The short-term association between environmental variables and mortality: evidence from Europe

Tutorial session

Jens Robben, Katrien Antonio and Torsten Kleinow

Applied Statistics Workshop ISBA (UCLouvain) | April 18, 2025

Prologue

Introduction

Workshop

https://jensrobben.github.io/Workshop-LLN/

The GitHub page where slides, R code and data sets are available.

Authors of the paper

Jens Robben ✉

Katrien Antonio ✉

Torsten Kleinow ✉

Goals of the tutorial session

Goals of the tutorial session

In this tutorial session you will learn how to:

- download and visualize the NUTS 2 regions in Belgium
- download and illustrate weekly mortality and weather data for Belgium
- calibrate a weekly mortality baseline model
- use a machine learning model to associate deviations from the mortality baseline with weather data
- construct in-sample mortality predictions.

The tutorial is a simplified version of our paper

Outline of the workshop:

- NUTS 2 regions in Belgium
- Download and import mortality data
- Download and import weather data
- Weekly mortality baseline model
- Modelling deviations from the mortality baseline
- Results

The full working paper is available on arXiv: https://arxiv.org/abs/2405.18020.

Corresponding R code is available on GitHub: https://github.com/jensrobben/EnvVar-Mortality-Europe.

NUTS 2 regions in Belgium

NUTS 2 regions in Belgium

We focus on mortality data in NUTS 2 regions in Belgium. NUTS 2 is the second most detailed level in the Nomenclature of Territorial Units for Statistics (see here).

We read the NUTS level shapefile using the `read_sf` function from the {sf} package. This shapefile contains geospatial data for all NUTS regions in Europe. We have pre-downloaded it from Eurostat and stored it in our GitHub repository.

```
shapef <- read_sf('../data/shapefile/NUTS_RG_01M_2021_4326.shp')
```

We then filter the shapefile to include only NUTS 2 level regions (`LEVL_CODE == 2`) in Belgium (`CNTR_CODE == 'BE'`) using the `filter` function from the {dplyr} package, and arrange the data by NUTS 2 code.

```
shapef <- shapef %>%
  dplyr::filter(LEVL_CODE == 2, CNTR_CODE == "BE") %>%
  arrange(NUTS_ID)
```

We extract the `NAME_LATN` attribute from the shapefile and find that the Belgian NUTS 2 regions correspond to the 10 provinces in Belgium and Brussels.

```
tail(shapef$NAME_LATN)
## [1] "Prov. West-Vlaanderen" "Prov. Brabant Wallon"  "Prov. Hainaut"
## [4] "Prov. Liège"           "Prov. Luxembourg (BE)" "Prov. Namur"
```

The `geometry` attribute stores the geographic data for the Belgian provinces (+ Brussels). Let's examine the geometry of the first region in the shapefile:

```
shapef[1,]$geometry
## Geometry set for 1 feature
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: 4.244969 ymin: 50.76425 xmax: 4.480477 ymax: 50.9136
## Geodetic CRS:  WGS 84
```

- The geometry type `MULTIPOLYGON` indicates that the shapefile is made up of multiple polygons outlining the boundaries of the Belgian provinces.
- The dimension `XY` indicates that the polygons are defined in two dimensions (longitude and latitude).
- The bounding box (`xmin`,`ymin`, `xmax`, `ymax`) reflects the rectangular area of (long, lat) coordinates encompassing the region.
- The geodetic CRS `WGS 84` is the coordinate reference system (CRS) used for this shapefile.

We can visualize the Belgian NUTS 2 regions (provinces) using `ggplot`. The key function for plotting geographical data is the layer `geom_sf` from the {ggplot2} package. This function uses the geometry column from our shapefile. The `fill = NUTS_ID` aesthetic fills each NUTS 2 region with a different colour.

```
ggplot(shapef) +
  geom_sf(aes(fill = NUTS_ID, geometry = geometry))
```

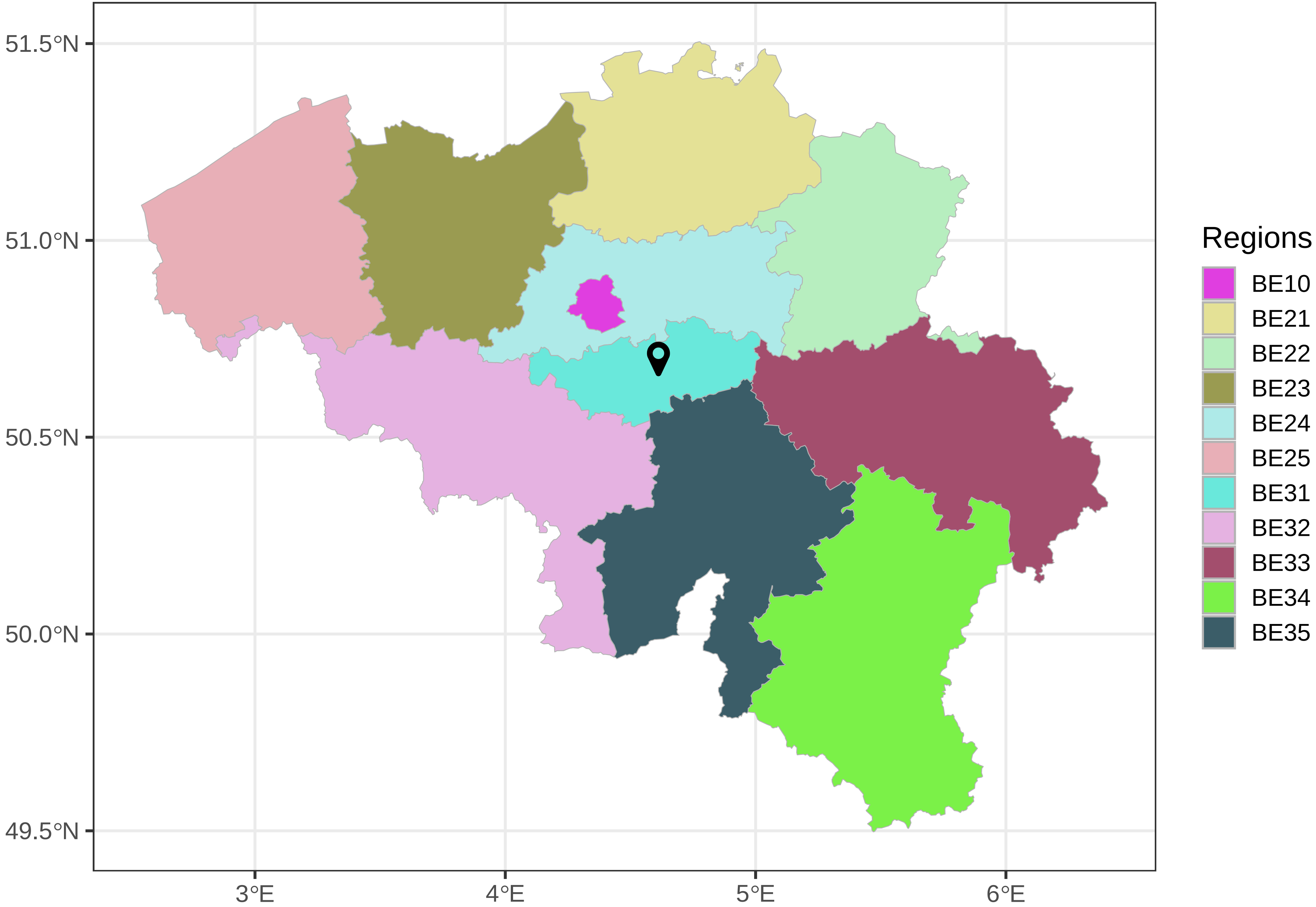
The above code provides a basic plot. We can enhance it by specifying a custom color palette.

```
colours ← distinctColorPalette(k = 11)

load.fontawesome()
location ← fontawesome(c('fa-map-marker'))

ggplot(shapef) +
  geom_sf(aes(fill = NUTS_ID,
              geometry = geometry), col = 'gray70') +
  geom_text(aes(x = 4.6118, y = 50.6666, label = location),
            family = 'fontawesome-webfont', vjust = 0.05,
            data = shapef[1,], size = 7) +
  scale_fill_manual(name = 'Regions', values = colours) +
  theme_bw(base_size = 15)  + xlab('') + ylab('') +
  ggtitle("NUTS 2 regions (provinces) Belgium")
```

NUTS 2 regions (provinces) Belgium



Download and import mortality data

Download and import mortality data

We do a live download using the function `get_eurostat` from the {eurostat} package to retrieve Belgian death counts by week (from 2000 on), sex, 5-year age group and NUTS 2 region from Eurostat.

```
# Download weekly death counts from Eurostat (approx. 1 min)
dxtw.raw ← get_eurostat(id = 'demo_r_mwk2_05', time_format = 'raw',
                        cache = FALSE, filters = list(geo = shapef$NUTS_ID))
```

- `id` refers to the unique identifier of the data set we download (see here)
- `time_format = 'raw'` specifies we want to preserve the original date object
- `cache = FALSE` avoids caching and speeds up the running process
- `filters` limits the data to the NUTS 2 Belgian regions.

If you encounter issues with the live download, we have pre-downloaded this weekly mortality data set, and made it available in the GitHub repository. You can load this R object as follows:

```
dxtw.raw ← readRDS('../data/eurostat/dxtw_BE_NUTS2.rds')
```

We set appropriate column names using the `colnames` function:

```
colnames(dxtw.raw) ← c('Freq', 'Age', 'Sex', 'Unit',  'Region', 'Time', 'Deaths')
```

We visualize an extract of the downloaded Eurostat data:

```
head(dxtw.raw)
```

Freq	Age	Sex	Unit	Region	Time	Deaths
W	TOTAL	T	NR	BE10	2000-W01	298
W	TOTAL	T	NR	BE10	2000-W02	260
W	TOTAL	T	NR	BE10	2000-W03	281
W	TOTAL	T	NR	BE10	2000-W04	242
W	TOTAL	T	NR	BE10	2000-W05	245
W	TOTAL	T	NR	BE10	2000-W06	197

```r
dxtw.BE <- dxtw.raw %>%
  dplyr::filter(Age == 'TOTAL', Sex == 'T') %>%
  mutate(ISOYear = as.integer(substr(Time, 1, 4)),
         ISOWeek = as.integer(substr(Time, 7, 8)))
```

We filter the mortality data set `dxtw.raw` on the total age class (`Age == 'TOTAL'`) and unisex data (`Sex == 'T'`) using the `filter` function from the {dplyr} package. Using the function `mutate`, we create two new columns, `ISOYear` and `ISOWeek`. Hereto we extract the year (digits 1-4) and week (digits 7-8) from the `Time` column in `dxtw.raw` using the function `substr`.

```r
dxtw.BE <- dxtw.BE %>%
  dplyr::filter(ISOYear <= 2019) %>%
  select(c('ISOYear', 'ISOWeek', 'Region', 'Deaths'))
```

To exclude the COVID-19 pandemic, we filter the data up to ISO-year 2019. We retain only the relevant columns: `ISO-year`, `ISO-week`, `Region`, and `Deaths` using the `select` function from {dplyr}.

```r
# Add date
wdate   <- ISOweek2date(sprintf("%d-W%02d-%d",
                                dxtw.BE$ISOYear,
                                dxtw.BE$ISOWeek, 1))

dxtw.BE <- dxtw.BE %>%
  mutate('Date' = wdate)
```

Next, we use the `ISOweek2date` function from the {ISOweek} package to create a new column `Date`, representing the start date of each ISO-week.

