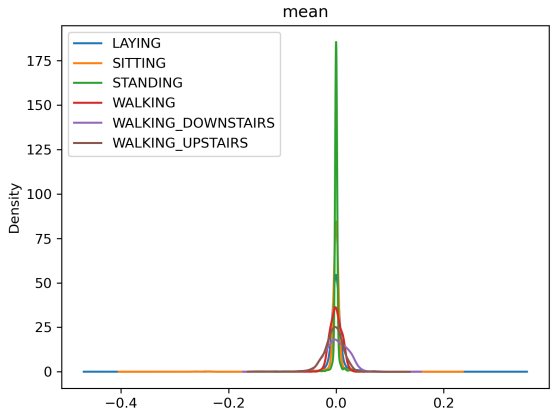

Figure 13: Distribution of the mean for all activities

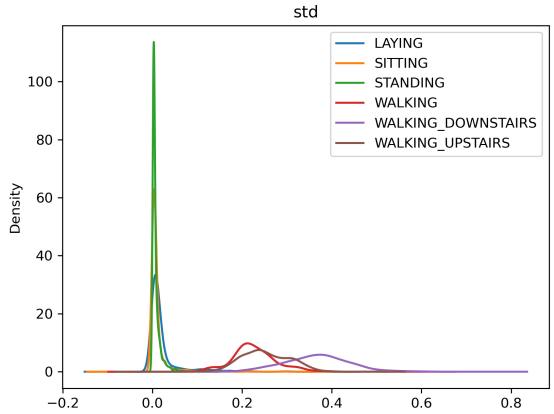

Figure 14: Distribution of the standard deviation for all activities

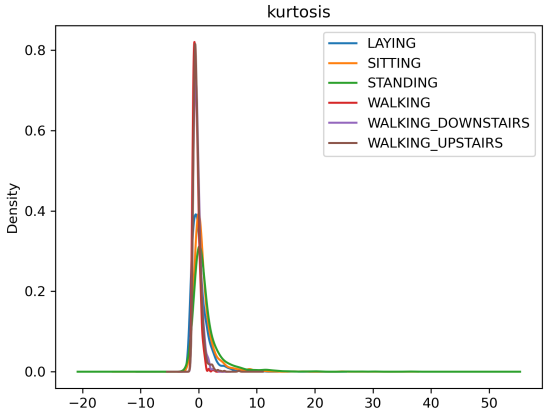

Figure 15: Distribution of the kurtosis for all activities

Exercise 4.6

a) The original signal and frequency spectra of the six activities are shown in Figures 16-21. The DC-offset was removed by subtracting the mean of the data from the data.

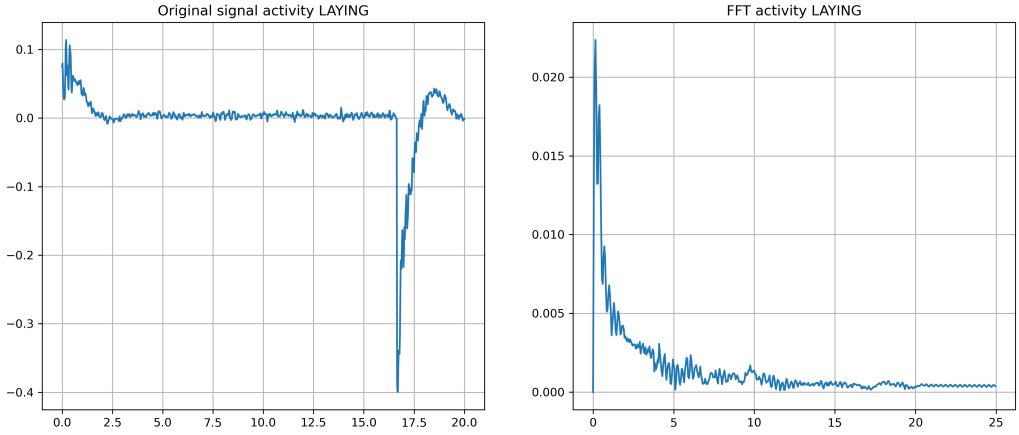

Figure 16: Original signal and frequency spectra for the laying activity

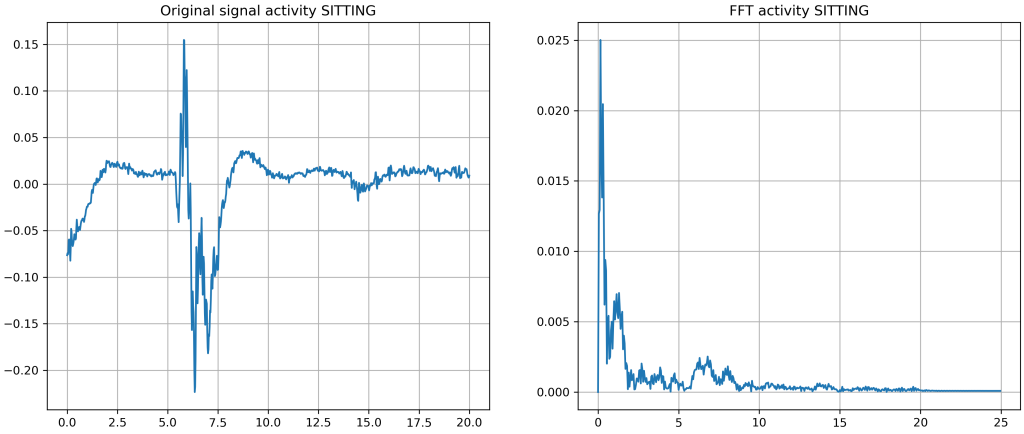

Figure 17: Original signal and frequency spectra for the sitting activity

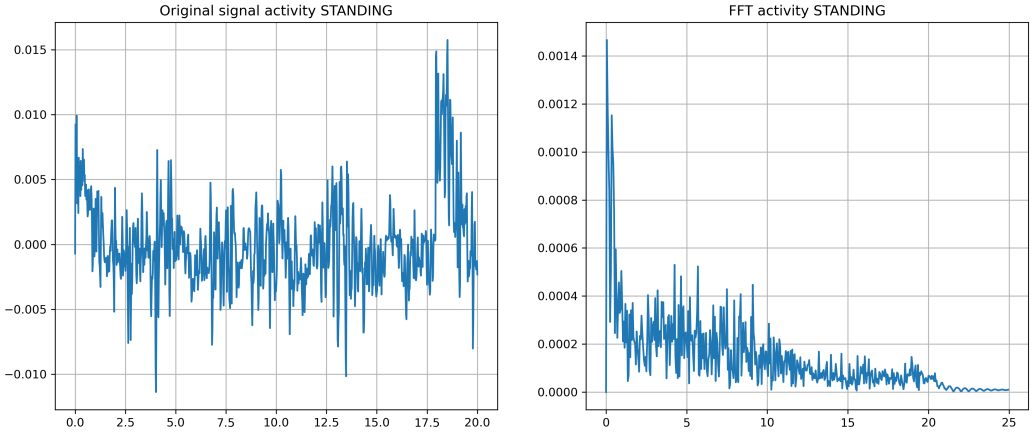

Figure 18: Original signal and frequency spectra for the standing activity

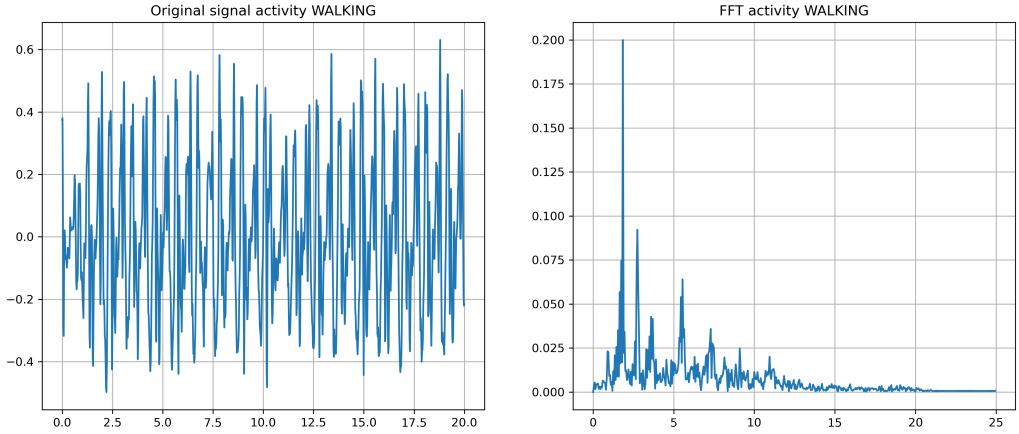

Figure 19: Original signal and frequency spectra for the walking activity

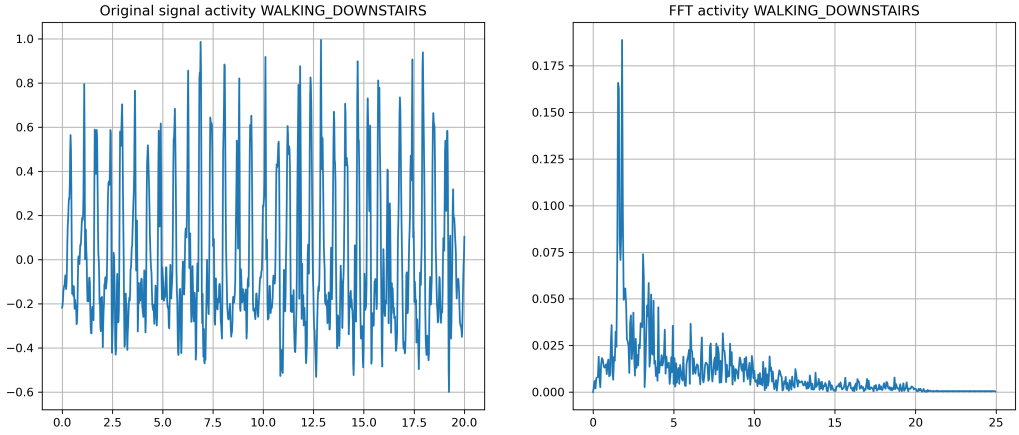

Figure 20: Original signal and frequency spectra for the walking downstairs activity

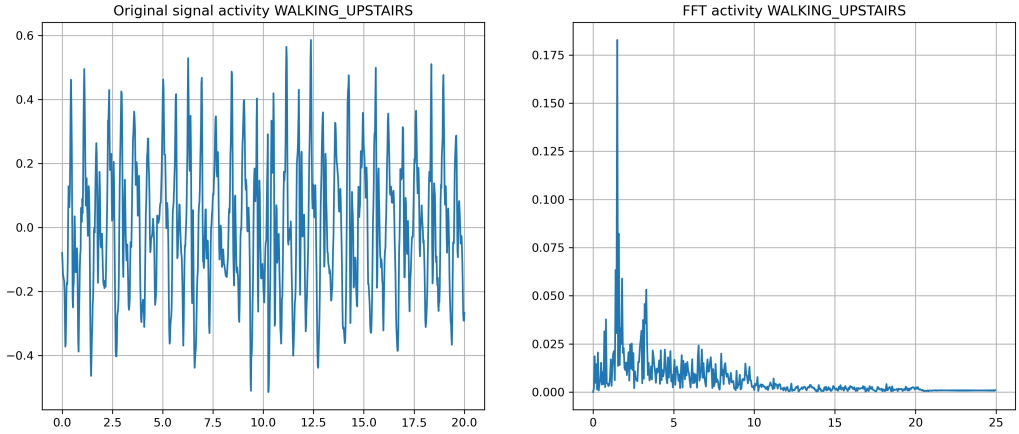

Figure 21: Original signal and frequency spectra for the walking upstairs activity

b) A clear difference can be observed in the spectra between the walking (Figures 16-18) and non-walking activities (Figures 19-21). The non-walking activities show a clear peak around 0 Hz and the amplitudes of higher frequencies are decreasing. The walking activities show a peak around 2 Hz and and one smaller peak around 3 Hz. So, the spectra can discriminate between the walking and non-walking activities.

