

Assignment 1 - Web scraping & basic data analysis

Tobias Abraham Haider

2025-04-11

Contents

Preparation	1
Imports	1
Global variables	2
Data download	2
Analyses	2
48 hour weather report	2
24 hour weather report	4
Annual weather report	9
Conclusion	13

This is my submission for the first assignment done for the course *Data Acquisition and Survey Methods* during the summer semester 2025.

Preparation

Imports

The assignment is done mainly using the libraries `rvest` for data extraction from the html and `dplyr` for processing of the data table.

```
library(knitr)
library(glue)
library(lubridate)
library(stringr)
library(rvest)
library(dplyr)
library(ggplot2)
```

Global variables

As specified, the web page `timeanddate.com` is used as a data source. My name is Tobias Haider. My choice for the city is Tallinn, Estonia, a city that I like very much and has weather that is a little different from the weather in Vienna.

Using `rvest`, it is easy to load all the web pages used in the analyses below. Sources:

- Tallinn weather forecast (next 48 hours)
- Tallinn weather detail forecast (next 24 hours)
- Tallinn weather yearly summary

```
time_now <- Sys.time()

country_name <- "estonia"
city_name <- "tallinn"

weather_forecast_48h_url <- glue("https://www.timeanddate.com/weather/{country_name}/{city_name}")
weather_forecast_detail_24h_url <- glue("https://www.timeanddate.com/weather/{country_name}/{city_name}")
weather_summary_annual_url <- glue("https://www.timeanddate.com/weather/{country_name}/{city_name}/climate")
```

Data download

With the `read_html` function a GET request is made and the response body is parsed as a html. I am not interested in any status codes and just assume that the data downloads succeed.

```
weather_forecast_48h_html <- read_html(weather_forecast_48h_url)
weather_forecast_detail_24h_html <- read_html(weather_forecast_detail_24h_url)
weather_summary_annual_html <- read_html(weather_summary_annual_url)
```

Analyses

Starting from this section, the pre-loaded html is processed. The strategy is the following for all reports:

- Locate the relevant element within the web page
- Parse the table representation into a tibble
- Clean up the representation so that the result is a valid data table

48 hour weather report

The main weather report is easily accessible. The page contains a `table` tag with a static id `wt-48` (probably an abbreviation for weather table 48 hours). It can directly be transformed to a tibble without custom parsing.

Afterwards, it is transposed and some columns are removed.

```
weather_forecast_48h_parsed <- weather_forecast_48h_html %>%
  html_element("table#wt-48") %>%
  html_table(na.strings = "-") %>%
  t() %>%
```

Forecast for the next 48 hours








	Friday	Saturday				Sunday	
	Evening	Night	Morning	Afternoon	Evening	Night	Morning
Forecast							
Temperature	1 °C	-3 °C	0 °C	6 °C	4 °C	1 °C	4 °C
	Mostly clear.	Partly cloudy.	Overcast.	Sprinkles. Overcast.	Partly cloudy.	Clear.	Sunny.
Feels Like	-3 °C	-8 °C	-5 °C	1 °C	1 °C	-2 °C	2 °C
Wind Speed	12 km/h	11 km/h	23 km/h	27 km/h	14 km/h	8 km/h	8 km/h
Wind Direction	NNW ↓	SSW ↑	SSW ↑	WSW ↗	W →	WNW ↘	SSW ↑
Humidity	58%	76%	73%	85%	70%	83%	66%
Dew Point	-7 °C	-7 °C	-4 °C	3 °C	-1 °C	-2 °C	-2 °C
Visibility	11 km	9 km	9 km	10 km	7 km	10 km	12 km
Probability of Precipitation	0%	0%	1%	6%	5%	0%	0%
Amount of Rain	-	-	-	0.3 mm	-	-	-
* Updated Friday, 11 April 2025 16:42:56 Tallinn time - Weather by CustomWeather, © 2025							

Figure 1: 48h forecast web page

```
as_tibble() %>%
slice(2:n()) %>%
select(3:12)

names(weather_forecast_48h_parsed) <- c("temp", "description", "feels_like", "wind_speed", "wind_direct
```