This concludes the construction of a data set with weekly deaths for Belgium, stored in `dxtw.BE`. On to the exposures!

Your turn

Q: Visualize the evolution of the weekly death counts in the NUTS 2 region that includes Louvain-la-Neuve.

```r
# Select region encompassing Louvain-la-Neuve
region <- ...

# Filter the data for the specified region
dxtw.BW <- ... %>% filter( ... )

# Create the plot
ggplot( ... ) +
  geom_line(aes(x = ... , y = ... ), colour = RCLRbg) +
  xlab('Date') + ylab('Death counts') +
  theme_bw(base_size = 15)
```

Your turn

A: Louvain-la-Neuve is located in the province Brabant Wallon. The NUTS 2 code for this province is BE31, which can be found in the `shapef` object.

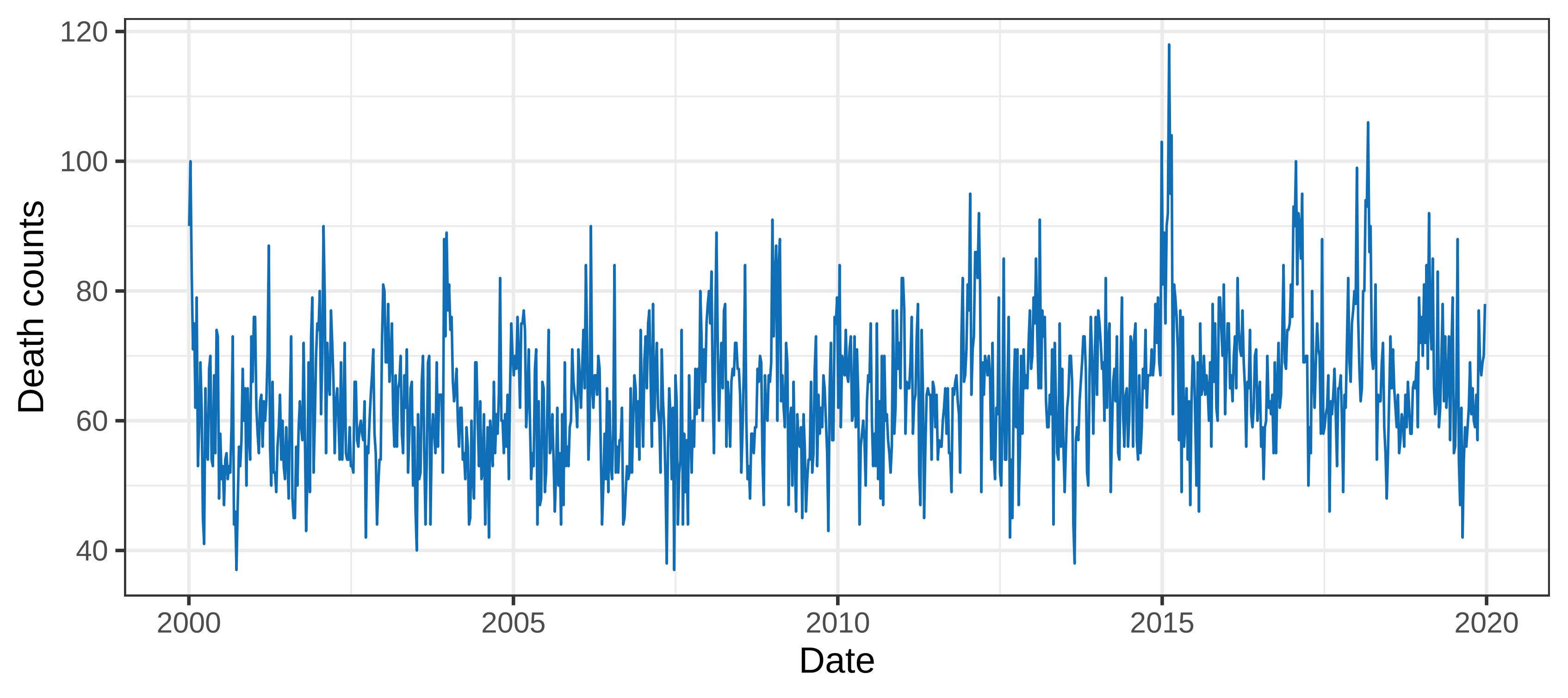
```
# Select region encompassing Brabant Wallon
region <- 'BE31'

# Filter the data for the specified region
dxtw.BW <- dxtw.BE %>% filter(Region == region)

# Create the plot
ggplot(dxtw.BW) +
  geom_line(aes(x = Date, y = Deaths), colour = RCLRbg) +
  xlab('Date') + ylab('Death counts') +
  theme_bw(base_size = 15)
```

Your turn

A: Louvain-la-Neuve is located in the province Brabant Wallon. The NUTS 2 code for this province is BE31, which can be found in the `shapef` object.



Exposures

```r
Pxt.raw ← get_eurostat(id = 'demo_r_d2jan',
                        time_format = 'raw',
                        cache = FALSE,
                        filters =
                          list(geo =
                                 shapef$NUTS_ID))

colnames(Pxt.raw) ← c('Freq', 'Unit', 'Sex', 'Age',
                       'Region', 'Time', 'Pop')
```

```r
Pxt.raw ←
  readRDS('../data/eurostat/Pxt_BE_NUTS2.rds')
```

We perform a live download (again) using the `get_eurostat` function to retrieve population counts on January 1st, categorized by age, sex, and NUTS 2 region. The function's parameters are similar to those used for downloading weekly death counts, with the main difference being the identifier: `demo_r_d2jan` (see here).

We specify the column names using the function `colnames`.

If you encounter issues with the live download, we have pre-downloaded the population count data set. You can load this R object using the `readRDS` function.

We display a sample of the data set `Pxt.raw`, which contains the age-specific population counts for the Belgian provinces:

```
head(Pxt.raw)
```

Freq	Unit	Sex	Age	Region	Time	Pop
A	NR	T	TOTAL	BE10	1990	964385
A	NR	T	TOTAL	BE10	1991	960324
A	NR	T	TOTAL	BE10	1992	951217
A	NR	T	TOTAL	BE10	1993	950339
A	NR	T	TOTAL	BE10	1994	949070
A	NR	T	TOTAL	BE10	1995	951580

```r
Pxt.BE <- Pxt.raw %>%
  dplyr::filter(Age == 'TOTAL',
                Sex == 'T') %>%
  mutate(Time = as.integer(
    as.character(Time)))
```

We extract data from the `Pxt.raw` data set related to the total age group (`Age == 'TOTAL'`) and unisex population (`Sex == 'T'`) using the `filter` function from {dplyr}. Additionally, we convert the `Time` column from its original factor to an integer type.

```r
Extw.BE <- Pxt.BE %>% group_by(Region) %>%
  reframe(Time, 'Expo' =
            c((Pop[-length(Time)] + Pop[-1])/2,
              Pop[length(Time)])/52.18) %>%
  dplyr::filter(Time <= 2019, Time >= 2000)
```

We define the annual exposure in each year t, region r as a mid-year population estimate and assume a constant weekly exposure within that year:

$$E_t^{(r)} = \frac{P_t^{(r)} + P_{t+1}^{(r)}}{2}$$

$$E_{t,w}^{(r)} = \frac{E_t^{(r)}}{52.18},$$

with 52.18 the average number of weeks in a year.

```r
Df <- dxtw.BE %>%
  left_join(Extw.BE, by = c('ISOYear' =  'Time',
                            'Region' = 'Region'))
```

We merge the death counts `dxtw.BE` with the exposure estimates `Extw.BE` based on the year and NUTS 2 region using the `left_join` function from {dplyr}.

Your turn

Q: Calculate the weekly mortality rates $q_{t,w}^{(r)}$ using the exposure estimates for the province Antwerp. Visualize these as a function of time.

Your turn

A: We first filter the mortality data set `Df` (with regional, weekly death counts and exposures) to focus only on data from the province of Antwerp (BE21). Then, we compute the mortality rates $q_{t,w}^{(r)}$ as $1 - \exp(-d_{t,w}^{(r)}/E_{t,w}^{(r)})$. Lastly, we visualize the results using {ggplot}.