Regarding the non-walking activities, the standing activity is also significantly different from laying and sitting. The amplitude of the frequency around 0 Hz for the standing activity is significantly lower than the same peak for sitting and laying. A side effect is that the the scale of the y-axis is smaller and causes that noise on the signal is better visible due to small scale.

Exercise 4.7

a) People walk with a frequency between 1.4-1.8Hz[1] and breathe around 15-18 times a minute [2], which corresponds to a frequency between 0.25-0.3Hz. The breathing is most likely only significant for the activities in which the user should be still, e.g. during laying, sitting and standing. During these activities, no movement is to be expected hence breathing is not taken into consideration. Therefore, as a lower limit, a frequency of 0.6Hz is used and as an upper limit a frequency of 3Hz.

b) The filtered and frequency spectra of filtered signals are shown in Figures 22-24.

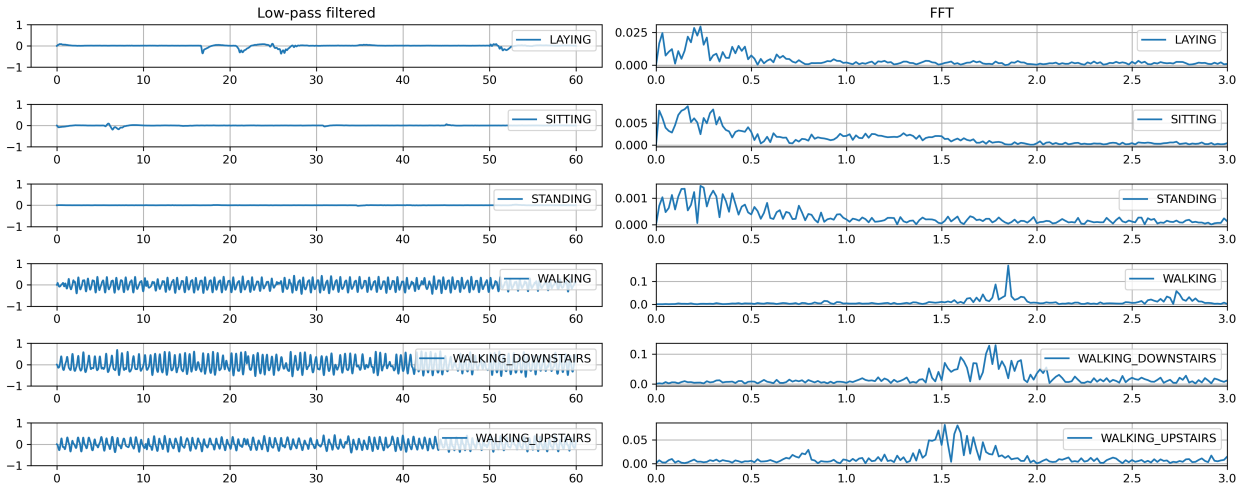

Figure 22: Results for the signal and corresponding frequency spectra when applying a low-pass filter.

[1]Pachi, A. & Ji, Tianjian. (2005). Frequency and velocity of people walking.The Structural Engineer, 83, pp. 36-40.

[2]Warliah, Rohman, Rusmin. (2012). Model Development of Air Volume and Breathing Frequency in Human Respiratory System Simulation. Procedia - Social and Behavioral Sciences, 67, pp. 260-268.

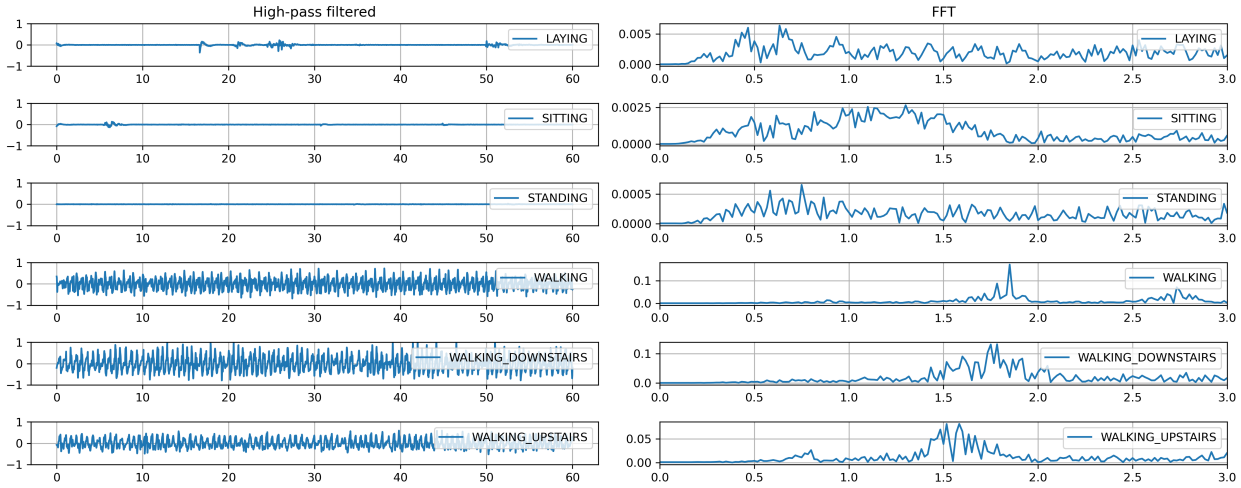

Figure 23: Results for the signal and corresponding frequency spectra when applying a high-pass filter.

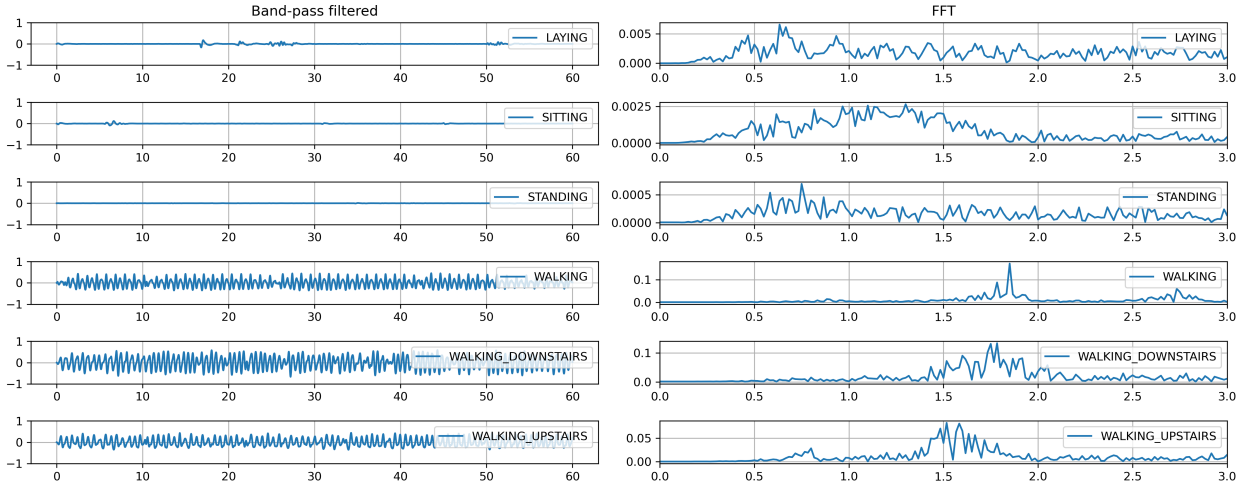

Figure 24: Results for the signal and corresponding frequency spectra when applying a band-pass filter.

The low-pass filter filters the higher frequencies from the movements. As a result, possible breathing disturbances are still present in the signal. Comparing to the band-pass filtered signal, this difference can be seen in the small peaks for frequencies between 0-1Hz for the walking activities. The high-pass filter filters the breathing artefacts but does not filter high frequency noise. Again comparing to the band-pass filtered signal, the difference can be seen in the higher frequencies, between 2-3Hz. The high-pass filtered signal shows slightly increased amplitude of the FFT, however the difference is minimal.

In conclusion, the band-pass filter is preferred since it filters both the low-frequency breathing

and possible high(er) frequency artefacts.