Personally, I do not like having the weekday and the strings `morning`, `afternoon`, `evening` and `night` in the table, because this data will lose its meaning if time passes. To avoid confusion, I removed these columns and replace them by calculated time stamps estimating the expected time for which the forecast is made.

```
weather_forecast_48h <- weather_forecast_48h_parsed %>%
  mutate(time = time_now + seq(6, 42, by = 6) * 3600) %>% # 6 hour interval
  relocate(time, .before = 1)

kable(weather_forecast_48h)
```

time	temp	description	feels_like	wind_speed	wind_direction	humidity	dew_point	visibility	precip_prob	rain_amount
2025-04-12 01:57:40	1 °C	Mostly clear.	-3 °C	12 km/h	NNW↑	58%	-7 °C	11 km	0%	NA
2025-04-12 07:57:40	- 3 °C	Partly cloudy.	-8 °C	11 km/h	SSW↑	76%	-7 °C	9 km	0%	NA
2025-04-12 13:57:40	0 °C	Overcast.	-5 °C	23 km/h	SSW↑	73%	-4 °C	9 km	1%	NA
2025-04-12 19:57:40	6 °C	Sprinkles. Overcast.	1 °C	27 km/h	WSW↑	85%	3 °C	10 km	6%	0.3 mm
2025-04-13 01:57:40	4 °C	Partly cloudy.	1 °C	14 km/h	W↑	70%	-1 °C	7 km	5%	NA
2025-04-13 07:57:40	1 °C	Clear.	-2 °C	8 km/h	WNW↑	83%	-2 °C	10 km	0%	NA
2025-04-13 13:57:40	4 °C	Sunny.	2 °C	8 km/h	SSW↑	66%	-2 °C	12 km	0%	NA

The result is a nice table containing all the information from the web page. It can now be used for reports and exports to other systems. Note that this table is still not a valid data table, because the entries still contain a lot of free text and unit characters. This is done in the next tasks and opens up the possibility to plot the data columns.

24 hour weather report

The detailed forecast is also easily accessible. The `table` element with the id `wt-hbh` holds all the information we need.

```
weather_forecast_detail_24h_parsed <- weather_forecast_detail_24h_html %>%
  html_element("table#wt-hbh") %>%
  html_table(na.strings = "-") %>%
  as.matrix() %>% # there are columns with no name that prevent indexing
  as_tibble(.name_repair = "unique") %>%
  slice(3:n() - 1) %>%
  select(c(-2, -7))

names(weather_forecast_detail_24h_parsed) <- c("time", "temp", "description", "feels_like", "wind_speed", "wind_direct
```

Detailed Hourly Forecast — Next 24 hours

Show weather on: Next 24 hours ▼








Time	Conditions		Comfort			Precipitation	
	Temp	Weather	Feels Like	Wind	Humidity	Chance	Amount
21:00 Fri, 11 Apr	 1 °C	Mostly clear.	-3 °C	12 km/h ↘	58%	0%	-
22:00	 -1 °C	Scattered clouds.	-4 °C	9 km/h ↓	65%	0%	-
23:00	 -2 °C	Broken clouds.	-4 °C	6 km/h ↓	71%	0%	-
00:00 Sat, 12 Apr	 -3 °C	Overcast.	-3 °C	3 km/h ↓	77%	0%	-
01:00	 -4 °C	Mostly cloudy.	-4 °C	2 km/h ↗	81%	0%	-
02:00	 -4 °C	Broken clouds.	-7 °C	6 km/h ↗	79%	0%	-
03:00	 -3 °C	Partly cloudy.	-8 °C	11 km/h ↗	76%	0%	-

Figure 2: 24h forecast web page

For the time stamps, the same approach as before can be applied. First, the initial forecast time is calculated. All following forecasts have an interval of one hour.

```
forecast_24h_start_time_hour_minute <- str_split(
  str_extract(weather_forecast_detail_24h_parsed$time[1], "\\d{1,2}:\\d{2}"),
  ":",
  simplify = TRUE
)
weather_forecast_detail_24h_start_time <- update(
  time_now,
  hour = as.integer(forecast_24h_start_time_hour_minute[1]),
  minute = as.integer(forecast_24h_start_time_hour_minute[2]),
  second = 0
)
if (weather_forecast_detail_24h_start_time < time_now) {
  weather_forecast_detail_24h_start_time <- weather_forecast_detail_24h_start_time + days(1)
}
```

To be able to plot the values, all units are removed from the entries.