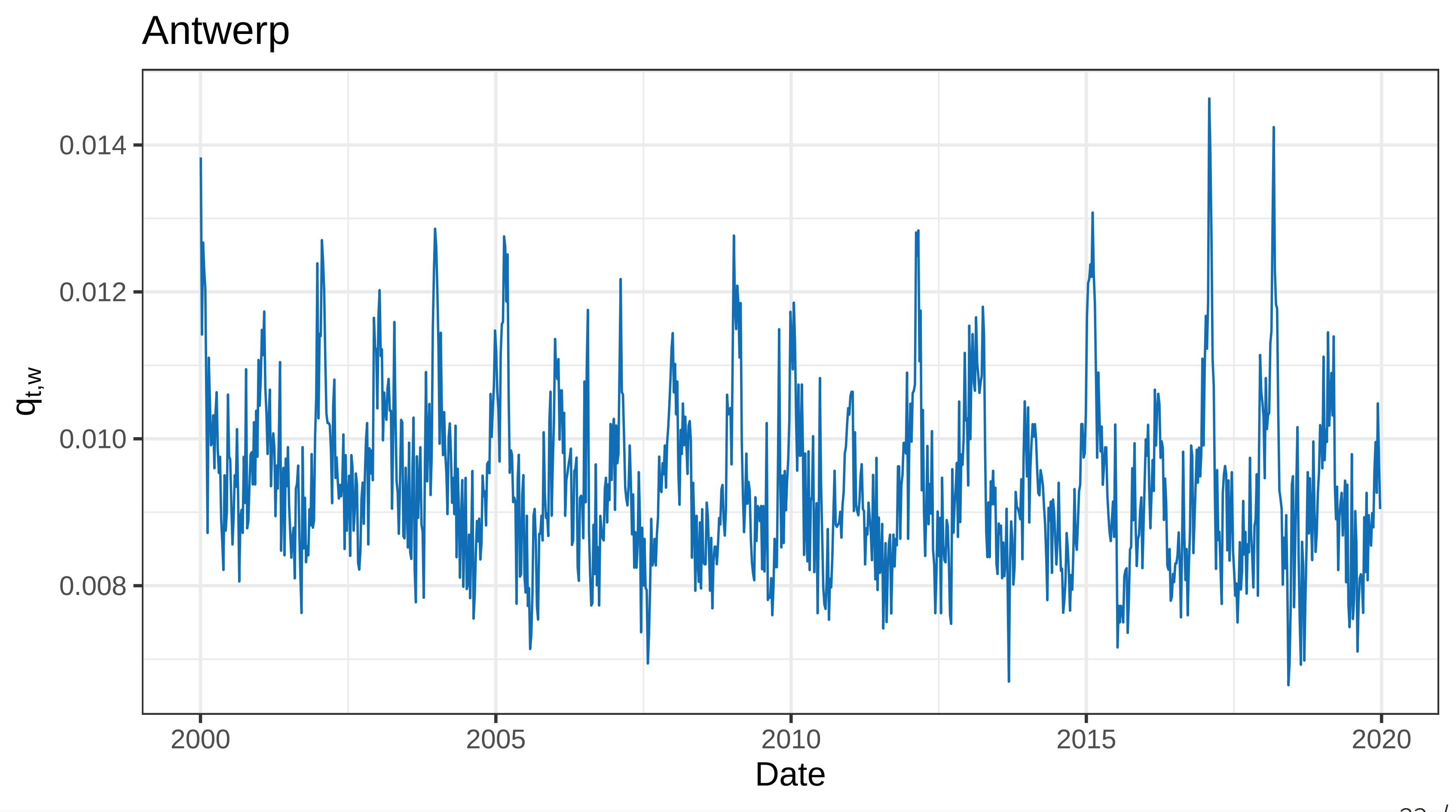
```r
# Filter on province Antwerp (BE21)
Df.ANT     <- Df %>% dplyr::filter(Region == 'BE21')

# Construct mortality rates
Df.ANT$qtw <- 1 - exp(-Df.ANT$Deaths/Df.ANT$Expo)

# Plot
ggplot(Df.ANT) +
  geom_line(aes(x = Date, y = qtw), col = RCLRbg) +
  ylab(bquote(q['t,w'])) +
  ggtitle('Antwerp') +
  theme_bw(base_size = 15)
```

Your turn

A: We first filter the mortality data set `Df` (with regional, weekly death counts and exposures) to focus only on data from the province of Antwerp (BE21). Then, we compute the mortality rates $q_{t,w}^{(r)}$ as $1 - \exp(-d_{t,w}^{(r)}/E_{t,w}^{(r)})$. Lastly, we visualize the results using {ggplot}.



Download and import weather data

Download and import weather data

In the R script eobs_data.R, we do a live download from the Copernicus Climate Data Store to retrieve daily weather factors throughout the years 2000-2024 (WARNING! - high computational time).

These weather factors include the daily levels of the maximum temperature (`tx`), average temperature (`tg`), minimum temperature (`tn`), relative humidity (`hu`), total precipitation amount (`rr`), and average wind speed (`fg`). The resulting data sets are stored for you in the GitHub repository. Let's read them into our environment:

```r
tn_NUTS2_daily <- readRDS('../data/eobs/BE_NUTS2_tn_daily.rds')
tg_NUTS2_daily <- readRDS('../data/eobs/BE_NUTS2_tg_daily.rds')
tx_NUTS2_daily <- readRDS('../data/eobs/BE_NUTS2_tx_daily.rds')
fg_NUTS2_daily <- readRDS('../data/eobs/BE_NUTS2_fg_daily.rds')
hu_NUTS2_daily <- readRDS('../data/eobs/BE_NUTS2_hu_daily.rds')
rr_NUTS2_daily <- readRDS('../data/eobs/BE_NUTS2_rr_daily.rds')
```

We examine the structure of one of these datasets:

```r
head(tx_NUTS2_daily)
```

Date	ISOYear	ISOWeek	Region	tx
2000-01-01	1999	52	BE10	8.010
2000-01-02	1999	52	BE10	9.050
2000-01-03	2000	1	BE10	8.710
2000-01-04	2000	1	BE10	9.380
2000-01-05	2000	1	BE10	8.445
2000-01-06	2000	1	BE10	10.785

Your turn

Q: Illustrate the evolution of the weather factors in the province Brabant Wallon as a function of time using {ggplot}.

Your turn

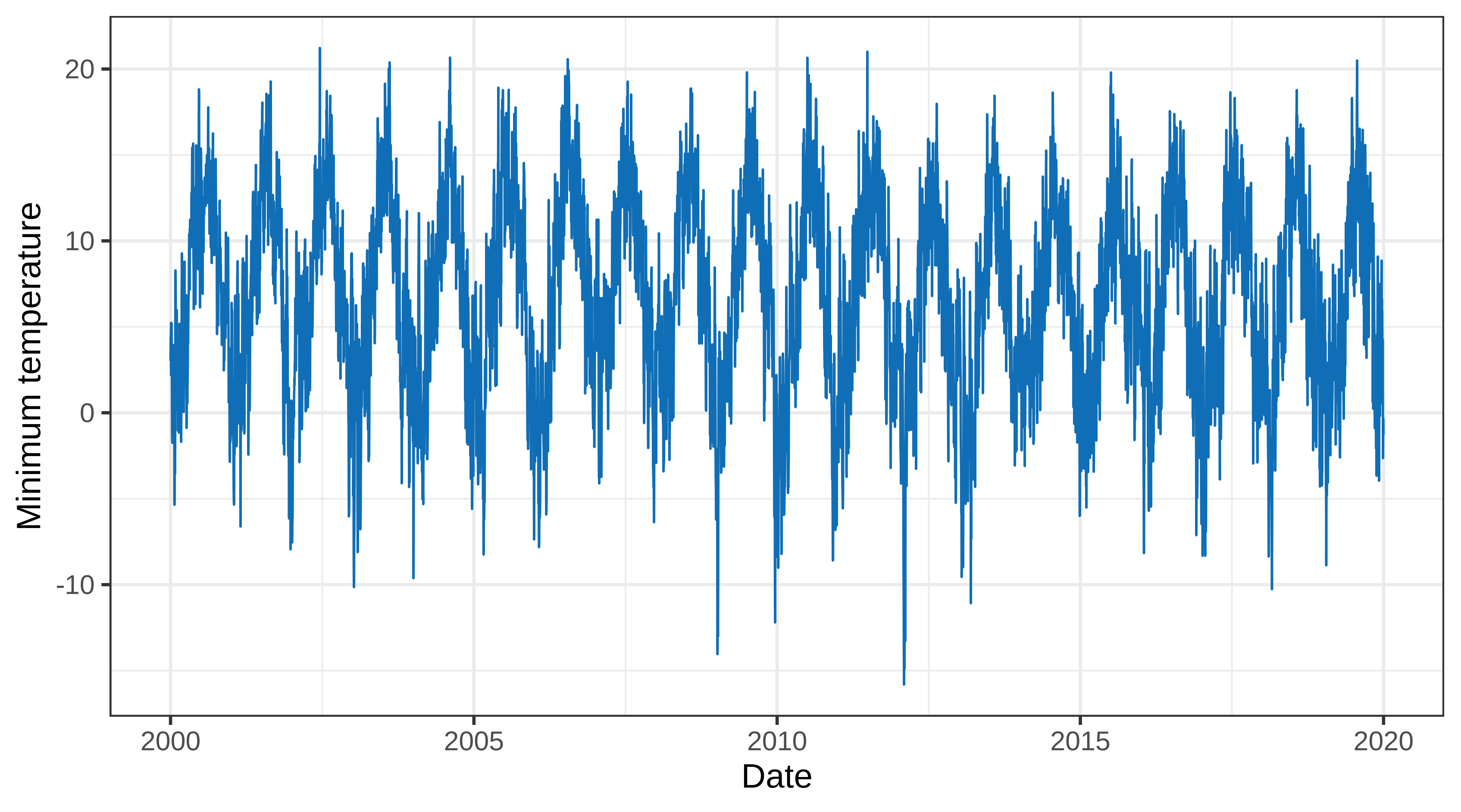
A: We provide the {ggplot} instructions to visualize the evolution of the minimum temperature in the NUTS 2 region of Brabant Wallon (BE31). We start by selecting the relevant region (BE31). Then, we filter the daily minimum temperature data set `tn_NUTS2_daily` for this region. Finally, we create the plot.

```r
region <- 'BE31' # Brabant Wallon
tn_BW_daily <- tn_NUTS2_daily %>%
  dplyr::filter(Region == region)

ggplot(tn_BW_daily) +
  geom_line(aes(x = Date, y = tn), colour = RCLRbg) +
  xlab('Date') + ylab('Minimum temperature') +
  theme_bw(base_size = 15)
```

