Learning LogBook Tree phenology analysis with R

Philipp Münker

2023-01-05

Contents

1 Introduction **5**

2 Tree phenology **7**

3 Tree dormancy **9**
3.1 Task 1 9
3.2 Task 2 9
3.3 Task 3 10

4 Climate change and impact projection **11**
4.1 Task 1 11
4.2 Task 2 12

5 Manual Chill Analysis **13**
5.1 Task 1 14
5.2 Task 2 14
5.3 Task 3 15

6 Chill Models **17**
6.1 Task 1 17
6.2 Task2 18
6.3 Task3 19

7 Making hourly temperatures **21**
7.1 Task 1 21
7.2 Task 2 24

8 Getting temperature data **33**
8.1 Task 1 33
8.2 Task 2 34
8.3 Task 3 35

9 Filling gaps in temperature records **37**
9.1 Task 1 37
9.2 Task 2 37

9.3 Task 3 39

Chapter 1

Introduction

In this learning Logbook, all units from the Tree phenology analysis with R module are documented. In addition to the tasks set at the end of each learning unit, this work is supplemented with additional materials and analyses. This is done using weather data taken from my own weather station. The corresponding data can be found at the following link: https://wettermuehle.de. Access data can be requested if desired.

Chapter 2

Tree phenology

If we consider fruit trees, their annual cycle can generally be described relatively easily. Starting in autumn, it is observed that almost all fruit trees shed their leaves and go into winter without foliage. Already in the autumn, the formation of the bud can often be observed. This bud then remains in a kind of winter dormancy throughout the winter and begins to grow with increasing temperatures in the spring. This process is usually followed by the flowering of the fruit trees with subsequent leaf development. Later in the year, fruits establish themselves from the buds, which mature at different times of the year. But how does the tree know when it can begin flowering induction and no longer expect strong frost? This can be described with the concept of dormancy. This can be divided into 4 phases.

Tree dormancy

- Dormancy establishment
- Endodormancy
- Ecodormancy
- Growth resumption

Dormacy establishment

- Controlled by tempeature and photoperiod.

Endodormancy

- Controlled by plant endogenous factors. Plants unable to growth even under favorable environmental conditions.

Ecodormancy

- After a certain level of chill, endodormancy has been overcome and buds recover the capacity to grow. Trees become acclimated to freezing tolerance and are not deeply dormant, but growth is still prevented by unsuitable environmental conditions. Temperature is the most important driver in this process.

Chapter 3

Tree dormancy

3.1 Task 1

Put yourself in the place of a breeder who wants to calculate the temperature requirements of a newly released cultivar. Which method will you use to calculate the chilling and forcing periods? Please justify your answer.

> Long-term phenological data does not exist, so a statistical approach is not optimal for newly released cultivars. It is better to work with an empirical approach. To do this, collect flower buds and place shoots in a chamber for 10 days under favorable conditions (temperature between 20 and 25 degrees). After 10 days, measure the weight of the shoots in the chamber and the shoots without the chamber. If the weight difference is greater than 30%, the cultivar is considered non-dormant. Otherwise, it is considered dormant

3.2 Task 2

Which are the advantages (2) of the BBCH scale compared with earlies scales?

> Not all parts of a tree are in the same development stage. Early scales only record the predominant state of the fruit tree. General principles for the design of a scale for plant growth stages include:
>
> - Growth stages are easily recognizable under field conditions
> - Growth stages are graded in the order of appearance (early scales do not do this)

- Two-digit code: Principal growth stages | Secondary growth stages
- Applicable for all cereals in all parts of the world. (Old scales can only be used for a specific group/fruit)

3.3 Task 3

Classify the following phenological stages of sweet cherry according to the BBCH scale:

Figure 3.1: Picture 1 BBCH =55; Picture 2 BBCH =67; Picture 3 BBCH =89

Chapter 4

Climate change and impact projection

4.1 Task 1

List the main drivers of climate change at the decade to century scale, and briefly explain the mechanism through which the currently most important driver affects our climate

> The most important factor that currently has the greatest influence on climate and climate change is greenhouse gases. The most important greenhouse gases are water vapor, (carbon dioxide) CO_2, (methane) CH_4, and (nitrous oxides) N_2O. Greenhouse gases can only absorb radiation of certain wavelengths. They absorb radiation with long wavelengths, which comes from the Earth's surface in the form of infrared radiation emitted by the warm Earth's surface. This radiation cannot leave the atmosphere and is trapped by the greenhouse gases, which returns it back to the Earth.

Table 4.1: Drivers of Climate Change

Drivers of climate change
Sun
Aerosols
Clouds
Ozone
Surface albedo
Greenhouse gases

4.2 Task 2

Explain briefly what is special about temperature dynamics of the recent decades, and why we have good reasons to be concerned

Over the past decades and throughout the last century, the temperature has been rising worldwide. Initially, this increase was relatively slow. The ten warmest years worldwide since 1880 were all measured after the millennium. The five warmest years worldwide were all recorded after 2014. This effect is also noticeable in Germany. Here, too, the ten warmest years were all measured after 2000, with one exception. If one places the temperature increase of the last decades in the climate history of the last one million years, it can be seen that there has never been such a strong temperature increase over such a relatively short period of time.

This rapid rise in temperature is developing its own dynamic. For example, high temperatures in the tundra cause the permafrost to thaw, releasing a large amount of CO2, a greenhouse gas that promotes even faster warming.

Chapter 5

Manual Chill Analysis

The `Winters_hours_gaps` data set has the columns: `Year`, `Month`, `Day`, `Hour`, `Temp_gaps`, `Temp`. First, the function `cleaned_data` is used to remove unnecessary columns such as `Temp_gaps()` from the data set.

```
#Clean Function
cleaned_data = function(data_source) {
  data_source =
    data_source[, c("Year", "Month", "Day", "Hour", "Temp")]
  return(data_source)
}

# Apply Function to Winters_hours_gaps
kable(head(cleaned_data(data_source = Winters_hours_gaps)),
      caption = "Cleaned  Dataset: Winters_hours_gaps")%>%
      kable_styling("striped", position = "left", font_size = 10)%>%
      scroll_box(width = "100%")
```

\begin{table}

\caption{Cleaned Dataset: Winters_hours_gaps}

Year	Month	Day	Hour	Temp
2008	3	3	10	15.127
2008	3	3	11	17.153
2008	3	3	12	18.699
2008	3	3	13	18.699
2008	3	3	14	18.842
2008	3	3	15	19.508

\end{table}

5.1 Task 1

Write a basic function that calculates warm hours (>25°C)

```
WH = function(hourtemps)
{
  hourtemps[, "warm_hours"] <- hourtemps$Temp >= 25.0
  return(hourtemps)
}
```