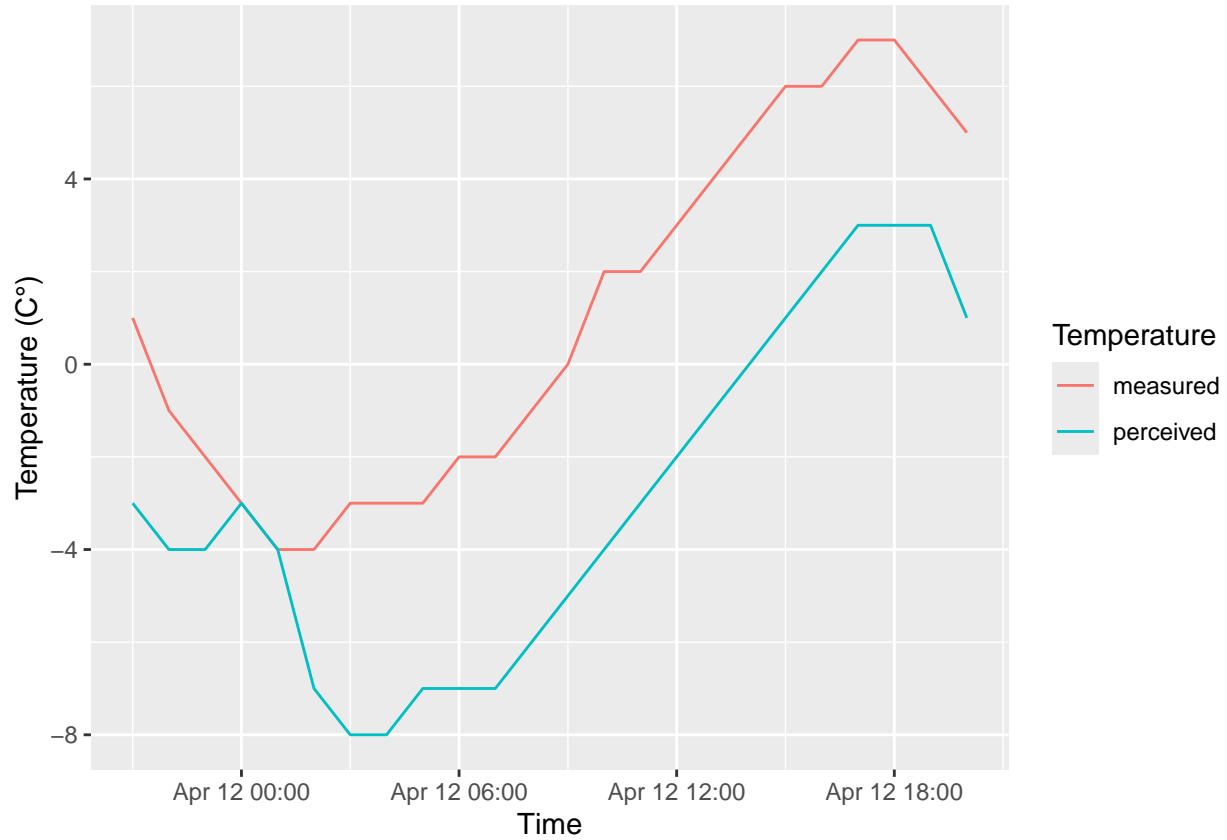
```
weather_forecast_detail_24h <- weather_forecast_detail_24h_parsed %>%
  mutate(
    time = weather_forecast_detail_24h_start_time + hours(row_number() - 1),
    temp = as.numeric(str_extract(temp, "-?[\\d\\.]+")),
    feels_like = as.numeric(str_extract(feels_like, "-?[\\d\\.]+")),
    wind_speed = as.numeric(str_extract(wind_speed, "[\\d\\.]+")),
    humidity = as.numeric(str_extract(humidity, "[\\d\\.]+")) / 100,
    precip_prob = as.numeric(str_extract(precip_prob, "[\\d\\.]+")) / 100,
    precip_amount = as.numeric(str_extract(precip_amount, "[\\d\\.]+"))
  )

kable(weather_forecast_detail_24h)
```

time	temp	description	feels_like	wind_speed	humidity	precip_prob	precip_amount
2025-04-11 21:00:00	1	Mostly clear.	-3	12	0.58	0.00	NA
2025-04-11 22:00:00	-1	Scattered clouds.	-4	9	0.65	0.00	NA
2025-04-11 23:00:00	-2	Broken clouds.	-4	6	0.71	0.00	NA
2025-04-12 00:00:00	-3	Overcast.	-3	3	0.77	0.00	NA
2025-04-12 01:00:00	-4	Mostly cloudy.	-4	2	0.81	0.00	NA
2025-04-12 02:00:00	-4	Broken clouds.	-7	6	0.79	0.00	NA
2025-04-12 03:00:00	-3	Partly cloudy.	-8	11	0.76	0.00	NA
2025-04-12 04:00:00	-3	Broken clouds.	-8	12	0.76	0.00	NA
2025-04-12 05:00:00	-3	Mostly cloudy.	-7	14	0.74	0.00	NA
2025-04-12 06:00:00	-2	Overcast.	-7	15	0.72	0.00	NA
2025-04-12 07:00:00	-2	Overcast.	-7	18	0.74	0.00	NA
2025-04-12 08:00:00	-1	Overcast.	-6	20	0.74	0.01	NA
2025-04-12 09:00:00	0	Overcast.	-5	23	0.73	0.02	NA
2025-04-12 10:00:00	2	Overcast.	-4	25	0.73	0.03	NA
2025-04-12 11:00:00	2	Overcast.	-3	27	0.75	0.05	NA
2025-04-12 12:00:00	3	Overcast.	-2	31	0.76	0.05	NA
2025-04-12 13:00:00	4	Overcast.	-1	29	0.78	0.05	NA
2025-04-12 14:00:00	5	Overcast.	0	28	0.84	0.04	NA
2025-04-12 15:00:00	6	Overcast.	1	27	0.85	0.06	NA
2025-04-12 16:00:00	6	Sprinkles. Overcast.	2	25	0.78	0.20	0.1
2025-04-12 17:00:00	7	Sprinkles. Overcast.	3	24	0.71	0.26	0.1
2025-04-12 18:00:00	7	Drizzle. Overcast.	3	22	0.66	0.21	0.1
2025-04-12 19:00:00	6	Cloudy.	3	20	0.65	0.10	0.0
2025-04-12 20:00:00	5	Mostly cloudy.	1	17	0.67	0.00	NA