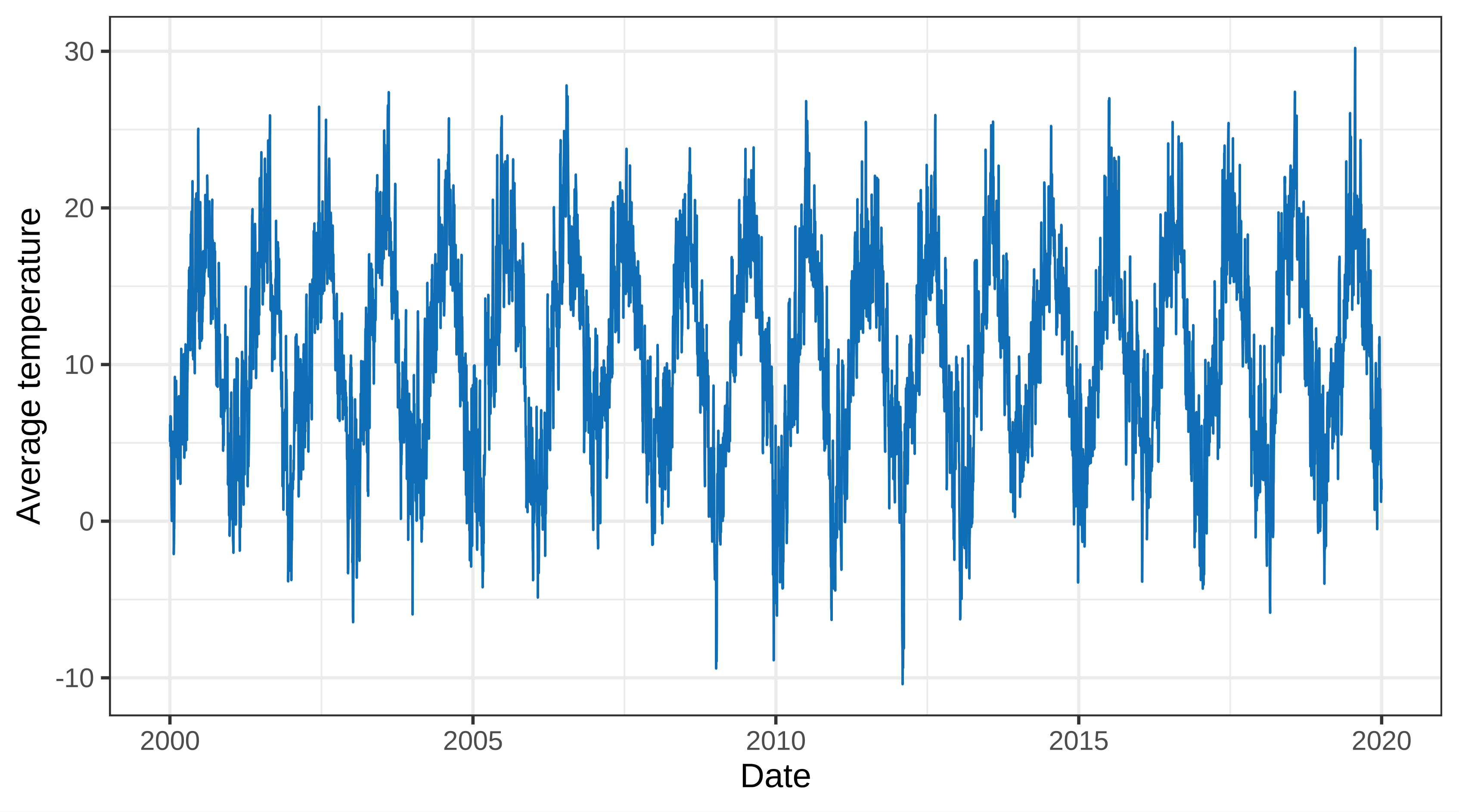
Your turn

A: Minimum temperature.



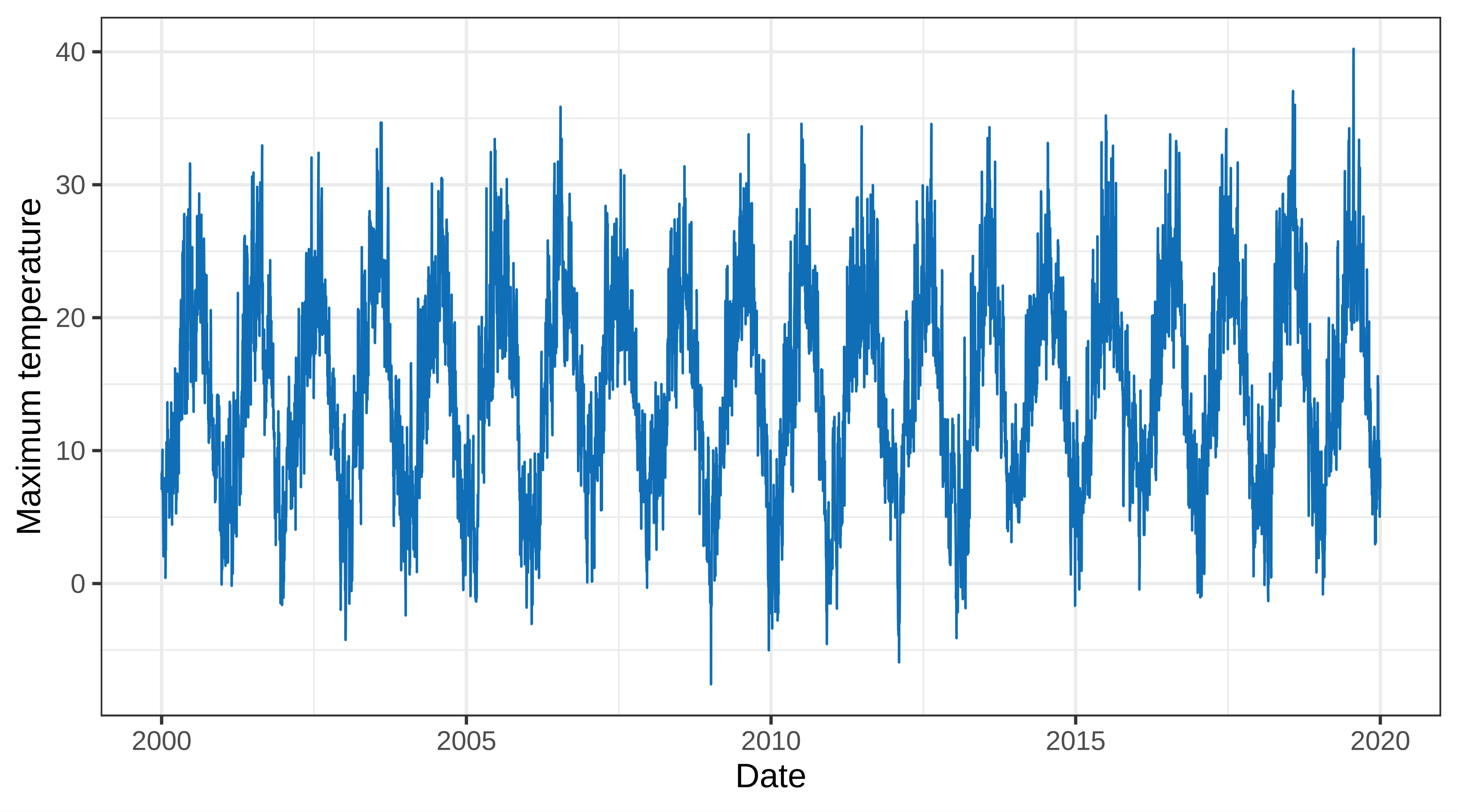
Your turn

A: Average temperature.



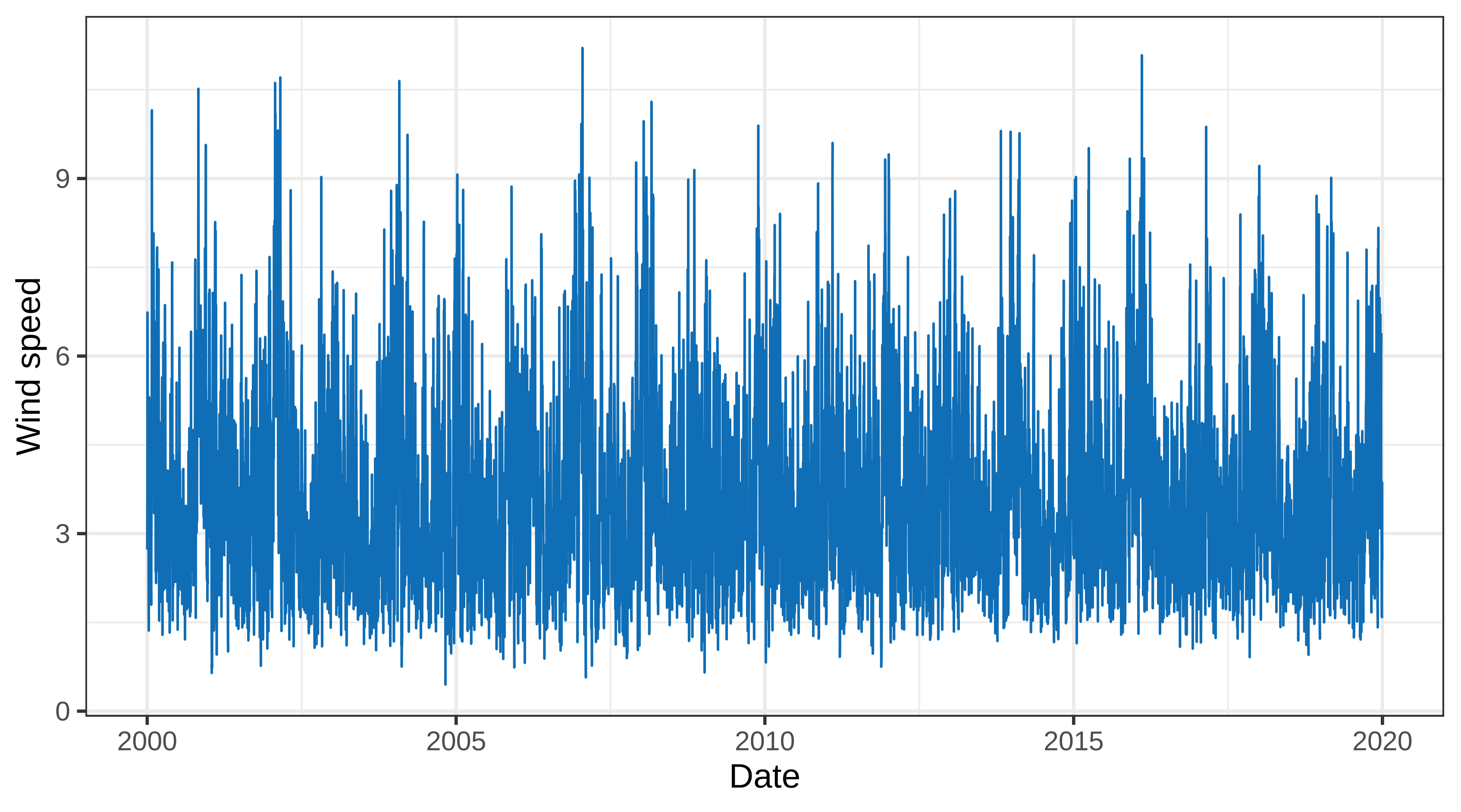
Your turn

A: Maximum temperature.