5.2 Task 2

Apply this function to the Winters_hours_gaps dataset

```
# have a look to the data set
kable(head(Winters_hours_gaps),caption =
        "Example Dataset: Winters_hours_gaps")%>%
        kable_styling("striped", position = "left", font_size = 10)%>%
        scroll_box(width = "100%")
```

\begin{table}

\caption{Example Dataset: Winters_hours_gaps}

Year	Month	Day	Hour	Temp_gaps	Temp
2008	3	3	10	15.127	15.127
2008	3	3	11	17.153	17.153
2008	3	3	12	18.699	18.699
2008	3	3	13	18.699	18.699
2008	3	3	14	18.842	18.842
2008	3	3	15	19.508	19.508

\end{table}

```
# Apply Function
hourtemps = cleaned_data(data_source = Winters_hours_gaps)
kable(head(WH(hourtemps = hourtemps)))%>%
        kable_styling("striped", position = "left", font_size = 10)%>%
        scroll_box(width = "100%")
```

Year	Month	Day	Hour	Temp	warm_hours
2008	3	3	10	15.127	FALSE
2008	3	3	11	17.153	FALSE
2008	3	3	12	18.699	FALSE
2008	3	3	13	18.699	FALSE
2008	3	3	14	18.842	FALSE
2008	3	3	15	19.508	FALSE

5.3 Task 3

Extend this function, so that it can take start and end dates as inputs and sums up warm hours between these dates

```r
warm_hours_function = function(Input_Data,
                               S_Jahr,
                               S_Monat,
                               S_Tag,
                               S_Stunde,
                               E_Jahr,
                               E_Monat,
                               E_Tag,
                               E_Stunde) {
  Start_Date <-
    which(
      hourtemps$Year == S_Jahr & hourtemps$Month == S_Monat &
        hourtemps$Day == S_Tag &
        hourtemps$Hour == S_Stunde
    )
  End_Date <- which(
    hourtemps$Year == E_Jahr & hourtemps$Month == E_Monat &
      hourtemps$Day == E_Tag & hourtemps$Hour == E_Stunde
  )

  # Apply Function Warm Hours (WH)
  hourtemps = WH(hourtemps = Input_Data)

  # Calculate warm_hours
  warm_hours = sum(hourtemps$warm_hours[Start_Date:End_Date])

  return(cat("The number of heat hours is:", paste(warm_hours)))
}

warm_hours_function(
  Input_Data = hourtemps,
  S_Jahr = 2008,
  S_Monat = 5,
  S_Tag = 1,
  S_Stunde = 12,
  E_Jahr = 2008,
  E_Monat = 8,
  E_Tag = 31,
  E_Stunde = 12
)
```

```
## The number of heat hours is: 957
```

Chapter 6

Chill Models

Counting chill hours can be done in various ways. `ChillR` offers some functions for this purpose. The simplest function for this is the `Chilling_Hours()` function. It records one chill hour for every temperature between 0 and 7.2 degrees. A slightly more complex function is the `Utah_Model()` function. It evaluates the measured temperatures and decides whether a full chill hour was reached or only half. For example, if the temperature is between 1 and 2 degrees, one chill hour has been reached. If it is between 3 and 4 degrees, two chill hours are recorded. The `Dynamic_model()` function is the most complex function. It is taken from an Excel sheet. The `chilling()` function combines the functions described above and presents the results in an overview.

6.1 Task 1

Run the chilling() function on the Winters_hours_gap dataset

```
# run chilling function on Winters_hours_gap dataset
output =
  chilling(make_JDay(Winters_hours_gaps),
           Start_JDay = 90,
           End_JDay = 100)

kable(output, caption ="chilling function on Winters_hours_gap") %>%
        kable_styling("striped", position = "left", font_size = 10)%>%
        scroll_box(width = "100%")
```

\begin{table}

\caption{chilling function on Winters_hours_gap}

Season	End_year	Season_days	Data_days	Perc_complete	Chilling_Hours	Utah_Mo
2007/2008	2008	11	11	100	40	15

\end{table}

6.2 Task2

Create your own temperature-weighting chill model using the `step_model()` function

> The `step_model` function has two arguments that the user can pass. One is a dataset of temperature data `HourTemp` and the other is a `data.frame()` (df) consisting of `lower`, `upper`, and `weight`. A pre-defined lower temperature range from, for example, -1000 °C to 0 °C is set, with all temperatures within this range being assigned a weight of 0. Assuming that hourly temperature data is provided as input, the temperature can be multiplied by the corresponding weight to obtain the amount of "chillhours". For example: -1 °C is within the range [-1000, 0] == 0, resulting in 0 chillhours. Another argument is `summ`. If `summ = TRUE`, the cumulative chillhours over a defined period will be output. If `summ = FALSE`, the weights of the chillhours will be output.

```
step_model = function (HourTemp,
                        df =
                          data.frame(
                            lower = c(-1000, 1.4, 2.4, 9.1, 12.4, 15.9, 18),
                            upper = c(1.4, 2.4, 9.1, 12.4, 15.9, 18, 1000),
                            weight = c(0, 0.5, 1, 0.5, 0, -0.5, -1)
                          ),
                        summ = TRUE)
{
  lower <- df$lower
  upper <- df$upper
  weight <- df$weight
  if (summ == TRUE)
    return(cumsum(sapply(HourTemp, function(x)
      weight[which(x >
                     lower & x <= upper)])))
  else
    return(sapply(HourTemp, function(x)
      weight[which(x >
                     lower & x <= upper)]))
}
```

> Here, only an "own data field" is defined with its own limits that

have their own weight. For example, from -100 °C to 0 °C, the weight is set to 0. In this case, no "chillhour" occurs. If the temperature is between 0 °C and 2 °C, the weight of the "chillhour" is 0.5. In this case, half a "chillhour" occurs.

```
own_df = data.frame (lower  = c(-100,0,  2, 4,  5, 6,   7    ),
                     upper  = c(  0, 2,  4, 5,  6, 7,   100  ),
                     weight = c(  0, 0.5,1, 1.5,1, 0.5, 0    ))
```