Code exercise 4.4, 4.5, 4.6 and 4.7

```
## Assignment 4.4, 4.5, 4.6 and 4.7

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.stats import kurtosis as kurt
from scipy.fftpack import fft
from scipy.signal import butter
from scipy.signal import lfilter

# Declare file names
file_activity_labels = "UCI HAR Dataset/activity_labels.txt"
file_features = "UCI HAR Dataset/features.txt"

file_train_total_acc_x = "UCI HAR Dataset/train/Inertial Signals/
    total_acc_x_train.txt"
file_train_total_acc_y = "UCI HAR Dataset/train/Inertial Signals/
    total_acc_y_train.txt"
file_train_total_acc_z = "UCI HAR Dataset/train/Inertial Signals/
    total_acc_z_train.txt"

file_test_total_acc_x = "UCI HAR Dataset/test/Inertial Signals/
    total_acc_x_test.txt"
file_test_total_acc_y = "UCI HAR Dataset/test/Inertial Signals/
    total_acc_y_test.txt"
file_test_total_acc_z = "UCI HAR Dataset/test/Inertial Signals/
    total_acc_z_test.txt"

# Import data
activity_labels = pd.read_csv(file_activity_labels, delimiter=" ", header=
    None, names=['id', 'activity'])
features = pd.read_csv(file_features, delimiter=" ", header=None, names=['
    id', 'feature'])

train_total_acc_x = pd.read_csv(file_train_total_acc_x, delimiter=" ",
    header=None, skipinitialspace=True)
train_total_acc_y = pd.read_csv(file_train_total_acc_y, delimiter=" ",
    header=None, skipinitialspace=True)
train_total_acc_z = pd.read_csv(file_train_total_acc_z, delimiter=" ",
    header=None, skipinitialspace=True)

test_total_acc_x = pd.read_csv(file_test_total_acc_x, delimiter=" ",
    header=None, skipinitialspace=True)
test_total_acc_y = pd.read_csv(file_test_total_acc_y, delimiter=" ",
    header=None, skipinitialspace=True)
```

```
33 test_total_acc_z = pd.read_csv(file_test_total_acc_z, delimiter=" ",
       header=None, skipinitialspace=True)
34
35 # Combine training and test data
36 total_acc_x = pd.concat([train_total_acc_x, test_total_acc_x])
37 total_acc_y = pd.concat([train_total_acc_y, test_total_acc_y])
38 total_acc_z = pd.concat([train_total_acc_z, test_total_acc_z])
39
40 ### 4.4a: dimensionality
41 print("[Exercise 4.4]")
42 print("[4.4a] Dimensionality")
43 print("Training set :", train_total_acc_x.shape)  # (7352, 128)
44 print("    Test set :", test_total_acc_x.shape)  # (2947, 128)
45 print()
46
47 ### 4.4b: reconstruct original signal
48 print("[4.4b] Variances")
49
50 # Calculate the variances of the accelerations of all 3 axes
51 x_var = total_acc_x.apply(lambda row: row.var(), axis=1).sum()
52 y_var = total_acc_y.apply(lambda row: row.var(), axis=1).sum()
53 z_var = total_acc_z.apply(lambda row: row.var(), axis=1).sum()
54 variances = [x_var, y_var, z_var]
55 greatest_var = ['x', 'y', 'z'][variances.index(max(variances))]
56
57 print("Variance x : %0.2f" % x_var)
58 print("Variance y : %0.2f" % y_var)
59 print("Variance z : %0.2f" % z_var)
60 print("Greatest variance : %s" % greatest_var)
61 print()
62
63 # Select the corresponding body acceleration file
64 file_test_body_acc = "UCI HAR Dataset/test/Inertial Signals/body_acc_%
       s_test.txt" % greatest_var
65 file_train_body_acc = "UCI HAR Dataset/train/Inertial Signals/body_acc_%
       s_train.txt" % greatest_var
66 file_train_Y = "UCI HAR Dataset/train/y_train.txt"
67
68 train_body_acc = pd.read_csv(file_train_body_acc, delimiter=" ", header=
       None, skipinitialspace=True)
69 train_Y = pd.read_csv(file_train_Y, delimiter=" ", header=None,
       skipinitialspace=True)
70
71 data = train_body_acc.loc[:, :63]  # remove half of the columns to solve
       overlap
72 raw_data = data.values  # convert dataframe to numpy array
73 raw_data = raw_data.flatten()  # flatten 2D dataset to 1D array
74 raw_labels = np.repeat(train_Y.values.flatten(), 64)  # obtain label per
       datapoint
75
```

```
76  # Couple the class labels with the raw data points
77  raw_signal = np.vstack((raw_data, raw_labels)).T
78
79  ### 4.5a: time domain features
80  print("[Exercise 4.5]")
81  # Calculate mean, std, kurtosis per window
82  mean, std, kurtosis, labels = [], [], [], []
83  for i in range(0, raw_data.size - 64, 64):
84      window = raw_data[i:i + 128]  # select the window
85      mean.append(window.mean())
86      std.append(window.std())
87      kurtosis.append(kurt(window))
88      labels.append(raw_labels[i])
89
90  ### 4.5b: feature plots
91  # Transform to dataframe for plotting
92  time_domain_features = pd.DataFrame(np.vstack((mean, std, kurtosis, labels
        )).T,
93                                      columns=["mean", "std", "kurtosis", "
                                            label"])
94  # Transform labels to strings
95  time_domain_features["label"] = time_domain_features["label"].transform(
        lambda c: activity_labels['activity'][c - 1])
96  grouped_features = time_domain_features.groupby("label")
97  for feature in ["mean", "std", "kurtosis"]:
98      plt.figure()
99      grouped_features[feature].plot.kde()
100     plt.title(feature)
101     plt.legend(grouped_features.groups.keys())
102     plt.savefig("figures/4.5_%s.png" % feature, dpi=300)
103
104 print("[4.5] Plot saved")
105 print()
106
107 ### 4.6
108 print("[Exercise 4.6]")
109
110
111 # Function to calculate the fft
112 def calculate_fft(x, T):
113     # x: list of signal points, T: sample time
114     s = np.ceil(np.size(x) / 2)
115     y = (2 / np.size(x)) * fft(x)  # normalise to get proper amplitude
116     y = y[0:int(s)]  # take half so the negative part is not used
117     ym = abs(y)  # magnitude
118     f = np.arange(0, s)
119     fspacing = 1 / (np.size(x) * T)
120     f = fspacing * f  # frequency axis
121     return f, ym
122
```

```
123
124 ### 4.6a: fft of all 6 activities
125 fs = 50  # sample frequency
126 T = 1 / fs  # sample time
127 nsamples = 1000  # amount of samples to transform
128
129 # Create dictionary for the activities
130 raw_signal = pd.DataFrame(raw_signal, columns=["signal", "label"])  #
        convert to pandas dataframe
131 raw_signal["label"] = raw_signal["label"].transform(lambda c:
        activity_labels['activity'][c - 1])  # add labels
132 activities_and_signals = raw_signal.groupby("label")["signal"].apply(list)
        .to_dict()  # make dictionary
133
134 for activity, signal in activities_and_signals.items():
135     signal = signal[0:nsamples]  # take nsamples from the signal
136     signal = signal - np.mean(signal)  # remove DC-offset
137     t = np.linspace(start=0, stop=T * np.size(signal), num=np.size(signal)
            )  # create time axis
138     f, y = calculate_fft(signal, T)  # fourier transform
139
140     # Plot
141     plt.figure(figsize=(15, 6))
142     plt.subplot(1, 2, 1)
143     plt.plot(t, signal)  # original signal
144     plt.grid(True)
145     plt.title("Original signal activity %s" % activity)
146     plt.ylim(-1, 1)
147
148     plt.subplot(1, 2, 2)
149     plt.plot(f, y)  # fft
150     plt.grid(True)
151     plt.title("FFT activity %s" % activity)
152     plt.savefig("figures/4.6_%s.png" % activity, dpi=300)
153
154 print("[4.6] Plots saved")
155 print()
156
157 ### 4.7
158 print("[Exercise 4.7]")
159
160 nsamples = 3000  # amount of samples to transform
161 fn = fs / 2  # nyquist frequency
162 upper_limit = 3 / fn  # upper limit cutoff frequency
163 lower_limit = 0.6 / fn  # lower limit cutoff frequency
164 filter_order = 4  # order of the filters
165
166 ## Low-pass filter
167 b_lp, a_lp = butter(filter_order, upper_limit, btype='lowpass', output='ba
        ')
```

```
168 i = 1
169
170 plt.figure(figsize=(15, 6))
171 for activity, signal in activities_and_signals.items():
172     signal = signal[0:nsamples]
173     signal = signal - np.mean(signal)
174     signal = lfilter(b_lp, a_lp, signal)
175     t = np.linspace(start=0, stop=T * np.size(signal), num=np.size(signal)
        )  # create time axis
176     f, y = calculate_fft(signal, T)  # fourier transform
177
178     # Plot filtered signal
179     plt.subplot(6, 2, i)
180     plt.grid(True)
181     plt.plot(t, signal, label=activity)  # original signal
182     plt.ylim(-1, 1)
183     plt.legend(loc="upper right")
184     if i == 1: plt.title("Low-pass filtered")
185     i += 1
186
187     # Plot frequency spectrum
188     plt.subplot(6, 2, i)
189     plt.plot(f, y, label=activity)  # fft
190     plt.grid(True)
191     plt.xlim(0, 3)
192     plt.legend(loc="upper right")
193     if i == 2: plt.title("FFT")
194     i += 1
195 plt.tight_layout()
196 plt.savefig("figures/4.7_lowpass.png", dpi=300)
197 plt.close()
198
199 ## High-pass filter
200 b_hp, a_hp = butter(filter_order, lower_limit, btype='highpass', output='
    ba')
201 i = 1
202
203 plt.figure(figsize=(15, 6))
204 for activity, signal in activities_and_signals.items():
205     signal = signal[0:nsamples]  # take nsamples from the signal
206     signal = signal - np.mean(signal)  # remove DC-offset
207     signal = lfilter(b_hp, a_hp, signal)  # filter signal highpass
208     t = np.linspace(start=0, stop=T * np.size(signal), num=np.size(signal)
        )  # create time axis
209     f, y = calculate_fft(signal, T)  # fourier transform
210
211     # Plot filtered signal
212     plt.subplot(6, 2, i)
213     plt.grid(True)
214     plt.plot(t, signal, label=activity)  # original signal
```

```
215     plt.ylim(-1, 1)
216     plt.legend(loc="upper right")
217     if i == 1: plt.title("High-pass filtered")
218     i += 1
219 
220     # Plot frequency spectrum
221     plt.subplot(6, 2, i)
222     plt.plot(f, y, label=activity)  # fft
223     plt.grid(True)
224     plt.xlim(0, 3)
225     plt.legend(loc="upper right")
226     if i == 2: plt.title("FFT")
227     i += 1
228 plt.tight_layout()
229 plt.savefig("figures/4.7_highpass.png", dpi=300)
230 plt.close()
231 
232 ## Band-pass filter
233 b_bp, a_bp = butter(filter_order, [lower_limit, upper_limit], btype='
        bandpass', output='ba')
234 i = 1
235 
236 plt.figure(figsize=(15, 6))
237 for activity, signal in activities_and_signals.items():
238     signal = signal[0:nsamples]
239     signal = signal - np.mean(signal)
240     signal = lfilter(b_bp, a_bp, signal)
241     t = np.linspace(start=0, stop=T * np.size(signal), num=np.size(signal)
            )  # create time axis
242     f, y = calculate_fft(signal, T)  # fourier transform
243 
244     # Plot filtered signal
245     plt.subplot(6, 2, i)
246     plt.grid(True)
247     plt.plot(t, signal, label=activity)  # original signal
248     plt.ylim(-1, 1)
249     plt.legend(loc="upper right")
250     if i == 1: plt.title("Band-pass filtered")
251     i += 1
252 
253     # Plot frequency spectrum
254     plt.subplot(6, 2, i)
255     plt.plot(f, y, label=activity)  # fft
256     plt.grid(True)
257     plt.xlim(0, 3)
258     plt.legend(loc="upper right")
259     if i == 2: plt.title("FFT")
260     i += 1
261 plt.tight_layout()
262 plt.savefig("figures/4.7_bandpass.png", dpi=300)
```

```
263 plt.close()
264
265 print("[4.7] Plots saved")
266 print()
```

Exercise 4.8

For the raw data, only the total acceleration data is used, as was recommended by one of the TAs. In the figures and tables below, the confusion matrix and summaries are shown for the feature and raw data.