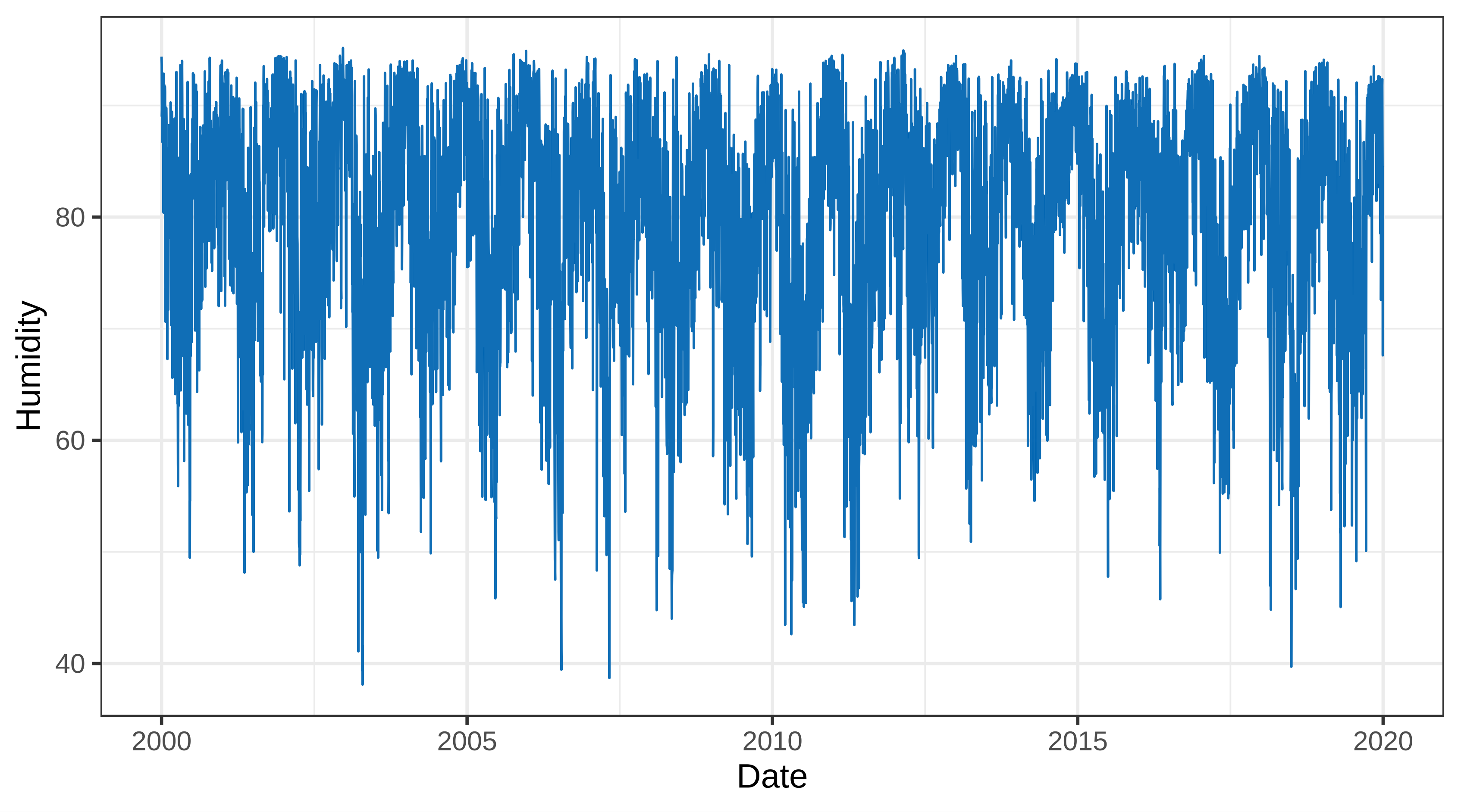
Your turn

A: Wind speed.



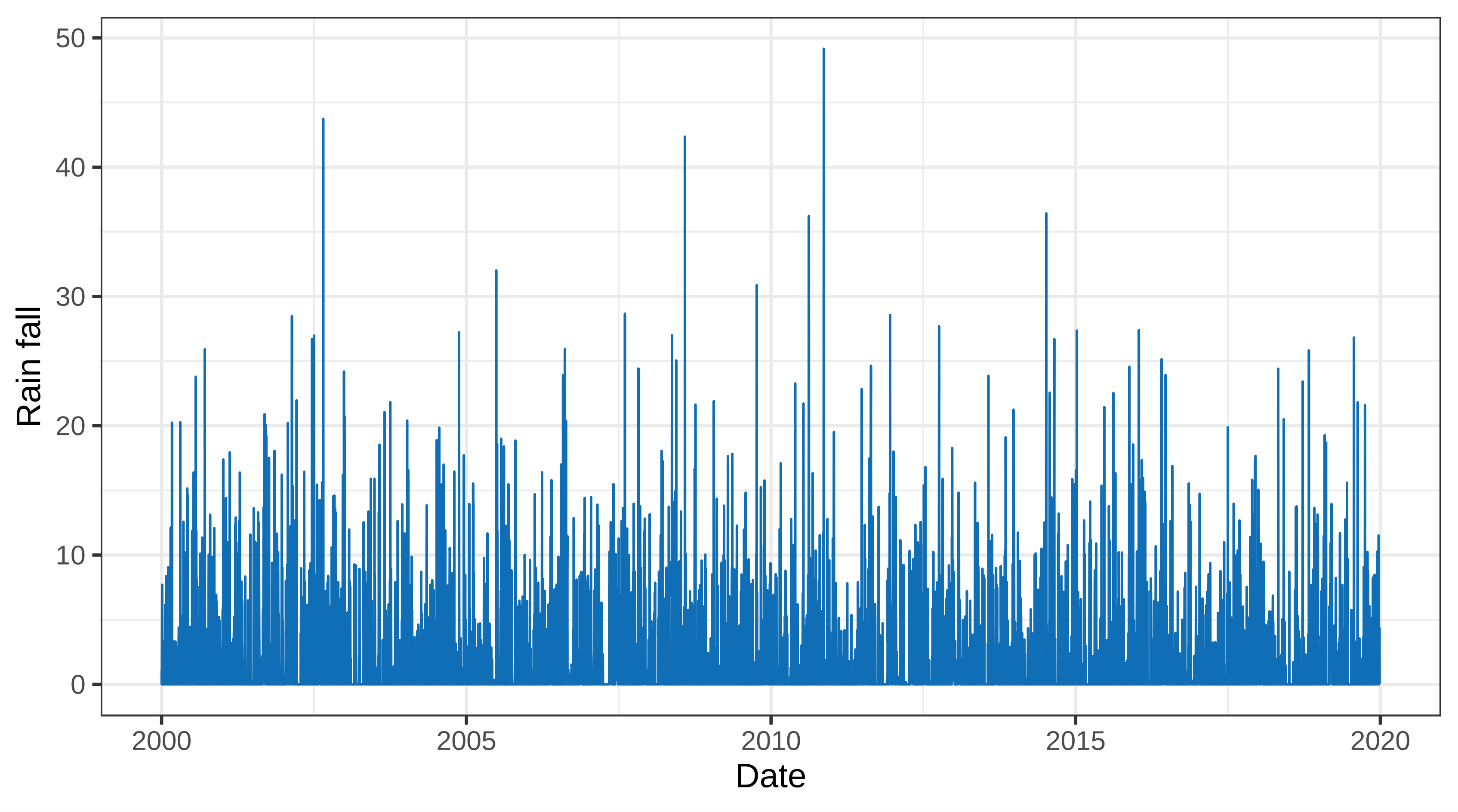
Your turn

A: Humidity.



Your turn

A: Rain fall.



From daily to weekly weather data

```
Df.weather ←
  plyr::join_all(list(tn_NUTS2_daily, tg_NUTS2_daily,
                      tx_NUTS2_daily, fg_NUTS2_daily,
                      hu_NUTS2_daily, rr_NUTS2_daily),
                 by = c('Date', 'ISOYear', 'ISOWeek',
                        'Region'))
```

We merge all the daily weather data sets (`tn_NUTS3_daily`, `tg_NUTS3_daily`, `tx_NUTS3_daily`, `fg_NUTS3_daily`, `hu_NUTS3_daily`, `rr_NUTS3_daily`) into one large data frame `Df.weather` using the `join_all` function from the {plyr} package. The merging is done based on common columns: `Date`, `ISOYear`, `ISOWeek`, and `Region`.

```
Df.weather ← Df.weather %>%
  group_by(ISOYear, ISOWeek, Region) %>%
  reframe('tn' = mean(tn, na.rm = TRUE),
          'tg' = mean(tg, na.rm = TRUE),
          'tx' = mean(tx, na.rm = TRUE),
          'hu' = mean(hu, na.rm = TRUE),
          'fg' = mean(fg, na.rm = TRUE),
          'rr' = mean(rr, na.rm = TRUE))
```

To align with the weekly time scale of the mortality statistics, we convert the daily weather data in `Df.weather` to weekly data. This is done by grouping the data by `ISOYear`, `ISOWeek`, and `Region`, and then calculating the weekly average for each weather variable (`tn`, `tg`, `tx`, `hu`, `fg`, `rr`). We exclude any missing daily values in the averaging process using the argument `na.rm = TRUE`.

```
Df ← Df %>% left_join(Df.weather)
```

We join the mortality data set `Df` with the weather data set `Df.weather` based on common columns (default).

Weekly mortality baseline model

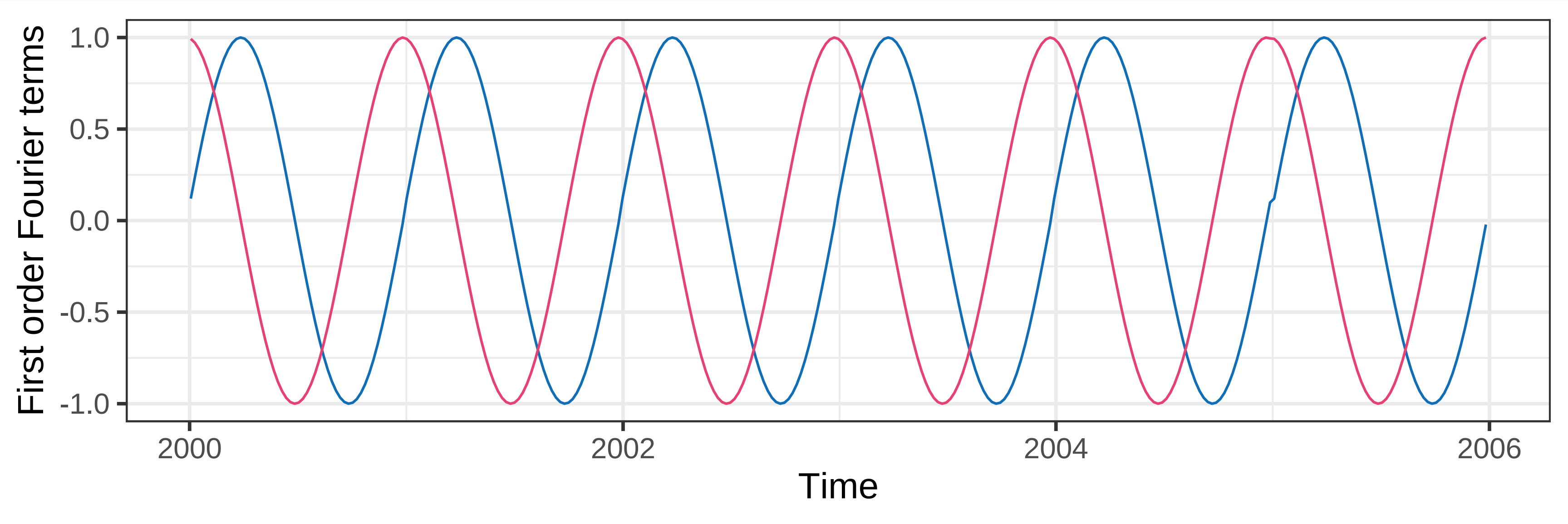
Model specifications

To model the weekly mortality baseline trend, we account for the seasonality observed in the region-specific weekly death counts by means of sine-cosine Fourier terms, defined as:

$$\left[\sin\left(2\pi\frac{n}{T}x\right),\cos\left(2\pi\frac{n}{T}x\right)\right]_{n\in\mathbb{N}},$$

with n the frequency or number of cycles and T the period of the sinusoidal terms.