After the dataframe with your own weights has been created, it can be implemented into the `step_model()` function.

```
use_step_model = function(x){step_model(x,own_df)}

# quick aplly
use_step_model(x = Winters_hours_gaps$Temp)[1:100]
```

```
##   [1]  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.5
##  [16]  1.0  1.0  1.0  1.5  2.5  3.5  4.5  4.5  4.5  4.5  4.5  4.5  4.5  4.5  4.5
##  [31]  4.5  4.5  4.5  4.5  4.5  4.5  4.5  4.5  4.5  4.5  4.5  4.5  4.5  4.5  4.5
##  [46]  4.5  4.5  4.5  4.5  4.5  4.5  4.5  4.5  4.5  4.5  4.5  4.5  4.5  4.5  4.5
##  [61]  4.5  5.0  5.5  6.0  7.0  8.5  9.5 10.5 11.5 12.5 13.0 13.0 13.0 13.0 13.0
##  [76] 13.0 13.0 13.0 13.0 13.0 13.0 13.0 13.0 13.0 13.0 13.5 14.0 15.0 16.5 18.0
##  [91] 19.5 21.0 22.0 23.0 23.0 23.0 23.0 23.0 23.0 23.0
```

6.3 Task3

Run this model on the Winters_hours_gaps dataset using the tempResponse() function

The `tempResponse()` function can display and summarize some chill models. Here is the model `weather_mill()` our own chilling model which is created by the `step_model()`. The modified `step_model()` function is renamed to `use_step_model()` and passed as a parameter to the tempResponse function (weather_mill = use_step_model).

```
output <-
  tempResponse(
    make_JDay(Winters_hours_gaps),
    Start_JDay = 30,
    End_JDay = 100,
    models = list(
      Chill_Portions = Dynamic_Model,
      GDH = GDH,
      weather_mill = use_step_model, # own model weather_mill
      Utah_Model = Utah_Model
```

Table 6.1: Summarized some models

Season	End_year	Season_days	Data_days	Perc_complete	Chill_Portions	GDH
2007/2008	2008	71	37.58333	52.93427	5.930439	8392.585

```
    )
  )

# display result
kable(output, caption = "Summarized some models") %>%
        kable_styling("striped", position = "left", font_size = 10)%>%
        scroll_box(width = "100%")
```

If the Utah Model is included, which is based on the default settings of the Step Model, a clear difference between the modified Step Model and the Utah Model can be observed.

Chapter 7

Making hourly temperatures

7.1 Task 1

Choose a location of interest, find out its latitude and produce plots of daily sunrise, sunset and daylength

> First, I would like to compare the lengths of days among locations of interest. I have selected Glogau in Poland, Zülpich in Germany, Tenerife in Spain, Moscow in Russia, and Karkaralinsk in Kazakhstan.

```
# initialize the variables with daylength, sunrise and sunset by the function daylength
Glogau      <- daylength(latitude = 51.40, JDay = 1:365)
Teneriffa   <- daylength(latitude = 28.19, JDay = 1:365)
Zuelpich    <- daylength(latitude = 50.42, JDay = 1:365)
Moskau      <- daylength(latitude = 55.45, JDay = 1:365)
Karkaralinsk <- daylength(latitude = 49.24, JDay = 1:365)

# Create a dataframe consisting of the variables "base" (days 1 to 365) and the
# respective locations and containing only the day length for each location.
df <- data.frame(
  base         = seq(length(Glogau[[1]])),
  Glogau       = Glogau[[3]],
  Teneriffa    = Teneriffa[[3]],
  Zülpich      = Zuelpich[[3]],
  Moskau       = Moskau[[3]],
  Karkaralinsk = Karkaralinsk[[3]]
)
```

Table 7.1: Differnt Locations

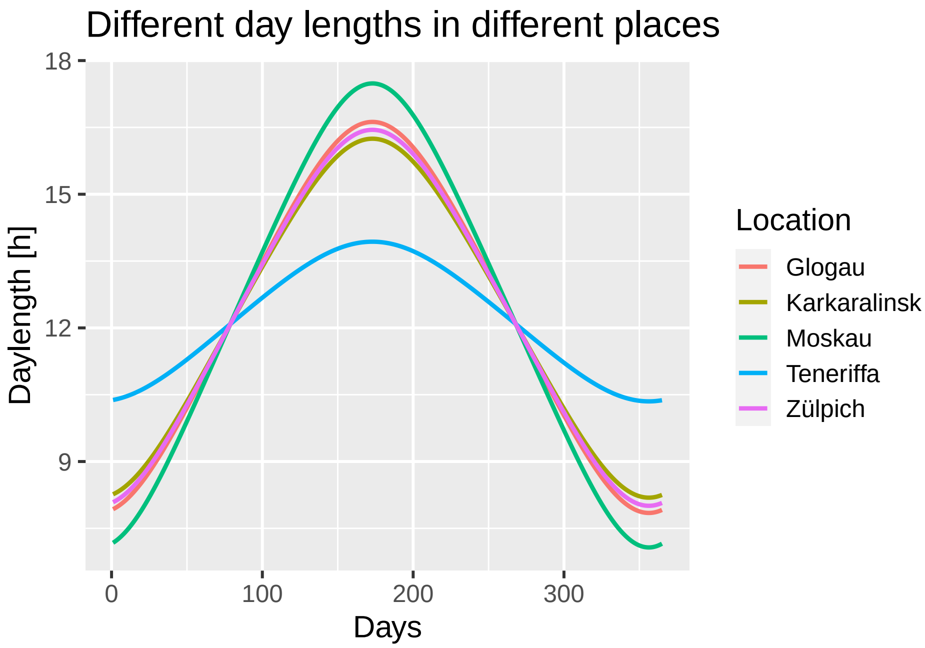
base	Glogau	Teneriffa	Zülpich	Moskau	Karkaralinsk
1	7.930050	10.38197	8.086864	7.175796	8.265160
2	7.948054	10.38872	8.104044	7.198073	8.281428
3	7.967737	10.39612	8.122830	7.222407	8.299217
4	7.989077	10.40415	8.143199	7.248768	8.318509
5	8.012048	10.41281	8.165129	7.277120	8.339282
6	8.036625	10.42209	8.188595	7.307424	8.361515

```
kable(head(df), caption = "Differnt Locations") %>%
       kable_styling("striped", position = "left", font_size = 10)%>%
        scroll_box(width = "100%")
```

```
# create a pivot table
df_long <-
  pivot_longer(df, -"base", names_to = "Location", values_to = "daylength")

kable(head(df_long)) %>%
      kable_styling("striped", position = "left", font_size = 10)%>%
      scroll_box(width = "100%")
```

base	Location	daylength
1	Glogau	7.930050
1	Teneriffa	10.381965
1	Zülpich	8.086864
1	Moskau	7.175796
1	Karkaralinsk	8.265160
2	Glogau	7.948054

```
# plot the result with ggplot
ggplot(df_long, aes(x = base, y = daylength, groupe = Location)) +
  geom_line(aes(color = Location), lwd = 1.0) +
  ggtitle("Different day lengths in different places") +
  labs(x = "Days", y = "Daylength [h]") + theme_gray(base_size = 15)
```



Create a summary of the sunrise, sunset, and day length for Moscow.