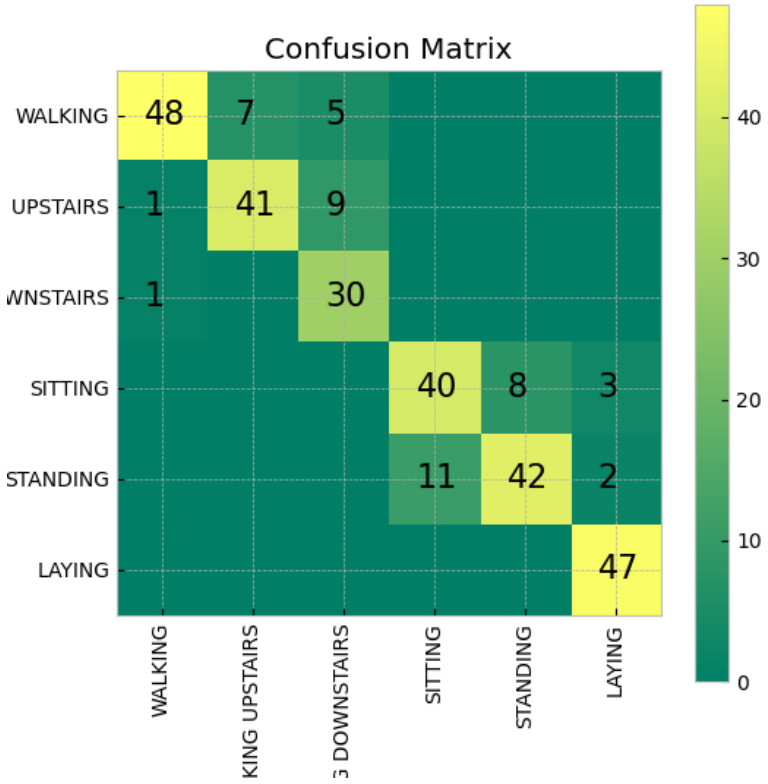

Figure 25: Confusion matrix of KNN DTW on feature data.

	Precision	Recall	f1-score	Support
WALKING	0.96	0.80	0.87	60
WALKING UPSTAIRS	0.85	0.80	0.83	51
WALKING DOWNSTAIRS	0.68	0.97	0.80	31
SITTING	0.78	0.78	0.78	51
STANDING	0.84	0.76	0.80	55
LAYING	0.90	1.00	0.95	47

Table 2: Summary of the confusion matrix of KNN DTW on feature data.

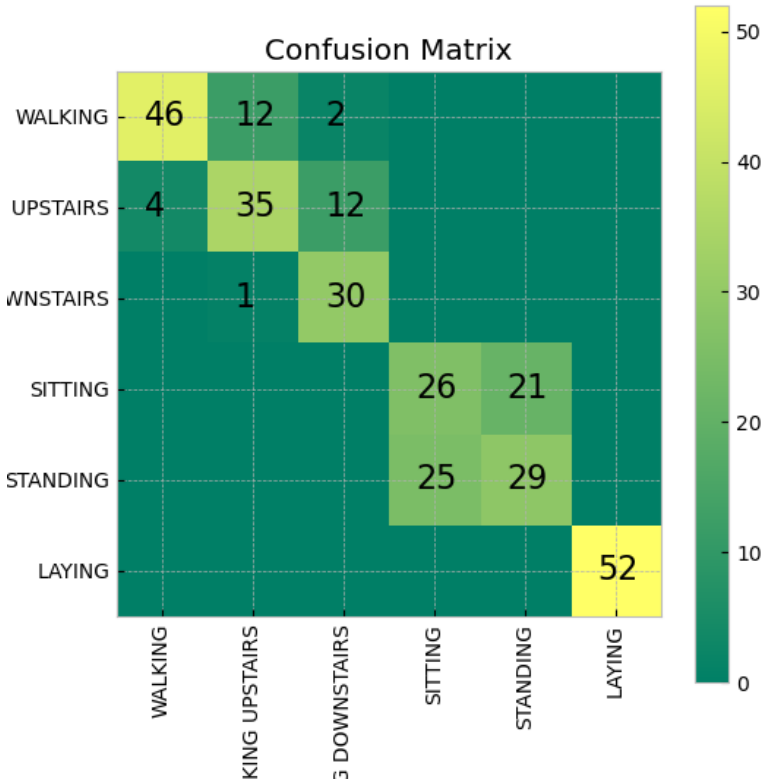

Figure 26: Confusion matrix of KNN DTW on raw data (total body acceleration in x).

	Precision	Recall	f1-score	Support
WALKING	0.92	0.77	0.84	60
WALKING UPSTAIRS	0.73	0.69	0.71	51
WALKING DOWNSTAIRS	0.68	0.97	0.80	31
SITTING	0.58	0.54	0.56	47
STANDING	0.58	0.54	0.56	54
LAYING	1.00	1.00	1.00	52

Table 3: Summary of the confusion matrix of KNN DTW on raw data (total body acceleration in x). Red indicates that the performance is decreased compared to the feature data (Table 2, and green that the performance is increased.

DTW performs worse on the raw data compared to the feature data for the walking, walking upstairs, sitting and standing activities. It performs equally for walking downstairs and better for laying. DTW perfoms worse on this dataset due to the small amount of data. In this case, only the raw total acceleration data in the x-direction is used. Depending on the orientation of the sensor, it is possible that there is little variation in the x-direction during walking, walking upstairs, sitting and standing. This could also explain why algorithm performs better on the laying activity, since this activity requires no movement. In that case, it would be odd that sitting and standing perform worse, since these activities also require no movement. It is possible that these activities are classified as laying, due to the lack of information.

Code:

```
## Assignment 4.8

import sys
import collections
import itertools
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import mode
from scipy.spatial.distance import squareform
from sklearn.metrics import classification_report, confusion_matrix

print("[Exercise 4.8]")

plt.style.use('bmh')

try:
    from IPython.display import clear_output
    have_ipython = True
except ImportError:
    have_ipython = False

class KnnDtw(object):
    """K-nearest neighbor classifier using dynamic time warping
    as the distance measure between pairs of time series arrays
```

```
25
26      Arguments
27      ---------
28      n_neighbors : int, optional (default = 5)
29          Number of neighbors to use by default for KNN
30
31      max_warping_window : int, optional (default = infinity)
32          Maximum warping window allowed by the DTW dynamic
33          programming function
34
35      subsample_step : int, optional (default = 1)
36          Step size for the timeseries array. By setting subsample_step = 2,
37          the timeseries length will be reduced by 50% because every second
38          item is skipped. Implemented by x[:, ::subsample_step]
39      """
40
41      def __init__(self, n_neighbors=5, max_warping_window=10000,
            subsample_step=1):
42          self.n_neighbors = n_neighbors
43          self.max_warping_window = max_warping_window
44          self.subsample_step = subsample_step
45
46      def fit(self, x, l):
47          """Fit the model using x as training data and l as class labels
48
49          Arguments
50          ---------
51          x : array of shape [n_samples, n_timepoints]
52              Training data set for input into KNN classifer
53
54          l : array of shape [n_samples]
55              Training labels for input into KNN classifier
56          """
57
58          self.x = x
59          self.l = l
60
61      def _dtw_distance(self, ts_a, ts_b, d=lambda x, y: abs(x - y)):
62          """Returns the DTW similarity distance between two 2-D
63          timeseries numpy arrays.
64
65          Arguments
66          ---------
67          ts_a, ts_b : array of shape [n_samples, n_timepoints]
68              Two arrays containing n_samples of timeseries data
69              whose DTW distance between each sample of A and B
70              will be compared
71
72          d : DistanceMetric object (default = abs(x-y))
73              the distance measure used for A_i - B_j in the
```

```
74              DTW dynamic programming function
75
76          Returns
77          -------
78          DTW distance between A and B
79          """
80
81          # Create cost matrix via broadcasting with large int
82          ts_a, ts_b = np.array(ts_a), np.array(ts_b)
83          M, N = len(ts_a), len(ts_b)
84          cost = sys.maxsize * np.ones((M, N))
85
86          # Initialize the first row and column
87          cost[0, 0] = d(ts_a[0], ts_b[0])
88          for i in range(1, M):
89              cost[i, 0] = cost[i - 1, 0] + d(ts_a[i], ts_b[0])
90
91          for j in range(1, N):
92              cost[0, j] = cost[0, j - 1] + d(ts_a[0], ts_b[j])
93
94          # Populate rest of cost matrix within window
95          for i in range(1, M):
96              for j in range(max(1, i - self.max_warping_window),
97                             min(N, i + self.max_warping_window)):
98                  choices = cost[i - 1, j - 1], cost[i, j - 1], cost[i - 1,
                        j]
99                  cost[i, j] = min(choices) + d(ts_a[i], ts_b[j])
100
101         # Return DTW distance given window
102         return cost[-1, -1]
103
104     def _dist_matrix(self, x, y):
105         """Computes the M x N distance matrix between the training
106         dataset and testing dataset (y) using the DTW distance measure
107
108         Arguments
109         ---------
110         x : array of shape [n_samples, n_timepoints]
111
112         y : array of shape [n_samples, n_timepoints]
113
114         Returns
115         -------
116         Distance matrix between each item of x and y with
117             shape [training_n_samples, testing_n_samples]
118         """
119
120         # Compute the distance matrix
121         dm_count = 0
122
```

```
123         # Compute condensed distance matrix (upper triangle) of pairwise
                dtw distances
124         # when x and y are the same array
125         if (np.array_equal(x, y)):
126             x_s = np.shape(x)
127             dm = np.zeros((x_s[0] * (x_s[0] - 1)) // 2, dtype=np.double)
128 
129             p = ProgressBar(dm.shape[0])
130 
131             for i in range(0, x_s[0] - 1):
132                 for j in range(i + 1, x_s[0]):
133                     dm[dm_count] = self._dtw_distance(x[i, ::self.
                            subsample_step],
134                                                       y[j, ::self.
                                                          subsample_step])
135 
136                     dm_count += 1
137                     p.animate(dm_count)
138 
139             # Convert to squareform
140             dm = squareform(dm)
141             return dm
142 
143         # Compute full distance matrix of dtw distances between x and y
144         else:
145             x_s = np.shape(x)
146             y_s = np.shape(y)
147             dm = np.zeros((x_s[0], y_s[0]))
148             dm_size = x_s[0] * y_s[0]
149 
150             p = ProgressBar(dm_size)
151 
152             for i in range(0, x_s[0]):
153                 for j in range(0, y_s[0]):
154                     dm[i, j] = self._dtw_distance(x[i, ::self.
                            subsample_step],
155                                                   y[j, ::self.
                                                      subsample_step])
156                     # Update progress bar
157                     dm_count += 1
158                     p.animate(dm_count)
159 
160             return dm
161 
162     def predict(self, x):
163         """Predict the class labels or probability estimates for
164         the provided data
165 
166         Arguments
167         ---------
```

```
168             x : array of shape [n_samples, n_timepoints]
169                 Array containing the testing data set to be classified
170
171         Returns
172         -------
173             2 arrays representing:
174                 (1) the predicted class labels
175                 (2) the knn label count probability
176         """
177
178         dm = self._dist_matrix(x, self.x)
179
180         # Identify the k nearest neighbors
181         knn_idx = dm.argsort()[:, :self.n_neighbors]
182
183         # Identify k nearest labels
184         knn_labels = self.l[knn_idx]
185
186         # Model Label
187         mode_data = mode(knn_labels, axis=1)
188         mode_label = mode_data[0]
189         mode_proba = mode_data[1] / self.n_neighbors
190
191         return mode_label.ravel(), mode_proba.ravel()
192
193
194 class ProgressBar:
195     """This progress bar was taken from PYMC
196     """
197
198     def __init__(self, iterations):
199         self.iterations = iterations
200         self.prog_bar = '[]'
201         self.fill_char = '*'
202         self.width = 40
203         self.__update_amount(0)
204         if have_ipython:
205             self.animate = self.animate_ipython
206         else:
207             self.animate = self.animate_noipython
208
209     def animate_ipython(self, iter):
210         print('\r', self, end="", flush=True)
211         sys.stdout.flush()
212         self.update_iteration(iter + 1)
213
214     def update_iteration(self, elapsed_iter):
215         self.__update_amount((elapsed_iter / float(self.iterations)) *
                100.0)
216         self.prog_bar += '  %d of %s complete' % (elapsed_iter, self.
```

```
            iterations)

    def __update_amount(self, new_amount):
        percent_done = int(round((new_amount / 100.0) * 100.0))
        all_full = self.width - 2
        num_hashes = int(round((percent_done / 100.0) * all_full))
        self.prog_bar = '[' + self.fill_char * num_hashes + ' ' * (
            all_full - num_hashes) + ']'
        pct_place = (len(self.prog_bar) // 2) - len(str(percent_done))
        pct_string = '%d%%' % percent_done
        self.prog_bar = self.prog_bar[0:pct_place] + \
                        (pct_string + self.prog_bar[pct_place + len(
                            pct_string):])

    def __str__(self):
        return str(self.prog_bar)


# Import the HAR dataset
total_acc_x_train_file = open('UCI HAR Dataset/train/Inertial Signals/
    total_acc_x_train.txt')
y_train_file = open('UCI HAR Dataset/train/y_train.txt', 'r')

total_acc_x_test_file = open('UCI HAR Dataset/test/Inertial Signals/
    total_acc_x_test.txt')
y_test_file = open('UCI HAR Dataset/test/y_test.txt', 'r')

# Create empty lists
x_train = []
y_train = []
x_test = []
y_test = []

# Mapping table for classes
labels = {1: 'WALKING', 2: 'WALKING UPSTAIRS', 3: 'WALKING DOWNSTAIRS',
          4: 'SITTING', 5: 'STANDING', 6: 'LAYING'}

# Loop through datasets
for x in total_acc_x_train_file:
    x_train.append([float(ts) for ts in x.split()])

for y in y_train_file:
    y_train.append(int(y.rstrip('\n')))

for x in total_acc_x_test_file:
    x_test.append([float(ts) for ts in x.split()])

for y in y_test_file:
    y_test.append(int(y.rstrip('\n')))