We visualize the Fourier terms with a frequency of $n = 1$ and a period of 52.18 (annual cycle).



We use the Serfling model specification to construct a baseline mortality model for the weekly death counts $D_{t,w}^{(r)}$ in a specific region r. This model assumes that the number of deaths random variable follows a Poisson distribution, with as mean the weekly exposure $E_{t,w}^{(r)}$ times the weekly force of mortality $\mu_{t,w}^{(r)}$:

$$D_{t,w}^{(r)} \sim \text{Poisson}\left(E_{t,w}^{(r)}\,\mu_{t,w}^{(r)}\right).$$

Hereby:

$$\log \mu_{t,w}^{(r)} = \beta_0^{(r)} + \beta_1^{(r)} t + \beta_2^{(r)} \sin\left(\frac{2\pi w}{52.18}\right) + \beta_3^{(r)} \cos\left(\frac{2\pi w}{52.18}\right) + \quad \beta_4^{(r)} \sin\left(\frac{2\pi w}{26.09}\right) + \beta_5^{(r)} \cos\left(\frac{2\pi w}{26.09}\right),$$

which includes an intercept, an annual time trend, and Fourier-terms to capture both annual and semi-annual frequencies.

You can determine the number of seasonal terms in each region using AIC or BIC (not covered in this tutorial).

We compute these Fourier terms and add them to our mortality dataset:

```
Df ← Df %>%
  mutate('fsin52' = sin(2*pi*ISOWeek/52.1775),
         'fcos52' = cos(2*pi*ISOWeek/52.1775),
         'fsin26' = sin(4*pi*ISOWeek/52.1775),
         'fcos26' = cos(4*pi*ISOWeek/52.1775))
```

Model calibration

```r
formula <- Deaths ~ ISOYear + fsin52 +
  fcos52 + fsin26 + fcos26
```

We construct the formula specification with the response variable `Deaths` on the one hand and the covariates `ISOYear` and the Fourier terms `fsin52`, `fcos52`, `fsin26`, and `fcos26` on the other hand.

```r
for(r in shapef$NUTS_ID){
  Dfr <- Df %>%
    dplyr::filter(Region == r)

  fit.r <- glm(formula, data = Dfr,
               offset = log(Dfr$Expo),
               family = poisson(link = 'log'))

  Df[which(Df$Region == r),'bDeaths'] <-
    fit.r$fitted.values
}
```

For each Belgian province r, we filter the mortality data set to include only data for that region, saving it as `Dfr`.

We then fit a Generalized Linear Model (GLM) on `Dfr` with as `offset` the logarithm of the exposure variable, i.e., `log(Dfr$Expo)`. We specify the Poisson distribution with a log-link using the `family` argument.

Finally, we add the estimated deaths to the mortality data set `Df` in a new column `bDeaths`.

Your turn

Q: Calculate and illustrate the estimated baseline and observed weekly mortality rates for Brussels and Antwerp.

Your turn

A: We start with Brussels (BE10). First, we filter the mortality data set `Df` on data for Brussels. Then we compute the observed and estimated weekly mortality rates and plot the results.

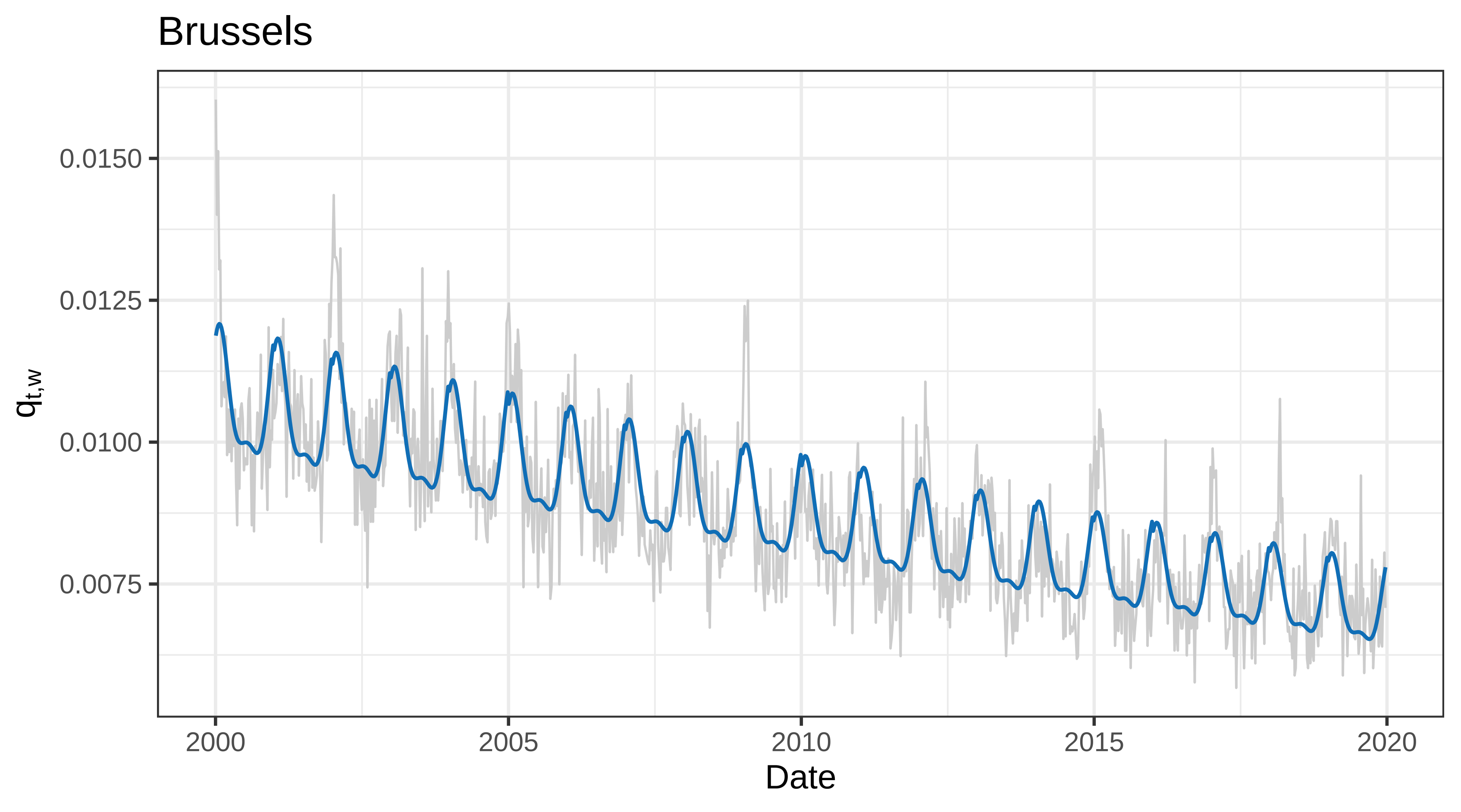
```r
# Filter on Brussels (BE10)
Df.BR <- Df %>% dplyr::filter(Region == 'BE10')

# Observed and estimated mortality rates
Df.BR$qtw.obs <- 1 - exp(-Df.BR$Deaths/Df.BR$Expo)
Df.BR$qtw.est <- 1 - exp(-Df.BR$bDeaths/Df.BR$Expo)

# Plot
ggplot(Df.BR) +
  geom_line(aes(x = Date, y = qtw.obs), col = 'gray80') +
  geom_line(aes(x = Date, y = qtw.est), col = RCLRbg,
            linewidth = 0.8) +
  ylab(bquote(q['t,w'])) +
  ggtitle('Brussels') +
  theme_bw(base_size = 15)
```