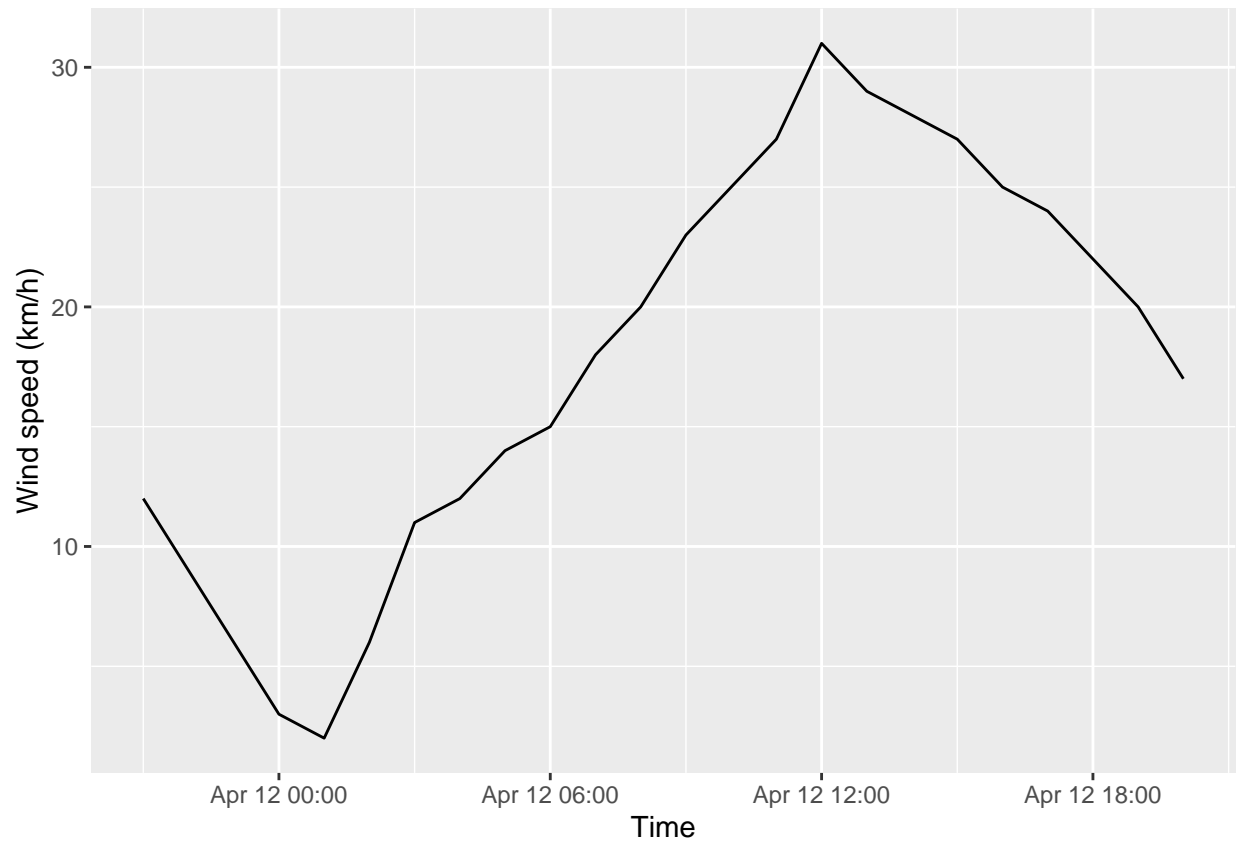
24 hour temperature forecast plot

```
weather_forecast_detail_24h %>% ggplot(aes(x = time)) +  
  geom_line(aes(y = temp, color = "measured")) +  
  geom_line(aes(y = feels_like, color = "perceived")) +  
  labs(x = "Time", y = "Temperature (C°)", color = "Temperature")
```