```

```
# Convert to numpy for efficiency
x_train = np.array(x_train)
y_train = np.array(y_train)
x_test = np.array(x_test)
y_test = np.array(y_test)

interval = 10
m = KnnDtw(n_neighbors=1, max_warping_window=10)
m.fit(x_train[::interval], y_train[::interval])
label, proba = m.predict(x_test[::interval])

print(classification_report(label, y_test[::interval], target_names=[l for
    l in labels.values()]))

conf_mat = confusion_matrix(label, y_test[::interval])

fig = plt.figure(figsize=(6, 6))
width = np.shape(conf_mat)[1]
height = np.shape(conf_mat)[0]

res = plt.imshow(np.array(conf_mat), cmap=plt.cm.summer, interpolation='
   nearest')
for i, row in enumerate(conf_mat):
    for j, c in enumerate(row):
        if c > 0:
            plt.text(j - .2, i + .1, c, fontsize=16)

cb = fig.colorbar(res)
plt.title('Confusion Matrix')
_ = plt.xticks(range(6), [l for l in labels.values()], rotation=90)
_ = plt.yticks(range(6), [l for l in labels.values()])
plt.show()
plt.savefig("figures/4.8_rawx.png", dpi=300)

print("[4.8] Finished")

#  [****************100%******************]  217120 of 217120 complete
#                         precision recall    f1-score    support
#             WALKING       0.96      0.80      0.87        60
#    WALKING UPSTAIRS       0.85      0.80      0.83        51
#  WALKING DOWNSTAIRS       0.68      0.97      0.80        31
#             SITTING       0.78      0.78      0.78        51
#            STANDING       0.84      0.76      0.80        55
#              LAYING       0.90      1.00      0.95        47
#
#           accuracy                           0.84       295
#          macro avg       0.84      0.85      0.84       295
#       weighted avg       0.85      0.84      0.84       295

#  [****************100%******************]  217120 of 217120 complete
```

```
310
311 #                       precision recall    f1-score    support
312 #            WALKING         0.92      0.77      0.84         60
313 #   WALKING UPSTAIRS         0.73      0.69      0.71         51
314 # WALKING DOWNSTAIRS         0.68      0.97      0.80         31
315 #             SITTING         0.51      0.55      0.53         47
316 #            STANDING         0.58      0.54      0.56         54
317 #              LAYING         1.00      1.00      1.00         52
318 #
319 #            accuracy                           0.74        295
320 #           macro avg         0.74      0.75      0.74        295
321 #        weighted avg         0.75      0.74      0.74        295
```

Exercise 4.9

a) The dataset contains `NaN` for some temperatures for the given countries. It was determined that no more `NaN` occur after 1862-12-01. To show the behaviour of the temperatures clearly over the months in a plot, it is chosen to only plot the 20 years after this point. The plot is shown in Figure 27. All temperature data from this date onward was used in DTW. The resulting table with minima is shown in Table 4.

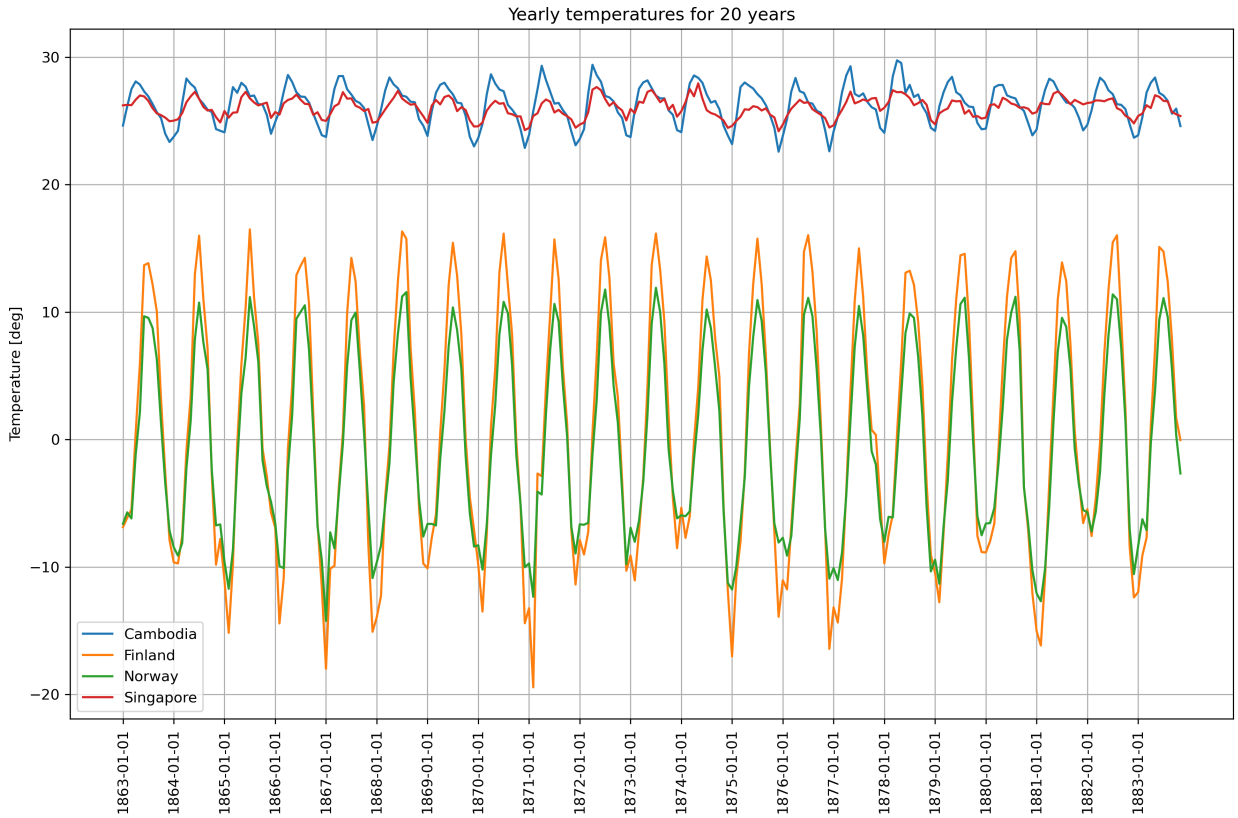

Figure 27: Temperatures over time for various countries.

	Norway	**Finland**	**Singapore**	**Cambodia**
Norway	0	4046	47527	47779
Finland	4046	0	45403	45655
Singapore	47527	45403	0	1283
Cambodia	47779	45655	1283	0

Table 4: Minimum distance in the temperatures between the countries using DTW.

From the plot it can be seen that the temperatures of Singapore and Cambodia are similar, and that the temperatures of Finland and Norway are similar. Finland and Norway show a large variance, with temperatures well below zero during winter and around 10 degrees in summer, while Singapore and Cambodia vary slightly around approximately 27 degrees. Therefore, it is expected that the distance between the similar countries are small (Finland–Norway and Singapore-Cambodia) and the distance between the Scandinavian and Asian countries are large. This expectation is supported by the DTW table. The distance between Finland and Norway is relatively small (4046) compared to the distance between for example Norway and Singapore (47527).

b) The results of the Dickey-Fuller test are shown in the figures and table below.

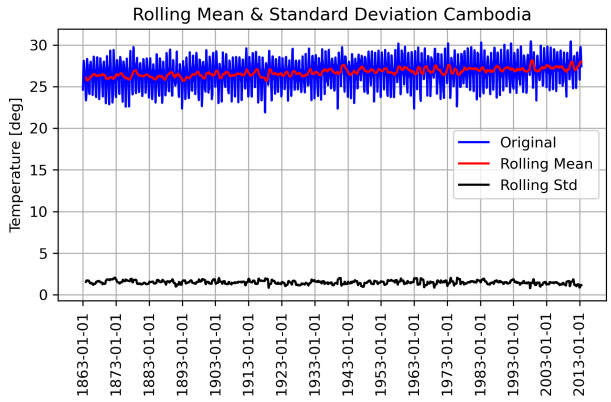

Figure 28: Dickey-Fuller test results for Cambodia.