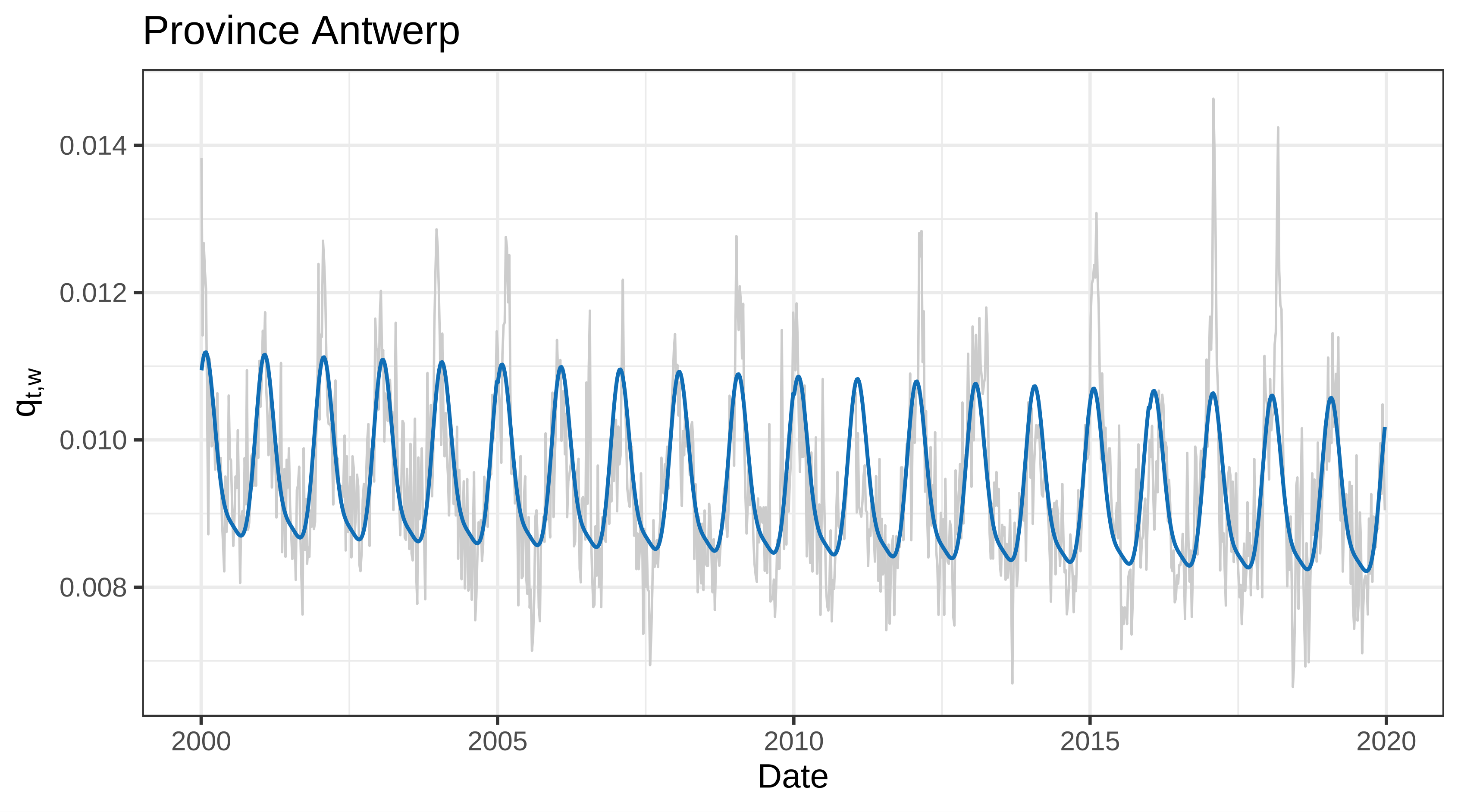
Your turn

A: We start with Brussels (BE10). First, we filter the mortality data set `Df` on data for Brussels. Then we compute the observed and estimated weekly mortality rates and plot the results.



Your turn

A: We repeat the procedure for Antwerp (BE21).



Modelling deviations from the mortality baseline

Model specifications

We associate deviations from the weekly mortality baseline model with region-specific weather features. We work with the following model assumptions:

$$\begin{aligned} D_{t,w}^{(r)} &\sim \text{Poisson}\left(\hat{b}_{t,w}^{(r)}\,\phi_{t,w}^{(r)}\right) \\ \phi_{t,w}^{(r)} &= f\left(\boldsymbol{c}_{t,w}^{(r)},\ l^1\left(\boldsymbol{c}_{t,w}^{(r)}\right),\ l^2\left(\boldsymbol{c}_{t,w}^{(r)}\right),\ldots,\ l^s\left(\boldsymbol{c}_{t,w}^{(r)}\right)\right). \end{aligned}$$

Here:

- $D_{t,w}^{(r)}$ represents the observed deaths in region $\boldsymbol{r}$ during week $\boldsymbol{w}$ of year $\boldsymbol{t}$
- $\hat{b}_{t,w}^{(r)}$ denotes the estimated baseline deaths based on the Serfling model in region $\boldsymbol{r}$, week $\boldsymbol{w}$ and year $\boldsymbol{t}$
- $\boldsymbol{c}_{t,w}^{(r)}$ the set of weather features
- $l^j(\boldsymbol{c}_{t,w}^{(r)})$ denotes the set of weather features, lagged by $\boldsymbol{j}$ weeks
- $f(\cdot)$ is a machine learning model that can possibly capture non-linear associations between target and inputs, as well as interaction effects between the input features. In this tutorial we work with an XGBoost model.

Feature construction

```r
vars <- c('tn', 'tg', 'tx',
          'hu', 'fg', 'rr')
```

We select the weather features of interest and save them in the variable `vars`.

```r
list.lagdf <- list()
for (v in vars){
  list.lagdf[[which(vars == v)]] <- Df %>%
    group_by(Region) %>%
    reframe(Date,
            !!paste0(v,'_l1') := lag(!!sym(v), 1),
            !!paste0(v,'_l2') := lag(!!sym(v), 2))
}

df.lag <-
  plyr::join_all(list.lagdf,
                 by = c('Region', 'Date'))
```

For each feature in `vars`, we group the data frame `Df` (containing mortality and weather data) by `Region`. We then use the `reframe` function from {dplyr} to create new columns with one-week and two-week lagged values for each weather feature by applying the `lag` function from {dplyr}. These lagged data frames are stored in the list `list.lagdf`. Lastly, we merge all the lagged data frames into a single data frame `df.lag` using the `join_all` function from {plyr}. Note the use of `!!` (injection operator) to force an early evaluation of the `paste0` object (check `help("!!")`).

```r
Df <- Df %>%
  left_join(df.lag, by = c('Region', 'Date')) %>%
  na.omit()
```

We merge the lagged weather features with the main data frame `Df` based on the columns `Region` and `Date` and remove missing observations using `na.omit`.

```r
vars.l1  ← paste0(vars, '_l1')
vars.l2  ← paste0(vars, '_l1')
vars.xgb ← c(vars, vars.l1, vars.l2)
```

We now create the input feature set to calibrate an XGBoost model. This set `vars.xgb` stores the original weather features and their lagged values (one-week and two-week lags).

```r
xgb.Df ← xgb.DMatrix(
  data  = as.matrix(Df %>% select(all_of(vars.xgb))),
  label = as.matrix(Df %>% select(Deaths))
)
```

We convert the data frame `Df` into an `xgb.DMatrix` object, which is required for the XGBoost model implementation in the package {xgboost}. The `data` argument includes the input features, and the `label` argument includes the response variable, which is the number of observed deaths.

```r
setinfo(xgb.Df, "base_margin", log(Df$bDeaths))
## [1] TRUE
```

We set the logarithm of the baseline number of deaths as an offset using the `setinfo` function from the {xgboost} package. As such, we try explaining deviations from the mortality baseline using the selected weather features.

Model calibration

```r
xgbcv ← xgb.cv(
  params = list(eta = 0.1,
                max_depth = 5,
                base_score = 0,
                objective = 'count:poisson'),
  data = xgb.Df,
  nfold = 10,
  early_stopping_rounds = 50,
  nrounds = 1000)
```

```r
xgbfit ← xgb.train(
  params = list(eta = 0.1,
                max_depth = 5,
                base_score = 0,
                objective = 'count:poisson'),
  data = xgb.Df,
  nrounds = xgbcv$best_iteration)
```

Using the `xgb.cv` function from the {xgboost} package, we perform `nfold` cross-validation to tune the number of boosting iterations. To limit computational time, we fix the learning rate `eta` at 0.1 and the maximum tree depth `max_depth` at 5. The initial prediction score `base_score` is set to 0 here, because we include the baseline number of deaths as the initial prediction (via the offset). We use Poisson regression by setting the `objective` to `'count:poisson'`. The `early_stopping_rounds` argument indicates that training will stop if performance does not improve for 50 rounds. The maximum number of boosting rounds is set to 1000 with `nrounds`.

We fit the XGBoost model on the entire training data `xgb.Df` using the optimal number of boosting rounds `xgbcv$best_iteration`.

Results

In-sample fit

Your turn

Q: We retrieve the estimated deaths from the calibrated machine learning model using the `predict` function. We add these to our mortality data set `Df` as a new column `xgbDeaths`.

```
Df$xgbDeaths ← predict(object = xgbfit, newdata = xgb.Df)
```

You can now visualize the observed deaths (in gray) alongside the deaths estimated by the baseline model (in blue) and the machine learning model (in red pink) for the cantons of Brabant Wallon and Antwerp. Complete the following {ggplot} code:

```
# Filter on Brabant Wallon province
Df.BW ← Df %>% …

# Plot
ggplot( … ) +
  geom_line(aes(x = … , y = … , col = 'Observed')) +
  geom_line(aes(x = … , y = … , col = 'XGBoost')) +
  geom_line(aes(x = … , y = … , col = 'Baseline')) +
  scale_colour_manual(values = c('gray80', RCLRbg, red_pink),
                      breaks = c('Observed', 'Baseline', 'XGBoost'), name = '') +
  xlab('Time') + ylab('Deaths') + theme_bw(base_size = 15) + ggtitle('Brabant Wallon')
```

In-sample fit

Your turn

A: We extract the mortality data set restricted to the province Brabant Wallon (`BE31`) using the `filter` function from the {dplyr} package. We plot the observed deaths (`Deaths`) in gray, the deaths estimated by the baseline model (`bDeaths`) in blue, and the deaths estimated by the machine learning model (`xgbDeaths`) in red pink.