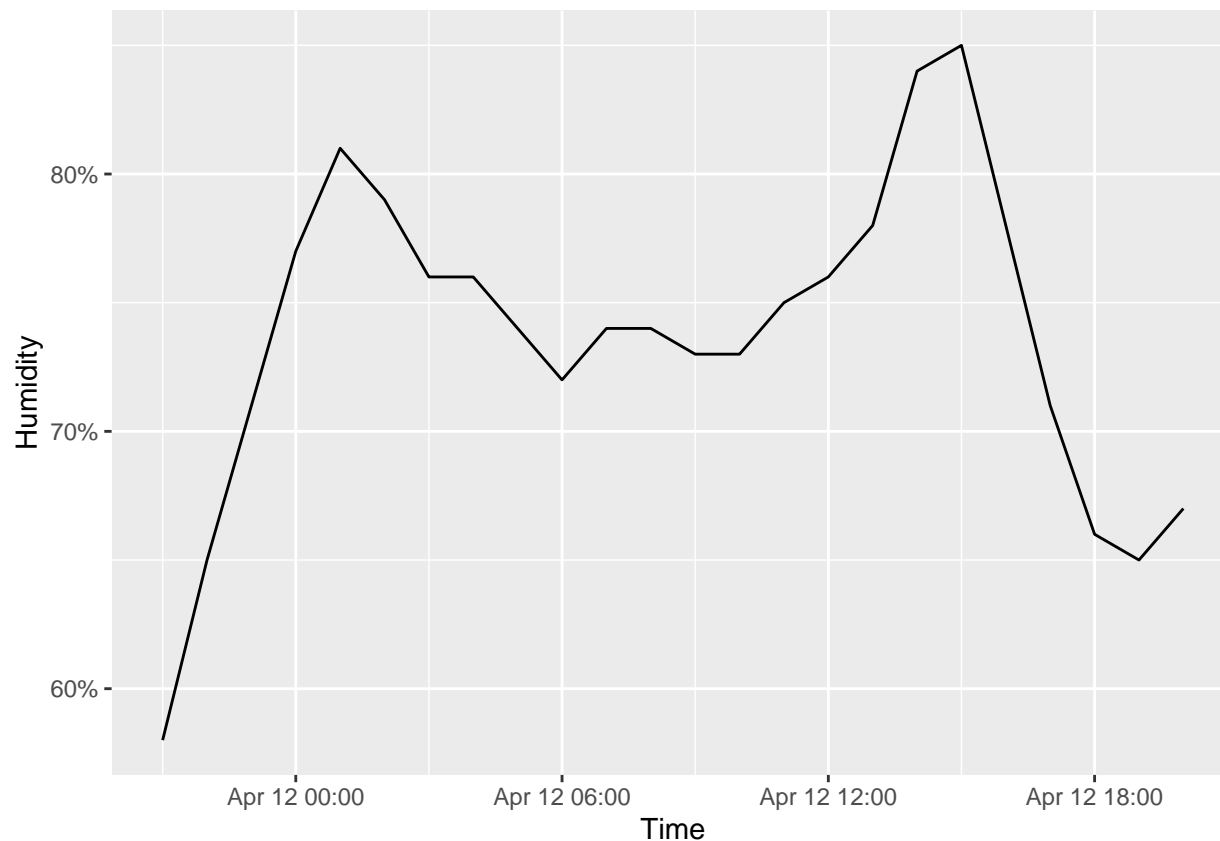
24 hour wind speed forecast plot

```
weather_forecast_detail_24h %>% ggplot(aes(x = time, y = wind_speed)) +  
  geom_line() +  
  labs(x = "Time", y = "Wind speed (km/h)")
```



24 hour humidity forecast plot

```
weather_forecast_detail_24h %>% ggplot(aes(x = time, y = humidity)) +  
  scale_y_continuous(labels = scales::percent) +  
  geom_line() +  
  labs(x = "Time", y = "Humidity")
```