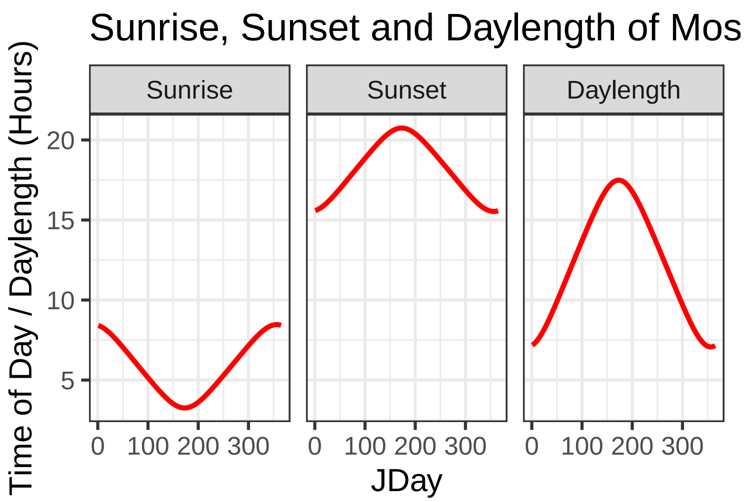
```
Days <- daylength(latitude = 55.45, JDay = 1:365)

Days_df <-
  data.frame(
    JDay = 1:365,
    Sunrise = Days$Sunrise,
    Sunset = Days$Sunset,
    Daylength = Days$Daylength
  )

Days_df<-melt(Days_df, id=c("JDay"))
```

Show the final result

```
ggplot(Days_df, aes(x = JDay, y = value)) + geom_line(lwd = 1.5, color = "red") + facet_grid(cols
  ylab("Time of Day / Daylength (Hours)") + theme_bw(base_size = 20) +
  ggtitle("Sunrise, Sunset and Daylength of Moskau")
```

Table 7.2: Weather Station Fuessenich

Location	State	GPS	Gauß_K
Zuelpich - Fuessenich	North Rhine-Westphalia	50.69527026369208, 6.615666577711913	Rechtswe



7.2 Task 2

Produce an hourly dataset, based on idealized daily curves, for the KA_weather dataset (included in chillR)

> The following two tasks were performed in a modified form. In order to demonstrate the application of the chillR package, it was decided to use a currently active weather station and use its data as a basis. Data on the weather station can be found in the table below.

First, corresponding data must be read in. The data are already prepared.

```
Zuelpich_hourly = read.table(
  "weather_data/Weather_Zuelpich_2019_hourly.csv",
  header = TRUE,
  sep = ","
)

Zuelpich_min_max = read.table("weather_data/Weather_Zuelpich_2019.csv",
                              header = TRUE,
                              sep = ",")
```

```
zuelpich_april = Zuelpich_hourly %>% filter("2019-04-01 00:00:00" < date) %>%
  filter("2019-04-05 00:00:00" > date)
```

Table 7.3: Dataset:Tmean Tmax Tmin

Tag	Mittel	Tmax	Tmin
1	7.7	17.810000	-0.420000
2	9.5	17.096667	2.693333
3	7.6	9.983333	5.481667
4	5.7	7.763333	3.566667

```r
zuelpich_april$date_new <- as.POSIXct(zuelpich_april[, 3])
zuelpich_april$date_newnew = as.Date(zuelpich_april[, 3])

kable(head(zuelpich_april), caption = "Dataset: zuelpich_april") %>%
        kable_styling("striped", position = "left", font_size = 10)%>%
       scroll_box(width = "100%")
```

\begin{table}

\caption{Dataset: zuelpich_april}

X	temperature	date	date_new	date_newnew
1412	3.4350000	2019-04-01 01:00:00	2019-04-01 01:00:00	2019-04-01
1413	2.0900000	2019-04-01 02:00:00	2019-04-01 02:00:00	2019-04-01
1414	1.2616667	2019-04-01 03:00:00	2019-04-01 03:00:00	2019-04-01
1415	0.6150000	2019-04-01 04:00:00	2019-04-01 04:00:00	2019-04-01
1416	0.0583333	2019-04-01 05:00:00	2019-04-01 05:00:00	2019-04-01
1417	-0.3566667	2019-04-01 06:00:00	2019-04-01 06:00:00	2019-04-01

\end{table}

Next, the lows and highs for the corresponding days must be determined from the data set containing hourly data.

```r
final <- zuelpich_april %>%
  group_by(Tag = day(date_newnew)) %>%
  summarise(
    Mittel =  round(mean(temperature, na.rm = TRUE), digits = 1),
    Tmax = max(temperature),
    Tmin = min(temperature)
  )

kable(final, caption = "Dataset:Tmean Tmax Tmin")%>%
        kable_styling("striped", position = "left", font_size = 10)%>%
        scroll_box(width = "100%")
```

Next, the dataset containing hourly temperature values must be extended with a column that will later represent the daily high and low values. First, the new column Tmax_Tmin is filled with NAs. Then the maximum and minimum values are taken from the previously generated dataset final. These values are compared with the hourly values. If they match, the maximum or minimum value found is written to the previously created column Tmax_Tmin. In this way, the daily maximum and minimum values are placed in the table zuelpich_april at the same place where they were also measured.

```
# generate new coloumn with NAs
zuelpich_april$Tmax_Tmin = NA

# match Tmin
for (i in seq(1, nrow(zuelpich_april))) {
  for (j in seq(1, nrow(final))) {
    if (zuelpich_april$temperature[i] == final$Tmax[j]) {
      zuelpich_april$Tmax_Tmin[i] <- final$Tmax[j]
    }
  }
}

# match Tmax
for(i in seq(1, nrow(zuelpich_april))) {
  for (j in seq(1, nrow(final))) {
    if (zuelpich_april$temperature[i] == final$Tmin[j]) {
      zuelpich_april$Tmax_Tmin[i] <- final$Tmin[j]
    }
  }
}


kable(zuelpich_april[1:25,], caption = "Dataset: TmaxTminMatch")%>%
        kable_styling("striped", position = "left", font_size = 10)%>%
        scroll_box(width = "100%")
```