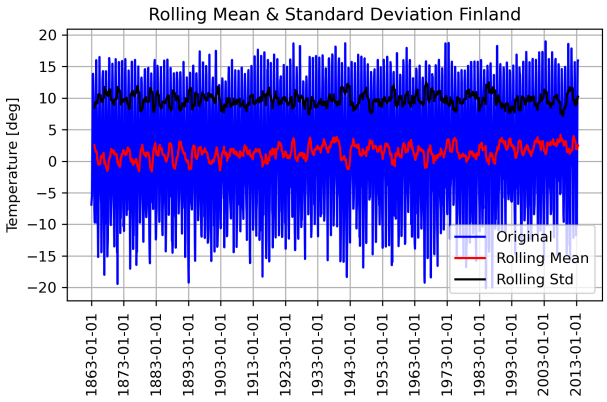

Figure 29: Dickey-Fuller test results for Finland.

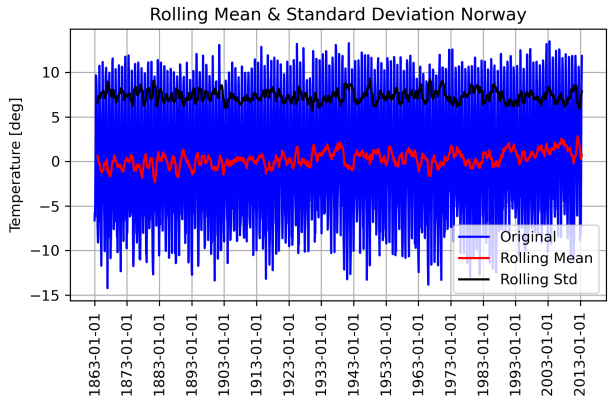

Figure 30: Dickey-Fuller test results for Norway.

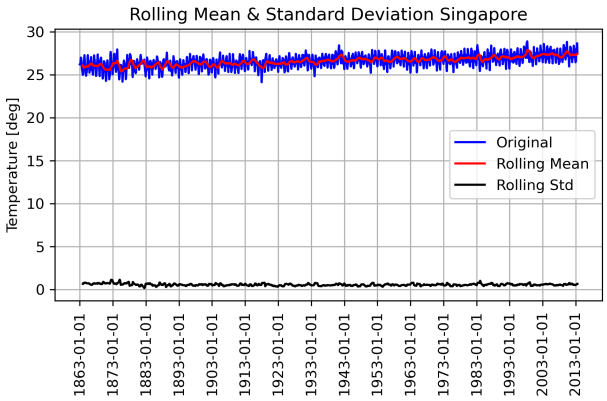

Figure 31: Dickey-Fuller test results for Singapore.

Test result	**Cambodia**	**Finland**	**Norway**	**Singapore**
Test Statistic	-3.832228	-5.684715	-4.526201	-3.031845
p-value	0.002594	8.330309e-07	0.000176	0.032026
#Lags used	25	24	24	25
Number of observations	1781	1782	1782	1781
Critical value (1%)	-3.434027	-3.434025	-3.434025	-3.434027
Critical value (5%)	-2.863164	-2.863163	-2.863163	-2.863164
Critical value (10%)	-2.567635	-2.567634	-2.567634	-2.567635

Table 5: Dickey-Fuller test results.

Regarding the figures, we can state that the series is stationary is the mean and variance are constant. In th plots it is seen that there is a slight increase in the mean over the years. The variances seem

somewhat constant. The increase in the mean could indicate that the time series are non-stationary.

In the Dickey-Fuller test, the null hypothesis is that the time series is non-stationary. If the test statistic is less than the critical value, the null-hypothesis can be rejected and it can be stated that the time series is stationary.

In Table 5 it is seen that the test statistic is lower than the 1% critical value for Cambodia, Finland and Norway. Hence, these temperature time series can be considered stationary with 99% confidence. For Singapore, the test statistic is lower than the 5% critical value, hence these time series can be considered with 95% confidence. The differences between the test statistic and critical values are however small, such that the conclusions must be taken with caution.

Additionally, we can take a look at the p-values. If the p-value is smaller than 0.06, the null-hypothesis can be rejected and the time series is considered stationary. In the table we can see that all p-values are below 0.05, hence the temperature time series can be considered stationary.

c) The new minima are shown in Table 6.

	Norway	**Finland**	**Singapore**	**Cambodia**
Norway	0	1437	1650	1454
Finland	1437	0	2565	2335
Singapore	1650	2565	0	421
Cambodia	1454	2335	421	0

Table 6: Minimum distance in the temperatures between the countries using DTW, after decomposing.

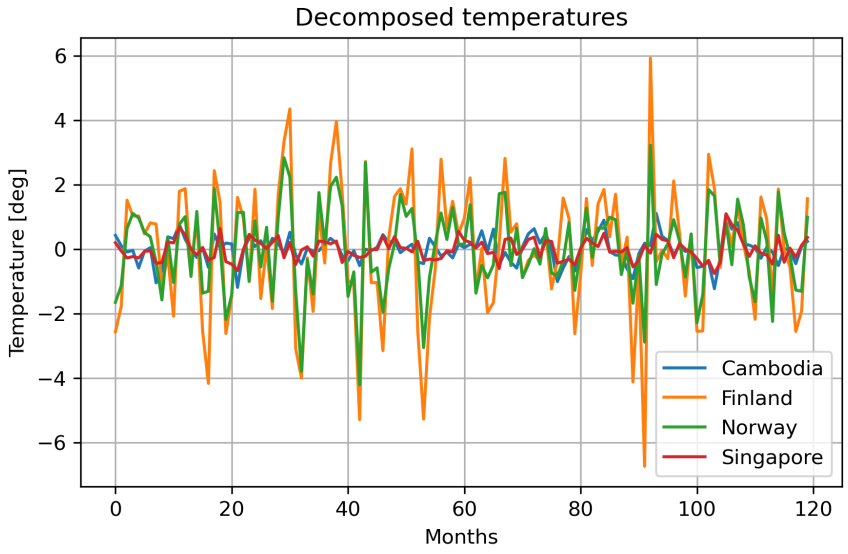

Figure 32: Decomposed temperature time series over a time period of 10 years.

By de-trending and removing seasonality, the time series are all located around zero (see Figure 32). As a result the countries that were already similar (Finland–Norway and Singapore–Cambodia) are even more similar. Additionally, the countries that were not similar are closer together, since their mean is now located around zero. The difference between these countries, for example Finland and Cambodia, are still greater than the distance between the similar countries due to the high variance in the Scandinavian countries. This is supported by the distances shown in Table 6. The distances between countries similar in temperature is still smaller than the distances between non-similar countries. All distances are smaller than before owing to the shift resulting in the time series being closer together.

d) The ACF and PACF charts are shown in Figure 33. To create these graphs, the difference data was used from the temperature data of Norway. 5 years were subtracted from the end of the dataset. These years are used to compare with the predicted temperatures in part e).

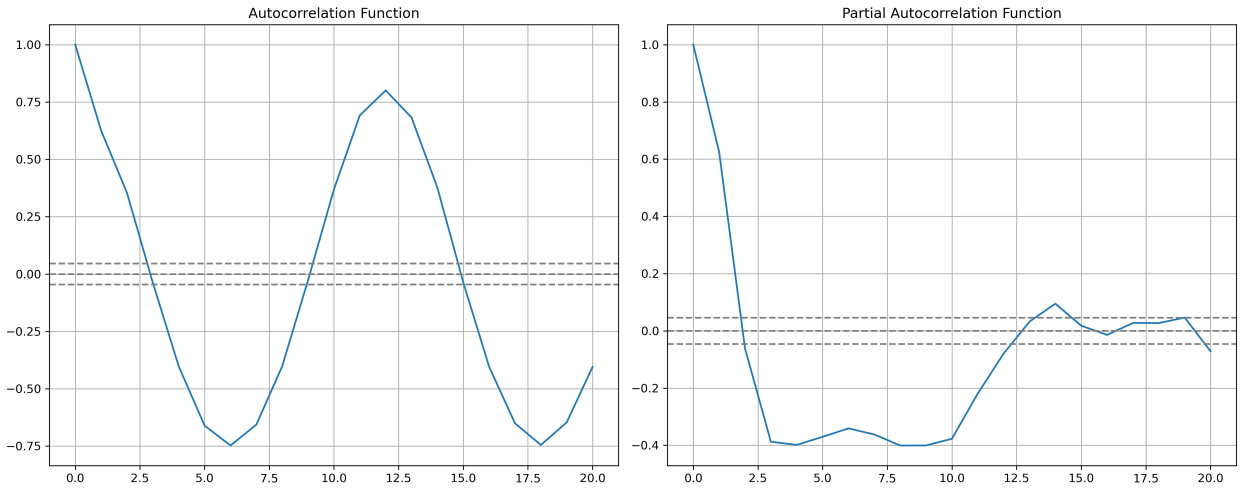

Figure 33: ACF and PACF plots for the decomposed data of Norway.

p is the lag value where the PACF chart crosses the upper confidence interval for the first time and q is the lag value where the ACF chart crosses the upper confidence interval for the first time. As can be derived from the figures, p equals 2 and q equals 3. The value for d was set to zero since a first order difference was used. The resulting AR, MA and ARIMA models are shown in Figures 34, 35 and 36, respectively. The model and data were only plotted for 10 years to clearly see the results.

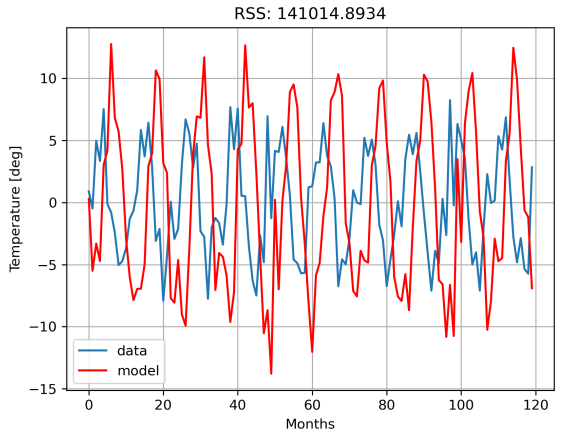

Figure 34: Result of the AR model on the temperature data of Norway, using p=2 and d=0.

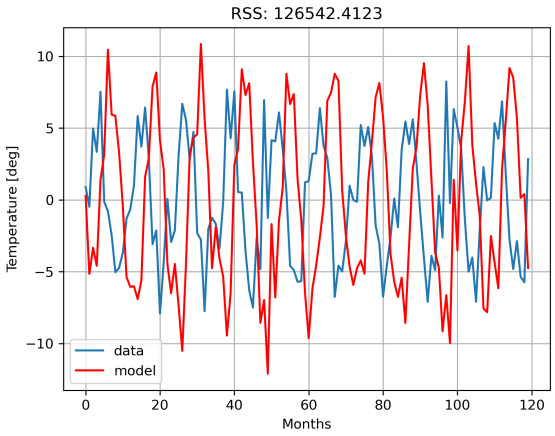

Figure 35: Result of the MA model on the temperature data of Norway, using q=3 and d=0.