Annual weather report

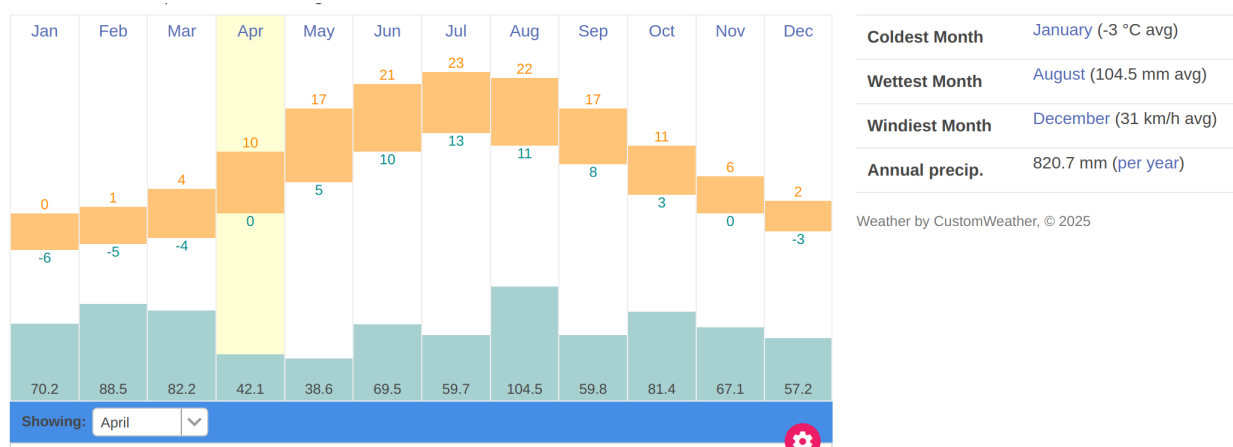


Figure 3: Annual weather report web page

Scraping data for the annual overview of the weather is quite tricky. The reason is the `div` elements that do not follow a common table structure. Also, the cells never just contain the clean numbers, but always some text or description that has no value.

Still, it is possible to retrieve the data in a rather fast way. My approach was to rely on the given `div` structure and, most importantly, the order of the fields. With the CSS selector one can easily select all `p`

tags containing the summary data. The result is a 1D flattened array with the size of 108 (12 * 9). For each month, there are exactly 9 values present:

- high_temp (first occurrence at index 1)
- low_temp (first occurrence at index 2)
- mean_temp (first occurrence at index 3)
- precipitation (first occurrence at index 4)
- humidity (first occurrence at index 5)
- dew_point (first occurrence at index 6)
- wind (first occurrence at index 7)
- pressure (first occurrence at index 8)
- visibility (first occurrence at index 9)

```
months_html <- weather_summary_annual_html %>%
  html_elements("section#climateTable > div.climate-month:not(.climate-month--allyear) > div.four > p")
  html_text()

weather_summary_annual_parsed <- tibble(
  month = seq(1, 12),
  high_temp = months_html[seq(1, 108, by = 9)],
  low_temp = months_html[seq(2, 108, by = 9)],
  mean_temp = months_html[seq(3, 108, by = 9)],
  precipitation = months_html[seq(4, 108, by = 9)],
  humidity = months_html[seq(5, 108, by = 9)],
  dew_point = months_html[seq(6, 108, by = 9)],
  wind = months_html[seq(7, 108, by = 9)],
  pressure = months_html[seq(8, 108, by = 9)],
  visibility = months_html[seq(9, 108, by = 9)]
)
```

Every cell contains a description that can be removed. The simplest approach is to extract the number present using a regex. I do not exactly know when the values are floating points and when not. Because of this, all floating point numbers are accepted.

For personal preference, percentage values are divided by 100.

```
weather_summary_annual <- weather_summary_annual_parsed %>%
  mutate(
    high_temp = as.numeric(str_extract(high_temp, "-?[\\d\\.]+")),
    low_temp = as.numeric(str_extract(low_temp, "-?[\\d\\.]+")),
    mean_temp = as.numeric(str_extract(mean_temp, "-?[\\d\\.]+")),
    precipitation = as.numeric(str_extract(precipitation, "[\\d\\.]+")),
    humidity = as.numeric(str_extract(humidity, "[\\d\\.]+")) / 100,
    dew_point = as.numeric(str_extract(dew_point, "-?[\\d\\.]+")),
    wind = as.numeric(str_extract(wind, "[\\d\\.]+")),
    pressure = as.numeric(str_extract(pressure, "[\\d\\.]+")),
    visibility = as.numeric(str_extract(visibility, "[\\d\\.]+"))
  )

kable(weather_summary_annual)
```

month	high_temp	low_temp	mean_temp	precipitation	humidity	dew_point	wind	pressure	visibility
1	0	-6	-3	70.2	0.90	-5	28	1011	9
2	1	-5	-2	88.5	0.88	-4	28	1013	9
3	4	-4	0	82.2	0.78	-4	30	1013	12
4	10	0	5	42.1	0.70	-1	29	1013	14
5	17	5	11	38.6	0.66	5	26	1015	12
6	21	10	15	69.5	0.70	10	27	1013	12
7	23	13	18	59.7	0.73	13	26	1012	12
8	22	11	17	104.5	0.77	13	24	1014	11
9	17	8	13	59.8	0.82	9	26	1015	10
10	11	3	7	81.4	0.85	4	27	1015	10
11	6	0	3	67.1	0.90	2	28	1013	9
12	2	-3	0	57.2	0.89	-2	31	1010	9

Annual weather metric summary

With the clean data table at hand, the summary can be created with `summarise` masic math function.