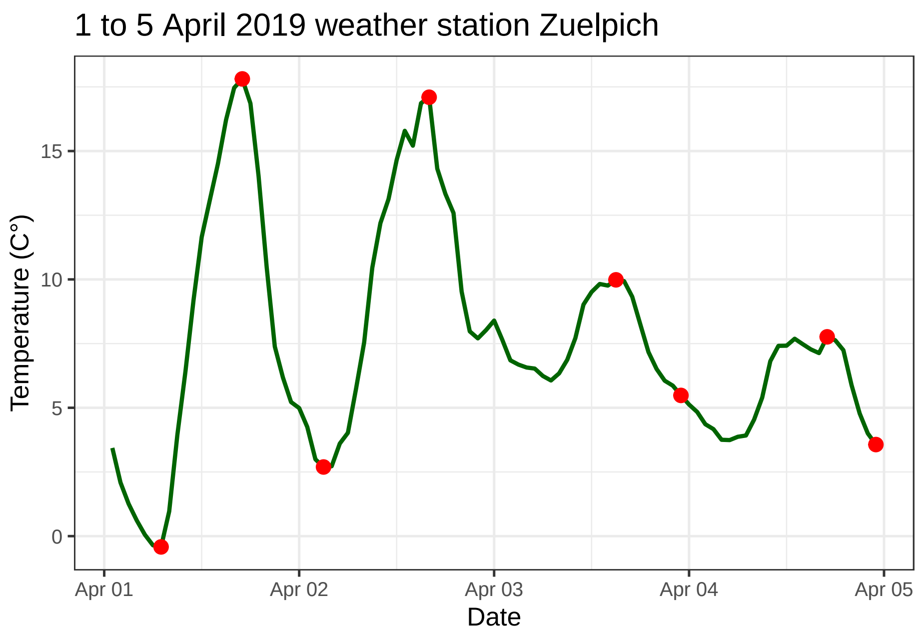
After creating a new column containing only the Tmax and Tmin temperatures, a plot can be created that shows the measured temperature history and includes information on Tmax and Tmin. The red dots symbolize the daily temperature values for Tmax and Tmin, respectively.

```
ggplot(data = zuelpich_april, aes(x = zuelpich_april[, 4], y = zuelpich_april[, 2])) +
  geom_line(size = 1.0, colour = "darkgreen") +
  geom_point(aes(y = zuelpich_april$Tmax_Tmin),
             colour = "red",
             size = 3.0) +
  labs(x = "Date", y = "Temperature (C°)") +
```

Table 7.4: Dataset: TmaxTminMatch

X	temperature	date	date_new	date_newnew	Tmax_Tmin
1412	3.4350000	2019-04-01 01:00:00	2019-04-01 01:00:00	2019-04-01	NA
1413	2.0900000	2019-04-01 02:00:00	2019-04-01 02:00:00	2019-04-01	NA
1414	1.2616667	2019-04-01 03:00:00	2019-04-01 03:00:00	2019-04-01	NA
1415	0.6150000	2019-04-01 04:00:00	2019-04-01 04:00:00	2019-04-01	NA
1416	0.0583333	2019-04-01 05:00:00	2019-04-01 05:00:00	2019-04-01	NA
1417	-0.3566667	2019-04-01 06:00:00	2019-04-01 06:00:00	2019-04-01	NA
1418	-0.4200000	2019-04-01 07:00:00	2019-04-01 07:00:00	2019-04-01	-0.42
1419	0.9666667	2019-04-01 08:00:00	2019-04-01 08:00:00	2019-04-01	NA
1420	3.9333334	2019-04-01 09:00:00	2019-04-01 09:00:00	2019-04-01	NA
1421	6.4183333	2019-04-01 10:00:00	2019-04-01 10:00:00	2019-04-01	NA
1422	9.2050000	2019-04-01 11:00:00	2019-04-01 11:00:00	2019-04-01	NA
1423	11.6483333	2019-04-01 12:00:00	2019-04-01 12:00:00	2019-04-01	NA
1424	13.0883333	2019-04-01 13:00:00	2019-04-01 13:00:00	2019-04-01	NA
1425	14.5100000	2019-04-01 14:00:00	2019-04-01 14:00:00	2019-04-01	NA
1426	16.2199999	2019-04-01 15:00:00	2019-04-01 15:00:00	2019-04-01	NA
1427	17.4650000	2019-04-01 16:00:00	2019-04-01 16:00:00	2019-04-01	NA
1428	17.8100000	2019-04-01 17:00:00	2019-04-01 17:00:00	2019-04-01	17.81
1429	16.8549999	2019-04-01 18:00:00	2019-04-01 18:00:00	2019-04-01	NA
1430	14.0283333	2019-04-01 19:00:00	2019-04-01 19:00:00	2019-04-01	NA
1431	10.4783333	2019-04-01 20:00:00	2019-04-01 20:00:00	2019-04-01	NA
1432	7.3850000	2019-04-01 21:00:00	2019-04-01 21:00:00	2019-04-01	NA
1433	6.1783334	2019-04-01 22:00:00	2019-04-01 22:00:00	2019-04-01	NA
1434	5.2233333	2019-04-01 23:00:00	2019-04-01 23:00:00	2019-04-01	NA
1435	4.9883333	2019-04-02 00:00:00	2019-04-02 00:00:00	2019-04-02	NA
1436	4.2500000	2019-04-02 01:00:00	2019-04-02 01:00:00	2019-04-02	NA

```
  ggtitle("1 to 5 April 2019 weather station Zuelpich") +
  theme_bw(base_size = 13)
```



First, the dataset `ZU_weather` must be created. The columns `DATE`, `Year`, `Month`, `Day`, `Tcontinue`, and `Temp_inter` are created. The `Temp_inter` column contains temperature data with large gaps that must be interpolated between.

```
ZU_weather = data.frame(
  DATE = zuelpich_april[, 4],
  Year = as.numeric(substr(zuelpich_april[, 4], 1, 4)),
  Month = as.numeric(substr(zuelpich_april[, 4], 6, 7)),
  Day = as.numeric(substr(zuelpich_april[, 4], 9, 10)),
  Tcontinue = zuelpich_april[, 2],
  Temp_inter = zuelpich_april[, 6]
)

kable(ZU_weather[1:8,])%>%
        kable_styling("striped", position = "left", font_size = 10)%>%
      scroll_box(width = "100%")
```

DATE	Year	Month	Day	Tcontinue	Temp_inter
2019-04-01 01:00:00	2019	4	1	3.4350000	NA
2019-04-01 02:00:00	2019	4	1	2.0900000	NA
2019-04-01 03:00:00	2019	4	1	1.2616667	NA
2019-04-01 04:00:00	2019	4	1	0.6150000	NA
2019-04-01 05:00:00	2019	4	1	0.0583333	NA
2019-04-01 06:00:00	2019	4	1	-0.3566667	NA
2019-04-01 07:00:00	2019	4	1	-0.4200000	-0.42
2019-04-01 08:00:00	2019	4	1	0.9666667	NA