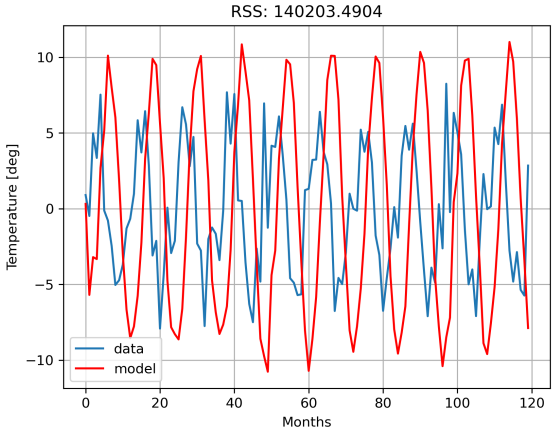

Figure 36: Result of the ARIMA model on the temperature data of Norway, using p=2, q=3 and d=0.

The models seem to correctly follow the shape of the data, but fall short in terms of lag and amplitude. The ARIMA model shows less noisy behaviour than the AR and MA model. Comparing the residual sum of squares (RSS), the best fit results from the ARIMA model.

e) The ARIMA model was chosen because this model gave the best results. The prediction was made for 5 years. The result is shown in Figure 37.

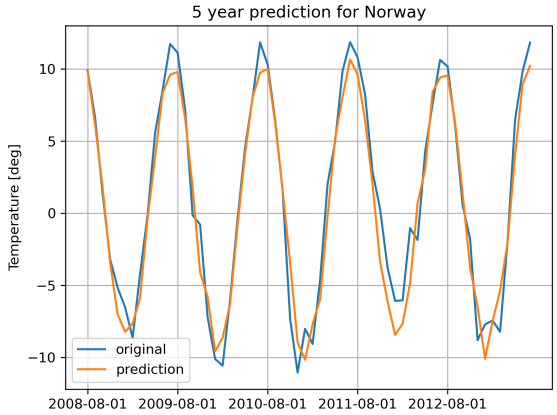

Figure 37: Original and predicted temperature data for Norway using the ARIMA model.

The original and predicted data are similar. The rise and fall in temperature are in agreement and no lag herin can be seen. There are however some small differences in the peaks of the temperatures. These differences are however small (maximum of 3 degrees). The overall quality is therefore satisfactory.