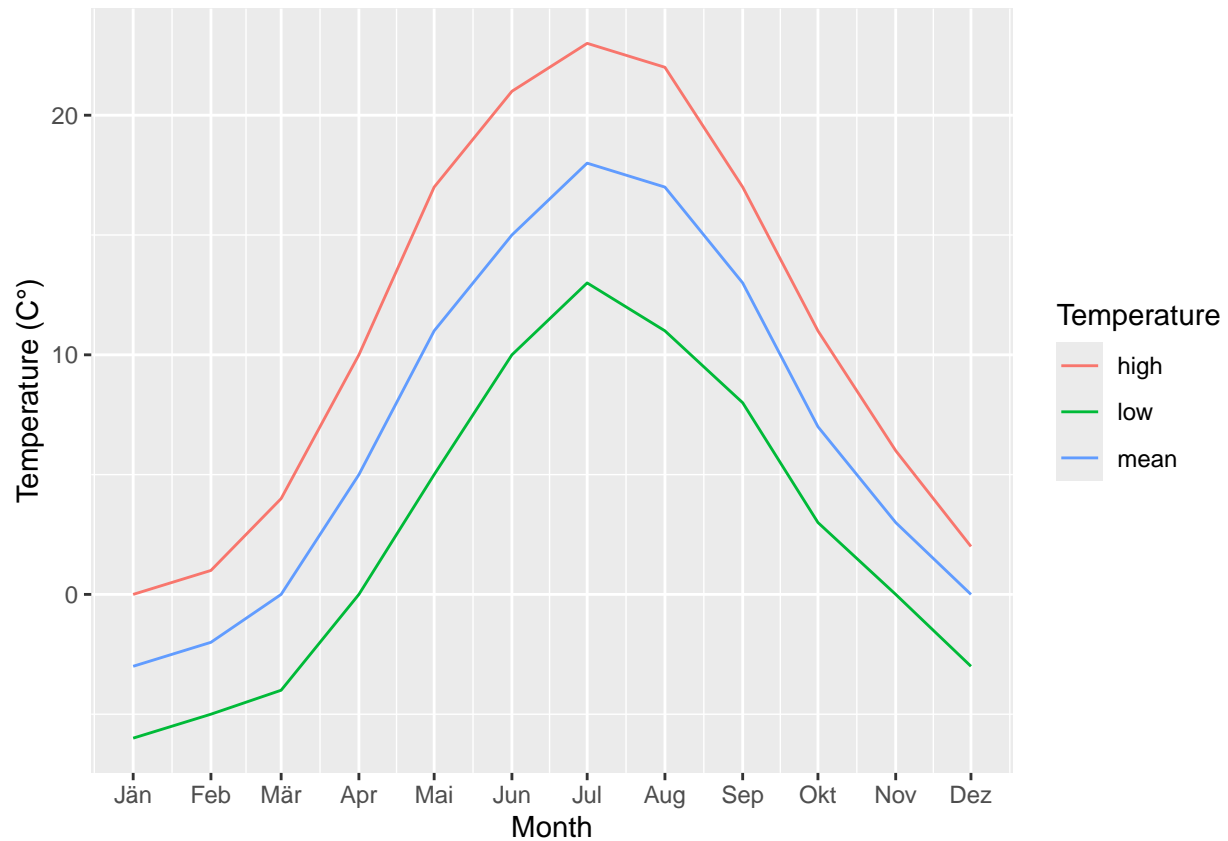
```
weather_summary_annual %>% summarise(
  minimum_temp = min(low_temp),
  maximum_temp = max(high_temp),
  mean_temp = mean(mean_temp),
  mean_precipitation = mean(precipitation)
) %>%
  t() %>%
  kable()
```

minimum_temp	-6.0
maximum_temp	23.0
mean_temp	7.0
mean_precipitation	68.4

Annual weather temperature plot

The temperature is plotted as a line plot. By plotting min, max and mean values with different colors, the spread of temperatures is clearly visible.