The next step is to use the `interpolate_gaps()` function to calculate the missing temperatures between `Tmax` and `Tmin`. The function `interpolate_gaps()` returns a list with two entries. The first entry of the list contains the interpolated values, which can be accessed using `$interp` or `[[1]]`. The second entry, `$missing`, gives information on whether a value needs to be interpolated or if a real value is present. The function `interpolate_gaps()` linearly interpolates between gaps in the temperature records. The interpolated values are written directly to the Temp_inter column using the first list entry created by the `interpolate_gaps()` function.

```
# interpolate between gaps in coloum Temp_inter
ZU_weather$Temp_inter <- interpolate_gaps(ZU_weather$Temp_inter)[[1]]

# have a look at the first 10 entries
kable(ZU_weather[1:10, ]) %>%
        kable_styling("striped", position = "left", font_size = 10)%>%
        scroll_box(width = "100%")
```

DATE	Year	Month	Day	Tcontinue	Temp_inter
2019-04-01 01:00:00	2019	4	1	3.4350000	-0.420
2019-04-01 02:00:00	2019	4	1	2.0900000	-0.420
2019-04-01 03:00:00	2019	4	1	1.2616667	-0.420
2019-04-01 04:00:00	2019	4	1	0.6150000	-0.420
2019-04-01 05:00:00	2019	4	1	0.0583333	-0.420
2019-04-01 06:00:00	2019	4	1	-0.3566667	-0.420
2019-04-01 07:00:00	2019	4	1	-0.4200000	-0.420
2019-04-01 08:00:00	2019	4	1	0.9666667	1.403
2019-04-01 09:00:00	2019	4	1	3.9333334	3.226
2019-04-01 10:00:00	2019	4	1	6.4183333	5.049

Thus, all gaps in the column `Temp_inter` are filled by linear interpolation. The interpolation is performed between the gaps.

The non-linear interpolation method considers the sun's position at the respective location in the interpolation. In addition, the `stack_hourly_temps()` function requires a dataset as input that only contains `Tmax` and `Tmin` values. In this example, this dataset is called `ZU_weather_min_max` and consists of five columns: `Year`, `Month`, `Day`, `Tmax`, and `Tmin`.

```
# create dataframe for non-linear interpolation
ZU_weather_min_max = data.frame(
  Year = as.numeric(substr(Zuelpich_min_max[, 2], 1, 4)),
  Month = as.numeric(substr(Zuelpich_min_max[, 2], 6, 7)),
  Day = as.numeric(substr(Zuelpich_min_max[, 2], 9, 10)),
  Tmax = final[, 3],
  Tmin = final[, 4]
)
kable(ZU_weather_min_max[1:10,])%>%
      kable_styling("striped", position = "left", font_size = 10)%>%
      scroll_box(width = "100%")
```

Year	Month	Day	Tmax	Tmin
2019	2	2	17.810000	-0.420000
2019	2	3	17.096667	2.693333
2019	2	4	9.983333	5.481667
2019	2	5	7.763333	3.566667
2019	2	6	17.810000	-0.420000
2019	2	7	17.096667	2.693333
2019	2	8	9.983333	5.481667
2019	2	9	7.763333	3.566667
2019	2	10	17.810000	-0.420000
2019	2	11	17.096667	2.693333

The function `stack_hourly_temps()` can be passed the entire dataset `ZU_weather_min_max`. This function sets the `Tmax` values to 6:00 PM and the Tmin values to 6:00 AM, without performing any interpolation between the times when the `Tmax` and `Tmin` values actually occurred. The resulting interpolation is written to a new dataset, `ZU_hourly`. A new column called `DATE` is then created in this dataset to store the date, and the first row is removed due to an index shift.

```
ZU_hourly = stack_hourly_temps(ZU_weather_min_max, latitude = 50.4)

ZU_hourly$hourtemps[, "DATE"] =
  ISOdate(
    ZU_hourly$hourtemps$Year,
    ZU_hourly$hourtemps$Month,
    ZU_hourly$hourtemps$Day,
    ZU_hourly$hourtemps$Hour
  )

ZU_hourly_mod = ZU_hourly[[1]][-1, ]
```

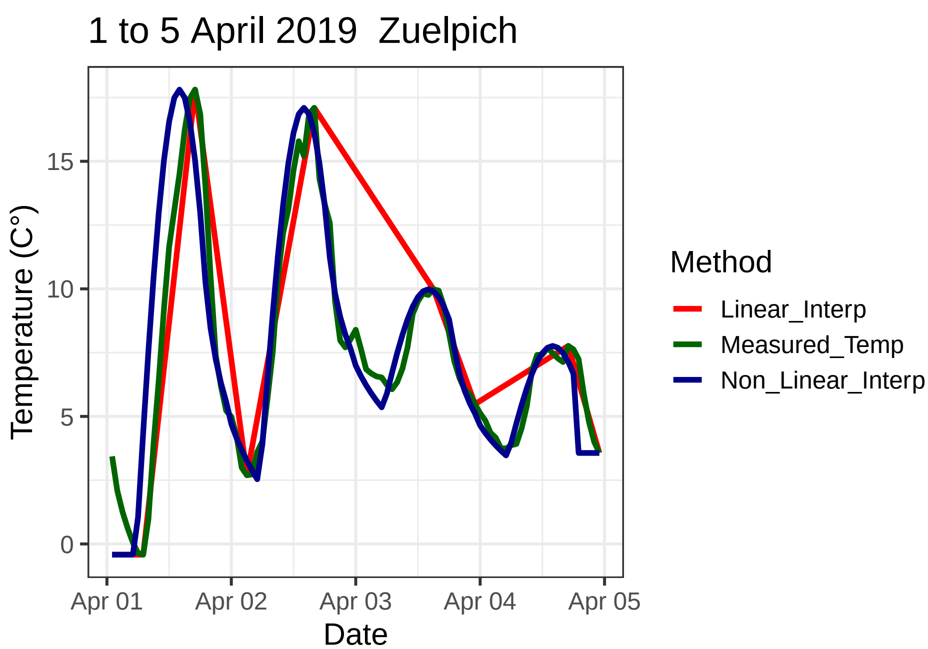
```r
kable(ZU_hourly_mod[1:10,])%>%
      kable_styling("striped", position = "left", font_size = 10)%>%
      scroll_box(width = "100%")
```