Code:

```
## Assignment 4.9

import sys
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.stats import mode
from scipy.spatial.distance import squareform
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.stattools import acf, pacf
from statsmodels.tsa.arima_model import ARIMA
import warnings
warnings.filterwarnings("ignore")  # ARIMA gave warnings

try:
    from IPython.display import clear_output
    have_ipython = True
except ImportError:
    have_ipython = False


class KnnDtw(object):
    """K-nearest neighbor classifier using dynamic time warping
    as the distance measure between pairs of time series arrays
```

```
26
27      Arguments
28      ---------
29      n_neighbors : int, optional (default = 5)
30          Number of neighbors to use by default for KNN
31
32      max_warping_window : int, optional (default = infinity)
33          Maximum warping window allowed by the DTW dynamic
34          programming function
35
36      subsample_step : int, optional (default = 1)
37          Step size for the timeseries array. By setting subsample_step = 2,
38          the timeseries length will be reduced by 50% because every second
39          item is skipped. Implemented by x[:, ::subsample_step]
40      """
41
42      def __init__(self, n_neighbors=5, max_warping_window=10000,
            subsample_step=1):
43          self.n_neighbors = n_neighbors
44          self.max_warping_window = max_warping_window
45          self.subsample_step = subsample_step
46
47      def fit(self, x, l):
48          """Fit the model using x as training data and l as class labels
49
50          Arguments
51          ---------
52          x : array of shape [n_samples, n_timepoints]
53              Training data set for input into KNN classifer
54
55          l : array of shape [n_samples]
56              Training labels for input into KNN classifier
57          """
58
59          self.x = x
60          self.l = l
61
62      def _dtw_distance(self, ts_a, ts_b, d=lambda x, y: abs(x - y)):
63          """Returns the DTW similarity distance between two 2-D
64          timeseries numpy arrays.
65
66          Arguments
67          ---------
68          ts_a, ts_b : array of shape [n_samples, n_timepoints]
69              Two arrays containing n_samples of timeseries data
70              whose DTW distance between each sample of A and B
71              will be compared
72
73          d : DistanceMetric object (default = abs(x-y))
74              the distance measure used for A_i - B_j in the
```

```
75              DTW dynamic programming function
76
77          Returns
78          -------
79          DTW distance between A and B
80          """
81
82          # Create cost matrix via broadcasting with large int
83          ts_a, ts_b = np.array(ts_a), np.array(ts_b)
84          M, N = len(ts_a), len(ts_b)
85          cost = 1e10 * np.ones((M, N))
86
87          # Initialize the first row and column
88          cost[0, 0] = d(ts_a[0], ts_b[0])
89          for i in range(1, M):
90              cost[i, 0] = cost[i - 1, 0] + d(ts_a[i], ts_b[0])
91
92          for j in range(1, N):
93              cost[0, j] = cost[0, j - 1] + d(ts_a[0], ts_b[j])
94
95          # Populate rest of cost matrix within window
96          for i in range(1, M):
97              for j in range(max(1, i - self.max_warping_window), min(N, i +
                    self.max_warping_window)):
98                  choices = cost[i - 1, j - 1], cost[i, j - 1], cost[i - 1,
                        j]
99                  cost[i, j] = min(choices) + d(ts_a[i], ts_b[j])
100                 # print("i=%d j=%d cost=%0.4g" % (i, j, cost[i, j]))
101
102         # print("Cost matrix:")
103         # print(self._print_cost_matrix(cost))
104
105         # Return DTW distance given window
106         return cost[-1, -1], cost
107
108     def _dist_matrix(self, x, y):
109         """Computes the M x N distance matrix between the training
110         dataset and testing dataset (y) using the DTW distance measure
111
112         Arguments
113         ---------
114         x : array of shape [n_samples, n_timepoints]
115
116         y : array of shape [n_samples, n_timepoints]
117
118         Returns
119         -------
120         Distance matrix between each item of x and y with
121             shape [training_n_samples, testing_n_samples]
122         """
```

```
123
124         # Compute the distance matrix
125         dm_count = 0
126
127         # Compute condensed distance matrix (upper triangle) of pairwise
                dtw distances
128         # when x and y are the same array
129         if (np.array_equal(x, y)):
130             x_s = np.shape(x)
131             dm = np.zeros((x_s[0] * (x_s[0] - 1)) // 2, dtype=np.double)
132
133             p = ProgressBar(np.shape(dm)[0])
134
135             for i in range(0, x_s[0] - 1):
136                 for j in range(i + 1, x_s[0]):
137                     dm[dm_count] = self._dtw_distance(x[i, ::self.
                            subsample_step],
138                                                       y[j, ::self.
                                                          subsample_step])
139
140                     dm_count += 1
141                     p.animate(dm_count)
142
143             # Convert to squareform
144             dm = squareform(dm)
145             return dm
146
147         # Compute full distance matrix of dtw distnces between x and y
148         else:
149             x_s = np.shape(x)
150             y_s = np.shape(y)
151             dm = np.zeros((x_s[0], y_s[0]))
152             dm_size = x_s[0] * y_s[0]
153
154             p = ProgressBar(dm_size)
155
156             for i in range(0, x_s[0]):
157                 for j in range(0, y_s[0]):
158                     dm[i, j], _ = self._dtw_distance(x[i, ::self.
                            subsample_step],
159                                                      y[j, ::self.
                                                         subsample_step])
160                     # Update progress bar
161                     dm_count += 1
162                     p.animate(dm_count)
163
164             return dm
165
166     def _print_cost_matrix(self, cost):
167         [i, j] = cost.shape
```

```
168             print("i=%d, j=%d" % (i, j))
169             cost[0][0] = 1
170             cost[0][1] = 1
171             for row in cost:
172                 r = "  "
173                 for c in row:
174                     r += ("%.4g" % c).rjust(11)
175                 print(r)
176 
177         def predict(self, x):
178             """Predict the class labels or probability estimates for
179             the provided data
180 
181             Arguments
182             ---------
183               x : array of shape [n_samples, n_timepoints]
184                   Array containing the testing data set to be classified
185 
186             Returns
187             -------
188               2 arrays representing:
189                   (1) the predicted class labels
190                   (2) the knn label count probability
191             """
192 
193             dm = self._dist_matrix(x, self.x)
194 
195             # Identify the k nearest neighbors
196             knn_idx = dm.argsort()[:, :self.n_neighbors]
197 
198             # Identify k nearest labels
199             knn_labels = self.l[knn_idx]
200 
201             # Model Label
202             mode_data = mode(knn_labels, axis=1)
203             mode_label = mode_data[0]
204             mode_proba = mode_data[1] / self.n_neighbors
205 
206             return mode_label.ravel(), mode_proba.ravel()
207 
208 
209 class ProgressBar:
210     """This progress bar was taken from PYMC
211     """
212 
213     def __init__(self, iterations):
214         self.iterations = iterations
215         self.prog_bar = '[]'
216         self.fill_char = '*'
217         self.width = 40
```

```
218         self.__update_amount(0)
219         if have_ipython:
220             self.animate = self.animate_ipython
221         else:
222             self.animate = self.animate_noipython
223 
224     def animate_ipython(self, iter):
225         print('\r', self, end="", flush=True)
226         sys.stdout.flush()
227         self.update_iteration(iter + 1)
228 
229     def update_iteration(self, elapsed_iter):
230         self.__update_amount((elapsed_iter / float(self.iterations)) *
               100.0)
231         self.prog_bar += '  %d of %s complete' % (elapsed_iter, self.
               iterations)
232 
233     def __update_amount(self, new_amount):
234         percent_done = int(round((new_amount / 100.0) * 100.0))
235         all_full = self.width - 2
236         num_hashes = int(round((percent_done / 100.0) * all_full))
237         self.prog_bar = '[' + self.fill_char * num_hashes + ' ' * (
               all_full - num_hashes) + ']'
238         pct_place = (len(self.prog_bar) // 2) - len(str(percent_done))
239         pct_string = '%d%%' % percent_done
240         self.prog_bar = self.prog_bar[0:pct_place] + \
241                         (pct_string + self.prog_bar[pct_place + len(
                               pct_string):])
242 
243     def __str__(self):
244         return str(self.prog_bar)
245 
246 
247 ### Exercise 4.9a
248 print("[Exercise 4.9]")
249 file_countries = "Earth Surface Temperature Study/
       GlobalLandTemperaturesByCountry.csv"
250 data_countries = pd.read_csv(file_countries)
251 
252 countries = ["Norway", "Finland", "Singapore", "Cambodia"]  # countries
       specified in the assignment
253 data_countries = data_countries[data_countries["Country"].isin(countries)]
        # remove data from other countries
254 
255 # Determine which years to plot to avoid NaN
256 index_nan = data_countries["Country"].isin(countries) & data_countries["
       AverageTemperature"].isna()
257 countries_grouped = data_countries[index_nan].groupby("Country")  # group
       data by country
258 
```

```
259 dates = []
260 for country, group in countries_grouped:
261     dates.append(group["dt"].values[-2])  # last date where nan occurs per
        country
262
263 dates.sort(reverse=True)  # sort newest date first
264 start_date = dates[0]  # minimum date from where to plot
265 print("[4.9a] No more NaN after", start_date)
266 npoints = 252  # amount of data points to plot
267
268 temperature_per_country = {}
269 timeline_per_country = {}
270 plt.figure(figsize=(12, 8))
271 for country, group in data_countries.groupby("Country"):
272     start_index = pd.Series(group["dt"] == start_date)  # find start index
273     start_index = start_index[start_index].index.values  # get value
274     plot_range = [int(start_index) - group.first_valid_index() + 1,
275                   int(start_index) - group.first_valid_index() + npoints]
276
277     temperatures = group["AverageTemperature"].to_numpy()[plot_range[0]:
        plot_range[1]]  # convert to numpy array
278     timeline = group["dt"].to_numpy()[plot_range[0]:plot_range[1]]  #
        convert dates to numpy array
279
280     # Plot
281     plt.plot(timeline, temperatures, label=country)
282
283     # Store (all) data in dict
284     temperature_per_country[country] = group["AverageTemperature"].
        to_numpy()[plot_range[0]:-2]
285     timeline_per_country[country] = group["dt"].to_numpy()[plot_range
        [0]:-2]
286
287 plt.xticks(rotation=90)
288 xticks = plt.gca().xaxis.get_major_ticks()
289 # Plot only every 12 labels
290 for i in range(len(xticks)):
291     if i % 12 != 0:
292         xticks[i].set_visible(False)
293 plt.tight_layout()
294 plt.ylabel("Temperature [deg]")
295 plt.legend()
296 plt.title("Yearly temperatures for %i years" % (npoints / 12 - 1))
297 plt.grid()
298 plt.tight_layout()
299 plt.savefig("figures/4.9a_temperatures.png", dpi=300)
300
301 print("[4.9a] Plot saved")
302 print()
303
```

```
304 dtw = KnnDtw()
305 print("[4.9a] Table with minimal DTW distance")
306 HeaderRow = " DISTANCE ".ljust(10)
307 for i1, c1 in enumerate(countries):
308     HeaderRow += c1.ljust(10)
309 print(HeaderRow)
310
311 for i1, c1 in enumerate(countries):
312     Row = (c1 + " ").rjust(10)
313     for i2, c2 in enumerate(countries):
314         distance, cost = dtw._dtw_distance(temperature_per_country[c1],
              temperature_per_country[c2])
315         Row += str(int(distance)).ljust(10)
316     print(Row)
317 print()
318
319 #  DISTANCE Norway    Finland   Singapore Cambodia
320 #    Norway 0         4046      47527     47779
321 #   Finland 4046      0         45403     45655
322 # Singapore 47527     45403     0         1283
323 #  Cambodia 47779     45655     1283      0
324
325 ### Exercise 4.9b
326 def test_stationarity(timeseries, country, timeline, ex):
327     # Determining rolling statistics
328     rolmean = pd.DataFrame(timeseries).rolling(window=12).mean()
329     rolstd = pd.DataFrame(timeseries).rolling(window=12).std()
330
331     # Plot rolling statistics:
332     orig = plt.plot(timeline, timeseries, color='blue', label='Original')
333     mean = plt.plot(rolmean, color='red', label='Rolling Mean')
334     std = plt.plot(rolstd, color='black', label='Rolling Std')
335     plt.legend(loc='best')
336     plt.ylabel("Temperature [deg]")
337     plt.grid()
338     plt.xticks(rotation=90)
339     xticks = plt.gca().xaxis.get_major_ticks()
340     # Plot only every 12 labels
341     for i in range(len(xticks)):
342         if i % 120 != 0:
343             xticks[i].set_visible(False)
344     plt.title('Rolling Mean & Standard Deviation %s' % country)
345     plt.tight_layout()
346     plt.savefig("figures/4.9%s_%s.png" % (ex, country), dpi=300)
347
348     # Perform Dickey-Fuller test:
349     print(country)
350     dftest = adfuller(timeseries, autolag='AIC')
351     dfoutput = pd.Series(dftest[0:4], index=['Test Statistic', 'p-value',
          '#Lags Used', 'Number of Observations Used'])
```

```
352     for key, value in dftest[4].items():
353         dfoutput['Critical Value (%s)' % key] = value
354     print(dfoutput)
355
356
357 print("[4.9b] Dickey-Fuller test results")
358 plt.figure(figsize=(6, 4))
359 for country in temperature_per_country:
360     plt.clf()
361     test_stationarity(temperature_per_country[country], country,
            timeline_per_country[country], 'b')
362     print()
363
364 ### 4.9c
365 temperature_decompose = {}
366 for country in temperature_per_country:
367     decomposition = seasonal_decompose(temperature_per_country[country],
            period=12)
368     trend = decomposition.trend
369     seasonal = decomposition.seasonal
370     residual = decomposition.resid
371     decomposed = residual
372     temperature_decompose[country] = decomposed[~np.isnan(decomposed)]
373
374 plt.clf()
375 for country in temperature_decompose:
376     plt.plot(temperature_decompose[country][:120], label=country)
377 plt.grid()
378 plt.legend()
379 plt.xlabel("Months")
380 plt.ylabel("Temperature [deg]")
381 plt.title("Decomposed temperatures")
382 plt.tight_layout()
383 plt.savefig("figures/4.9c.png", dpi=300)
384
385 dtw = KnnDtw()
386 print("[4.9c] Table with minimal DTW distance")
387 HeaderRow = " DISTANCE ".ljust(10)
388 for i1, c1 in enumerate(countries):
389     HeaderRow += c1.ljust(10)
390 print(HeaderRow)
391
392 for i1, c1 in enumerate(countries):
393     Row = (c1 + " ").rjust(10)
394     for i2, c2 in enumerate(countries):
395         distance, cost = dtw._dtw_distance(temperature_decompose[c1],
                temperature_decompose[c2])
396         Row += str(int(distance)).ljust(10)
397     print(Row)
398 print()
```

```
399
400 #  DISTANCE Norway    Finland   Singapore Cambodia
401 #    Norway 0         1437      1650      1454
402 #   Finland 1437      0         2565      2335
403 # Singapore 1650      2565      0         421
404 #  Cambodia 1454      2335      421       0
405
406 ### 4.9d
407 country = countries[0]
408 print("[4.9d] Forecasting for:", country)
409 # Remove five years for testing at the end
410 temperature_data_train = temperature_per_country[country][:-12 * 5]
411 temperature_data_test = temperature_per_country[country][-12 * 5:]
412 timeline_test = timeline_per_country[country][-12 * 5:]
413 temperature_diff = np.diff(temperature_per_country[country])  #
    differencing
414 temperature_diff = temperature_diff[~np.isnan(temperature_diff)]  # remove
     nan
415
416 lag_acf = acf(temperature_diff, nlags=20)
417 lag_pacf = pacf(temperature_diff, nlags=20, method='ols')
418
419 # Plot ACF:
420 plt.figure(figsize=(15, 6))
421 plt.subplot(121)
422 plt.plot(lag_acf)
423 plt.axhline(y=0, linestyle='--', color='gray')
424 plt.axhline(y=-1.96 / np.sqrt(len(temperature_diff)), linestyle='--',
    color='gray')
425 plt.axhline(y=1.96 / np.sqrt(len(temperature_diff)), linestyle='--', color
    ='gray')
426 plt.grid()
427 plt.title('Autocorrelation Function')
428
429 # Plot PACF:
430 plt.subplot(122)
431 plt.plot(lag_pacf)
432 plt.axhline(y=0, linestyle='--', color='gray')
433 plt.axhline(y=-1.96 / np.sqrt(len(temperature_diff)), linestyle='--',
    color='gray')
434 plt.axhline(y=1.96 / np.sqrt(len(temperature_diff)), linestyle='--', color
    ='gray')
435 plt.grid()
436 plt.title('Partial Autocorrelation Function')
437 plt.tight_layout()
438 plt.savefig("figures/4.9d_(p)acf.png", dpi=300)
439
440 upper_confidence = 1.96 / np.sqrt(len(temperature_decompose[country]))
441 p = np.argmax(lag_pacf < upper_confidence)
442 q = np.argmax(lag_acf < upper_confidence)
```

```
443 d = 0
444
445 print("[4.9d] p = %i, q = %i" % (p, q))
446 npoints = 120  # amount of data points to plot
447
448 ## AR model
449 plt.figure()
450 model = ARIMA(temperature_per_country[country], order=(p, d, 0))
451 results_AR = model.fit(disp=-1)
452 plt.plot(temperature_diff[0:npoints], label='data')
453 plt.plot(results_AR.fittedvalues[0:npoints], color='red', label='model')
454 plt.grid()
455 plt.xlabel("Months")
456 plt.ylabel("Temperature [deg]")
457 plt.legend()
458 plt.title('RSS: %.4f' % sum((results_AR.fittedvalues[:-1] -
       temperature_diff) ** 2))
459 plt.savefig("figures/4.9d_ar.png", dpi=300)
460
461 ## MA model
462 plt.clf()
463 model = ARIMA(temperature_per_country[country], order=(0, d, q))
464 results_MA = model.fit(disp=-1)
465 plt.plot(temperature_diff[0:npoints], label='data')
466 plt.plot(results_MA.fittedvalues[0:npoints], color='red', label='model')
467 plt.grid()
468 plt.xlabel("Months")
469 plt.ylabel("Temperature [deg]")
470 plt.legend()
471 plt.title('RSS: %.4f' % sum((results_MA.fittedvalues[:-1] -
       temperature_diff) ** 2))
472 plt.savefig("figures/4.9d_ma.png", dpi=300)
473
474 ## Combined model
475 plt.clf()
476 model = ARIMA(temperature_per_country[country], order=(p, d, q))
477 results_ARIMA = model.fit(disp=-1)
478 plt.plot(temperature_diff[0:npoints], label='data')
479 plt.plot(results_ARIMA.fittedvalues[0:npoints], color='red', label='model'
       )
480 plt.grid()
481 plt.xlabel("Months")
482 plt.ylabel("Temperature [deg]")
483 plt.legend()
484 plt.title('RSS: %.4f' % sum((results_ARIMA.fittedvalues[:-1] -
       temperature_diff) ** 2))
485 plt.savefig("figures/4.9d_arma.png", dpi=300)
486
487 exit()
488
```

```
489 ### 4.9e
490 predictions = results_ARIMA.predict(start=len(temperature_data_train),
491                                      end=len(
                                             temperature_data_train
                                             ) + len(
                                             temperature_data_test
                                             ) - 1,
492                                      typ='levels')
493 plt.clf()
494 plt.plot(timeline_test, temperature_data_test, label='original')
495 plt.plot(predictions, label='prediction')
496 plt.grid()
497 plt.legend()
498 plt.ylabel("Temperature [deg]")
499 xticks = plt.gca().xaxis.get_major_ticks()
500 # Plot only every 12 labels
501 for i in range(len(xticks)):
502     if i % 12 != 0:
503         xticks[i].set_visible(False)
504 plt.title('5 year prediction for %s' % country)
505 plt.savefig("figures/4.9e.png", dpi=300)
```