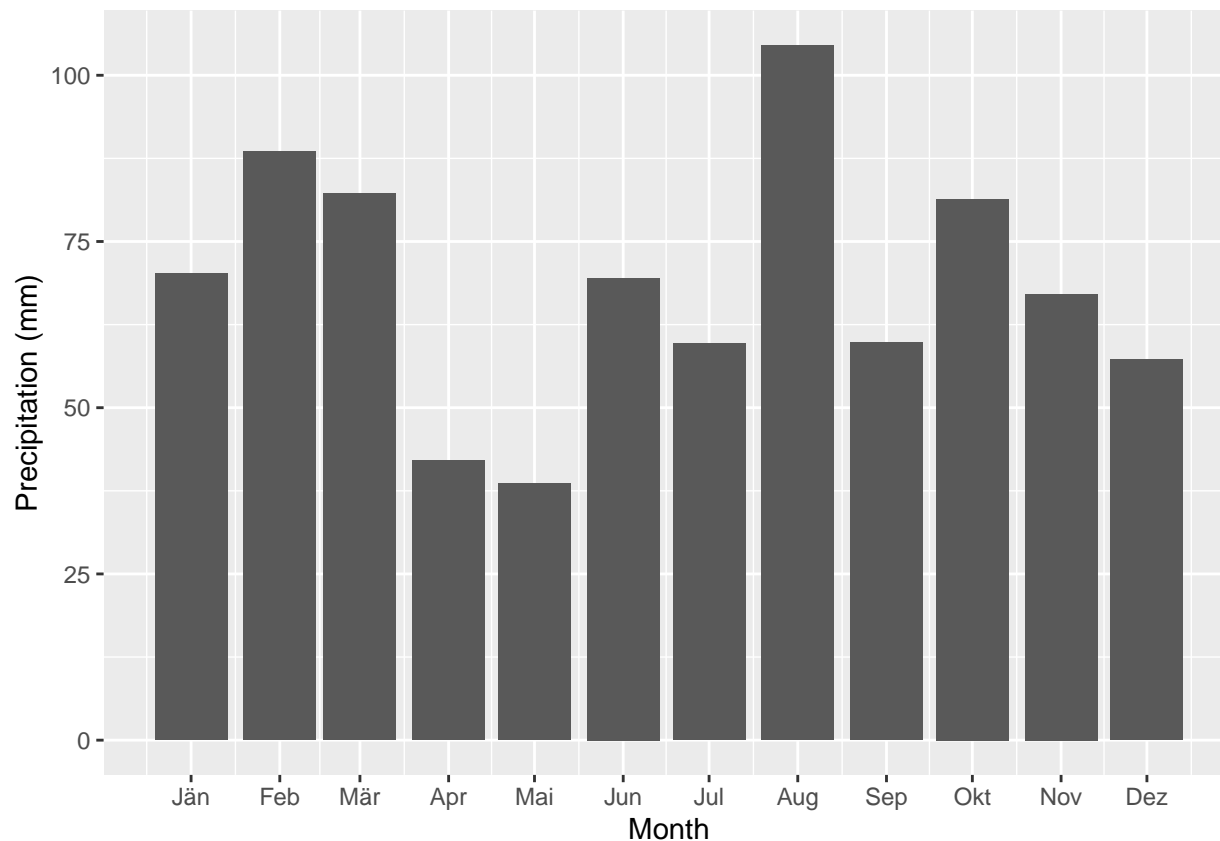
```
weather_summary_annual %>% ggplot(aes(x = ymd(paste(2025, month, 1, sep = "-")))) +
  geom_line(aes(y = low_temp, color = "low")) +
  geom_line(aes(y = mean_temp, color = "mean")) +
  geom_line(aes(y = high_temp, color = "high")) +
  scale_x_date(date_breaks = "months" , date_labels = "%b") +
  labs(x = "Month", y = "Temperature (C°)", color = "Temperature")
```



Annual weather precipitation plot

For the precipitation plot a bar chart is chosen. It is visibly more intuitive, because with precipitation one likes to compare months and not only analyze increases and decreases.

```
weather_summary_annual %>% ggplot(aes(x = ymd(paste(2025, month, 1, sep = "-")), y = precipitation)) +
  geom_col() +
  scale_x_date(date_breaks = "months" , date_labels = "%b") +
  labs(x = "Month", y = "Precipitation (mm)",)
```



Conclusion

This assignment shows how publicly available web pages can be used to retrieve valuable data. It is quite straightforward to process the data using `rvest` and `dplyr` methods. When the data table is at hand, creating plots with `ggplot2` is also a transparent process.

With this said, I feel like it is risky to employ such scripts in an automated manner. Many HTML tables are meant for visualization and not scraping. For some tables I have noticed that the number of columns change depending on the city. For example, precipitation is a field which can be split up into rain, snow, and potentially more. There is no specification on when and how this splitting happens on the website.

Furthermore, the default language of the client (device) used, changes the parsed content. It is important to keep this in mind and make as little assumptions as possible to keep the code robust.