	Year	Month	Day	Tmax	Tmin	JDay	Hour	Temp	DATE
609	2019	2	2	17.81	-0.42	33	1	-0.420000	2019-02-02 01:00:00
1217	2019	2	2	17.81	-0.42	33	2	-0.420000	2019-02-02 02:00:00
1825	2019	2	2	17.81	-0.42	33	3	-0.420000	2019-02-02 03:00:00
2433	2019	2	2	17.81	-0.42	33	4	-0.420000	2019-02-02 04:00:00
3041	2019	2	2	17.81	-0.42	33	5	-0.420000	2019-02-02 05:00:00
3649	2019	2	2	17.81	-0.42	33	6	-0.420000	2019-02-02 06:00:00
4257	2019	2	2	17.81	-0.42	33	7	-0.420000	2019-02-02 07:00:00
4865	2019	2	2	17.81	-0.42	33	8	2.355706	2019-02-02 08:00:00
5473	2019	2	2	17.81	-0.42	33	9	6.496980	2019-02-02 09:00:00
6081	2019	2	2	17.81	-0.42	33	10	10.253744	2019-02-02 10:00:00

Finally, a dataset is generated containing the actual measured temperature data, as well as the interpolated values calculated using both linear and non-linear interpolation. The results can be effectively visualized in a plot.

```r
# final_df = data.frame(
#   DATE = zuelpich_april[, 4],
#   Measured_Temp = zuelpich_april[, 2],
#   Linear_Interp  = ZU_weather[, 6],
#   Non_Linear_Interp = ZU_hourly_mod[, 8]
# )
#write.csv(final_df, "weather_data/final_df_non_linear.csv")

# read final dataframe
final_df_m = read.table("weather_data/final_df_non_linear.csv",
                        header = TRUE,
                        sep = ",")

#remove index
final_df_mx = final_df_m[, -1]

#generate Date
final_df_mx$DATE =  as.POSIXct(final_df_mx$DATE)

#create pivot table
final_df_mod  = pivot_longer(final_df_mx,
                             -"DATE",
                             names_to = "Method",
                             values_to = "Temperature")
```

```
#plot final result and compare methods
ggplot(data = final_df_mod, aes(x = DATE, y = Temperature
                                , colour = Method)) +
  geom_line(lwd = 1.3) +
  labs(x = "Date", y = "Temperature (C°)") +
  ggtitle("1 to 5 April 2019  Zuelpich") +
  scale_color_manual(values = c("red", "darkgreen", "darkblue")) +
  #facet_wrap(vars(Method)) +
  theme_bw(base_size = 15)
```



Chapter 8

Getting temperature data

8.1 Task 1

Choose a location of interest and find the 25 closest weather stations using the handle_gsod function

> I have decided to conduct a phonology analysis at a location in Poland. The city of Glogau, situated on the banks of the Oder, was chosen as the starting point. For more information about the location, the following website can be visited here. The climate at the location is relatively similar to Bonn, although the climate is generally more continental, which can be reflected in colder winters and drier, slightly warmer summers. Here is a picture of a typical landscape in Lower Silesia in Poland.
>
> Beautiful Glogau City
>
> Ok, here really beautiful old city of Glogau

```r
# station_list_poland = handle_gsod(action="list_stations",
#                               location=c(16.5,51.39),
#                               time_interval=c(1990,2020))
# kable(station_list_poland[1:10,]) %>% kable_styling("striped", position = "left", font_size = 1

# write.csv(station_list_poland,"weather_data/Poland_station_list.csv",row.names=FALSE)

list_poland_weather = read.table("weather_data/Poland_station_list.csv", header = TRUE, sep=",")
kable(list_poland_weather[1:10,], caption = "Station List") %>%
  kable_styling("striped", position = "left", font_size = 10)%>%
  scroll_box(width = "100%")
```

Table 8.1: Station List

chillR_code	STATION.NAME	CTRY	Lat	Long	BEGIN	END
124170_99999	LUBIN MIASTO	PL	51.417	16.200	19390810	19411231
124150_99999	LEGNICA	PL	51.200	16.200	19370401	20221110
124250_99999	WROCLAW I	PL	51.117	16.883	19280101	20220614
124240_99999	STRACHOWICE	PL	51.103	16.886	19390102	20221110
124280_99999	WROCLAW/STRACHOWICE	PL	51.100	16.883	20020601	20220901
124160_99999	WSCHOWA	PL	51.800	16.317	19360102	19420630
124180_99999	LESZNO	PL	51.833	16.533	19730101	20211212
125010_99999	MIROSLAWICE	PL	50.950	16.767	19390901	19420628
125230_99999	SWIDNICA	PL	50.850	16.483	19410114	19431231
125220_99999	SLEZA	PL	50.867	16.717	19400414	19420630

8.2 Task 2

Download weather data for the most promising station on the list

The most promising station for obtaining continuous weather recording is represented by the weather station in Leszno, located approximately 50 kilometers (by air) from Glogau. This location is listed as the seventh entry in the "Station List" table.

To avoid constantly loading the weather data, it is stored in the `weather_poland_leszno` variable. The file is then saved as a CSV and read in using the `read.table()` function.

```
# weather_poland_leszno <- handle_gsod(
#   action = "download_weather",
#   location = station_list_poland$chillR_code[7],
#   time_interval = c(1990, 2020))


# kable(weather_poland_leszno[[1]][[2]][1:10, ]) %>%
#   kable_styling("striped", position = "left", font_size = 10)

#write.csv(weather_poland_leszno[[1]][[2]],"weather_data/Poland_leszno_weather.csv",ro

weather_poland_leszno_place =
  read.table("weather_data/Poland_leszno_weather.csv", header = TRUE, sep=",")

kable(weather_poland_leszno_place[1:10, ], caption = "Weather Data Leszno ") %>%
   kable_styling("striped", position = "left", font_size = 10)%>%
   scroll_box(width = "100%")
```

Table 8.2: Weather Data Leszno

DATE	STN...	WBAN	YEAR	MONTH	DAY	TEMP	Count1	DEWP	Count2
1990-01-01 12:00:00	124180	99999	1990	1	1	-1.1111111	7	28.6	7
1990-01-02 12:00:00	124180	99999	1990	1	2	-1.7222222	8	25.9	8
1990-01-03 12:00:00	124180	99999	1990	1	3	-1.7777778	7	25.6	7
1990-01-04 12:00:00	124180	99999	1990	1	4	-2.4444444	7	22.5	7
1990-01-05 12:00:00	124180	99999	1990	1	5	-2.3888889	7	22.6	7
1990-01-06 12:00:00	124180	99999	1990	1	6	-4.7777778	7	21.3	7
1990-01-07 12:00:00	124180	99999	1990	1	7	-7.3333333	8	15.4	8
1990-01-08 12:00:00	124180	99999	1990	1	8	-4.5555556	8	18.0	8
1990-01-09 12:00:00	124180	99999	1990	1	9	0.8333333	6	33.1	6
1990-01-10 12:00:00	124180	99999	1990	1	10	2.7222222	7	36.3	7

Table 8.3: Cleaned Weather Data Leszno

DATE	Year	Month	Day	Tmin	Tmax	Tmean	Prec
1990-01-01 12:00:00	1990	1	1	-2.000000	-0.6111111	-1.1111111	0.000
1990-01-02 12:00:00	1990	1	2	-2.111111	-1.2777778	-1.7222222	0.000
1990-01-03 12:00:00	1990	1	3	-2.611111	-0.7777778	-1.7777778	0.000
1990-01-04 12:00:00	1990	1	4	-5.277778	-0.7777778	-2.4444444	0.000
1990-01-05 12:00:00	1990	1	5	-7.722222	-0.3888889	-2.3888889	0.000
1990-01-06 12:00:00	1990	1	6	-7.000000	0.0000000	-4.7777778	0.000
1990-01-07 12:00:00	1990	1	7	-11.611111	-2.7222222	-7.3333333	0.000
1990-01-08 12:00:00	1990	1	8	-10.500000	1.1111111	-4.5555556	0.000
1990-01-09 12:00:00	1990	1	9	-1.777778	2.1111111	0.8333333	0.000
1990-01-10 12:00:00	1990	1	10	1.500000	3.2777778	2.7222222	0.254

8.3 Task 3

Convert the weather data into chillR format

```
# weather_pl <- weather_poland_leszno$LESZNO[[2]]
# cleaned_weather_pl <- handle_gsod(weather_pl)

# kable(cleaned_weather_pl[1:20,]) %>%
#   kable_styling("striped", position = "left", font_size = 10)

#write.csv(cleaned_weather_pl,"weather_data/Poland_leszno_chillR_weather.csv",row.names=FALSE)

cleaned_weather_pl_leszno = read.table("weather_data/Poland_leszno_chillR_weather.csv", header =

kable(cleaned_weather_pl_leszno[1:10, ], caption = "Cleaned Weather Data Leszno") %>%
  kable_styling("striped", position = "left", font_size = 10)
```

Chapter 9

Filling gaps in temperature records

9.1 Task 1

Use chillR functions to find out how many gaps you have in this dataset (even if you have none, please still follow all further steps)

```
Leszno = read.csv("weather_data/Poland_leszno_chillR_weather.csv")

# detected many gaps
Leszno_QC = fix_weather(Leszno)$QC

kable(Leszno_QC, caption = "Quality Check  for Data Leszno ") %>%
  kable_styling("striped", position = "left", font_size = 10) %>%
  scroll_box(width = "100%")
```

> Upon closer examination, the seemingly promising location of Leszno exhibits many gaps in temperature data, particularly after 2005. These gaps must be filled with temperature data from neighboring stations.

9.2 Task 2

Create a list of the 25 closest weather stations using the handle_gsod function

```
# station_list_close_to_leszno<-
#   handle_gsod(action="list_stations",location=c(16.57,51.85),time_interval=c(1990,2020))
```

Table 9.1: Quality Check for Data Leszno

Season	End_year	Season_days	Data_days	Missing_Tmin	Missing_Tmax	Incomplete
1989/1990	1990	365	365	1	1	
1990/1991	1991	365	365	26	26	
1991/1992	1992	366	366	284	284	
1992/1993	1993	365	365	140	140	
1993/1994	1994	365	365	57	57	
1994/1995	1995	365	365	4	4	
1995/1996	1996	366	366	5	5	
1996/1997	1997	365	365	75	75	
1997/1998	1998	365	365	140	140	
1998/1999	1999	365	365	39	39	
1999/2000	2000	366	366	1	1	
2000/2001	2001	365	365	23	23	
2001/2002	2002	365	365	4	4	
2002/2003	2003	365	365	1	1	
2003/2004	2004	366	366	2	2	
2004/2005	2005	365	365	365	365	
2005/2006	2006	365	365	365	365	
2006/2007	2007	365	365	365	365	
2007/2008	2008	366	366	366	366	
2008/2009	2009	365	365	365	365	
2009/2010	2010	365	365	365	365	
2010/2011	2011	365	365	365	365	
2011/2012	2012	366	366	366	366	
2012/2013	2013	365	365	365	365	
2013/2014	2014	365	365	365	365	
2014/2015	2015	365	365	365	365	
2015/2016	2016	366	366	366	366	
2016/2017	2017	365	365	365	365	
2017/2018	2018	365	365	365	365	
2018/2019	2019	365	365	365	365	
2019/2020	2020	366	366	366	366	

Table 9.2: List of GSOD weather stations close to Leszno

X	chillR_code	STATION.NAME	CTRY	Lat	Long	BEGIN	END	distanc
4162	124180_99999	LESZNO	PL	51.833	16.533	19730101	20211212	3.1
4160	124160_99999	WSCHOWA	PL	51.800	16.317	19360102	19420630	18.3
4161	124170_99999	LUBIN MIASTO	PL	51.417	16.200	19390810	19411231	54.5
4122	123260_99999	KRZESINY	PL	52.332	16.966	20020601	20230103	60.1
4153	124030_99999	NOWE MIASTECZKO	PL	51.683	15.733	19360102	19420630	60.7
4118	123120_99999	BABIMOST	PL	52.139	15.799	20020603	20230103	61.9
4124	123300_99999	LAWICA	PL	52.421	16.826	19310102	20230103	65.9
4163	124190_99999	JAROCIN	PL	51.967	17.550	19391113	19420630	68.6
4150	124000_99999	ZIELONA GORA	PL	51.933	15.533	19310101	20230103	71.9
4156	124060_99999	SZPROTAWA-WIECHLICE	PL	51.561	15.585	19390815	20191108	75.3

```r
# write.csv(station_list_close_to_leszno,"station_list_close_to_leszno.csv" )

station_list_close_to_leszno = read.csv("station_list_close_to_leszno.csv")

kable(station_list_close_to_leszno[1:10,],
      caption="List of GSOD weather stations close to Leszno")%>%
  kable_styling("striped", position = "left", font_size = 10) %>%
  scroll_box(width = "100%")
```

9.3 Task 3

Identify suitable weather stations for patching gaps

After reviewing the nearest weather stations to the Leszno location, KRZESINY, BABIMOST and LAWICA stations were identified as suitable for filling in the gaps in temperature data. These stations are ranked 4th, 6th and 7th on the list.