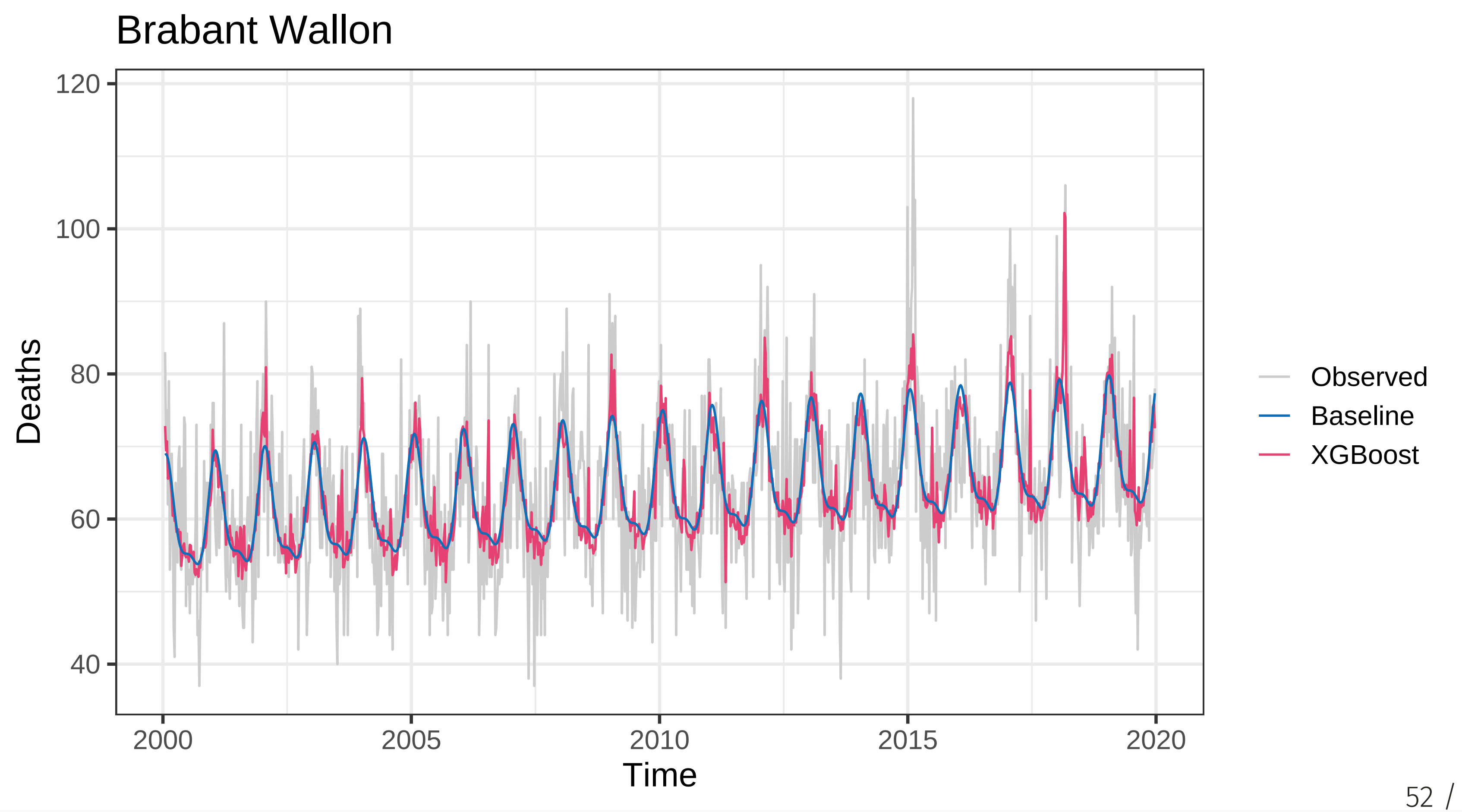
```r
# Filter on Brabant Wallon province
Df.BW <- Df %>% dplyr::filter(Region == 'BE31')

# Plot
ggplot(Df.BW) +
  geom_line(aes(x = Date, y = Deaths, col = 'Observed')) +
  geom_line(aes(x = Date, y = xgbDeaths, col = 'XGBoost')) +
  geom_line(aes(x = Date, y = bDeaths, col = 'Baseline')) +
  scale_colour_manual(values = c('gray80', RCLRbg, red_pink),
                      breaks = c('Observed', 'Baseline', 'XGBoost'),
                      name = '') +
  xlab('Time') + ylab('Deaths') + theme_bw(base_size = 15) +
  ggtitle('Brabant Wallon')
```

In-sample fit

Your turn

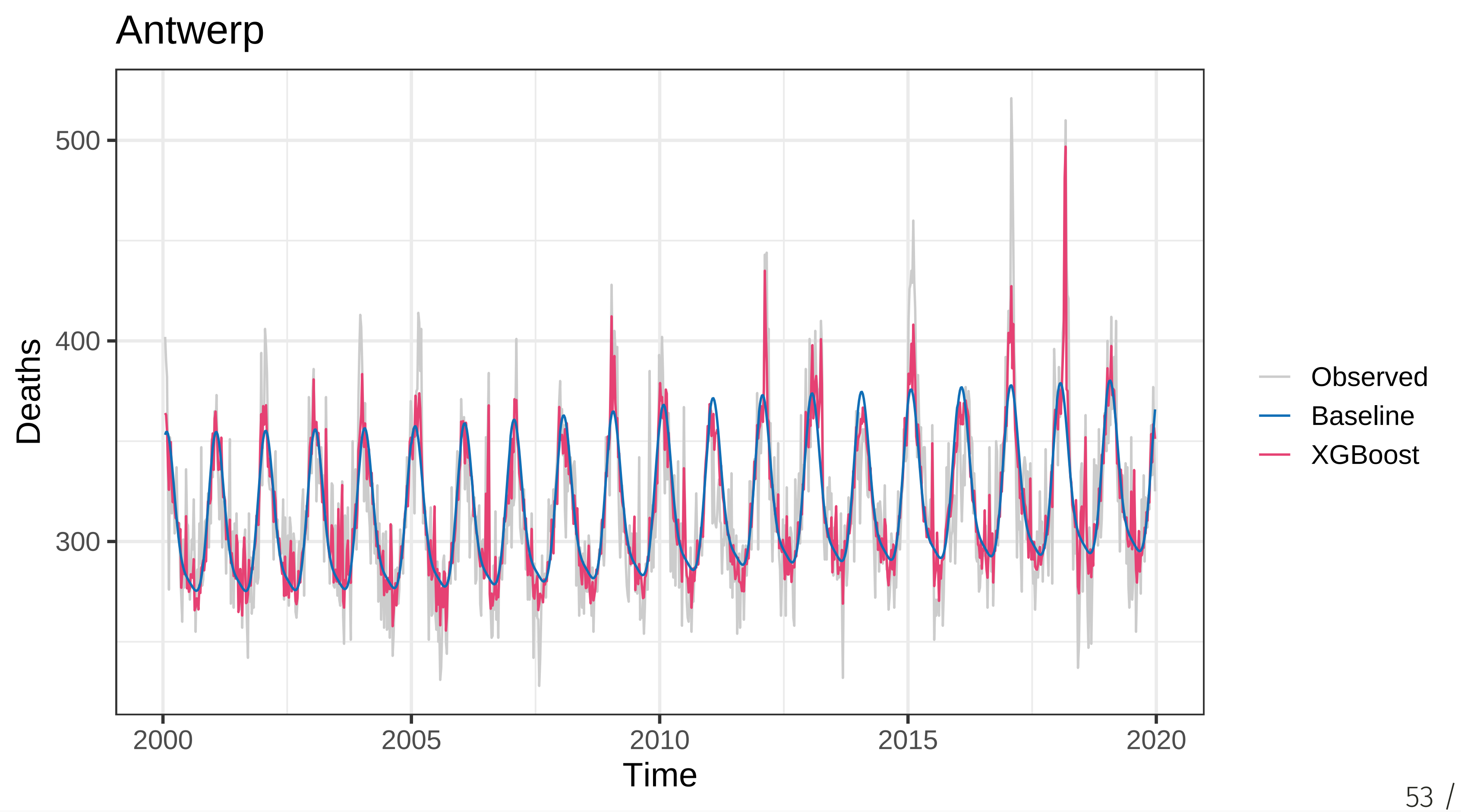
A:



In-sample fit

Your turn

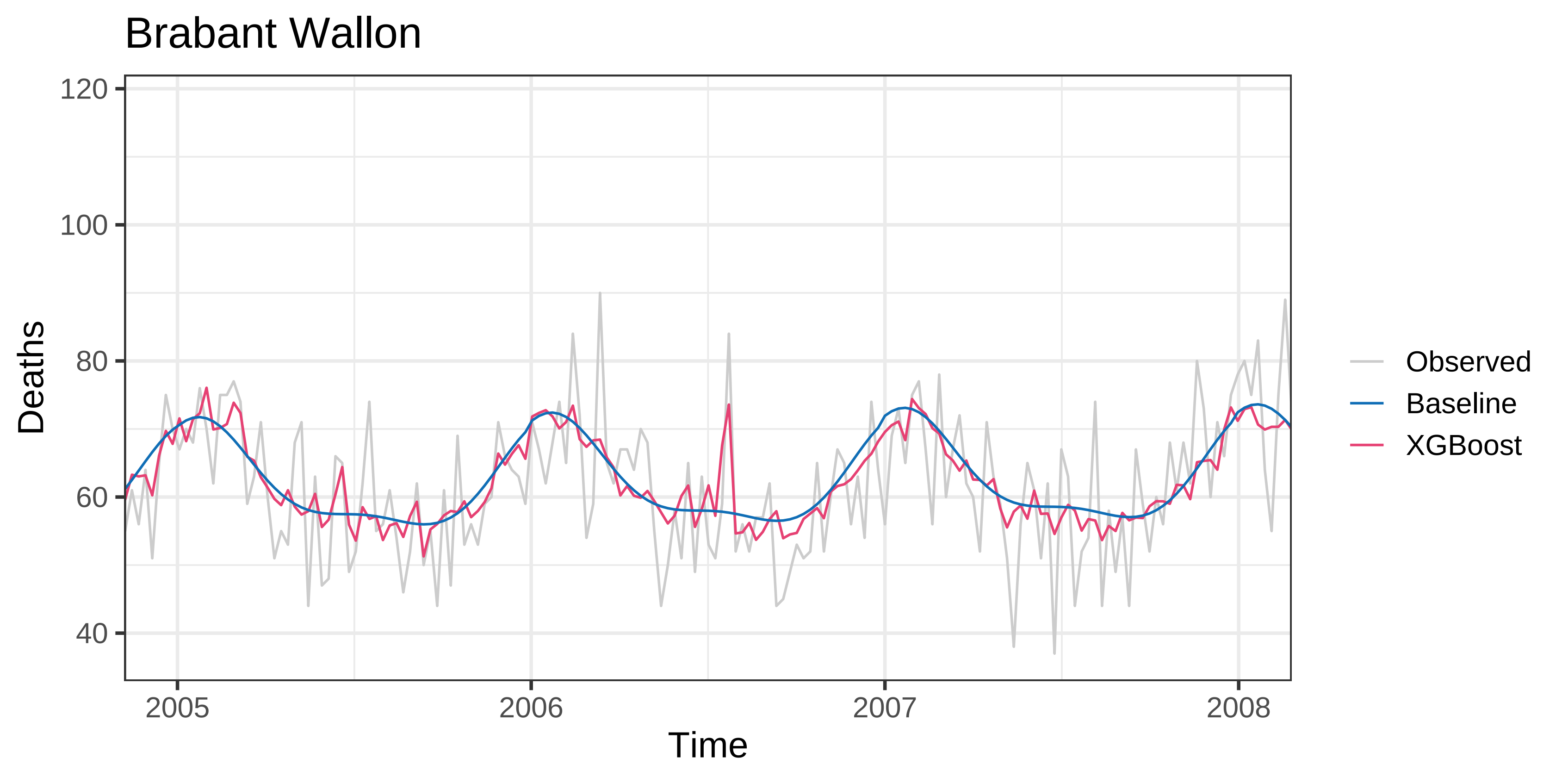
A: We repeat this procedure for the province of Antwerp.



In-sample fit

Your turn

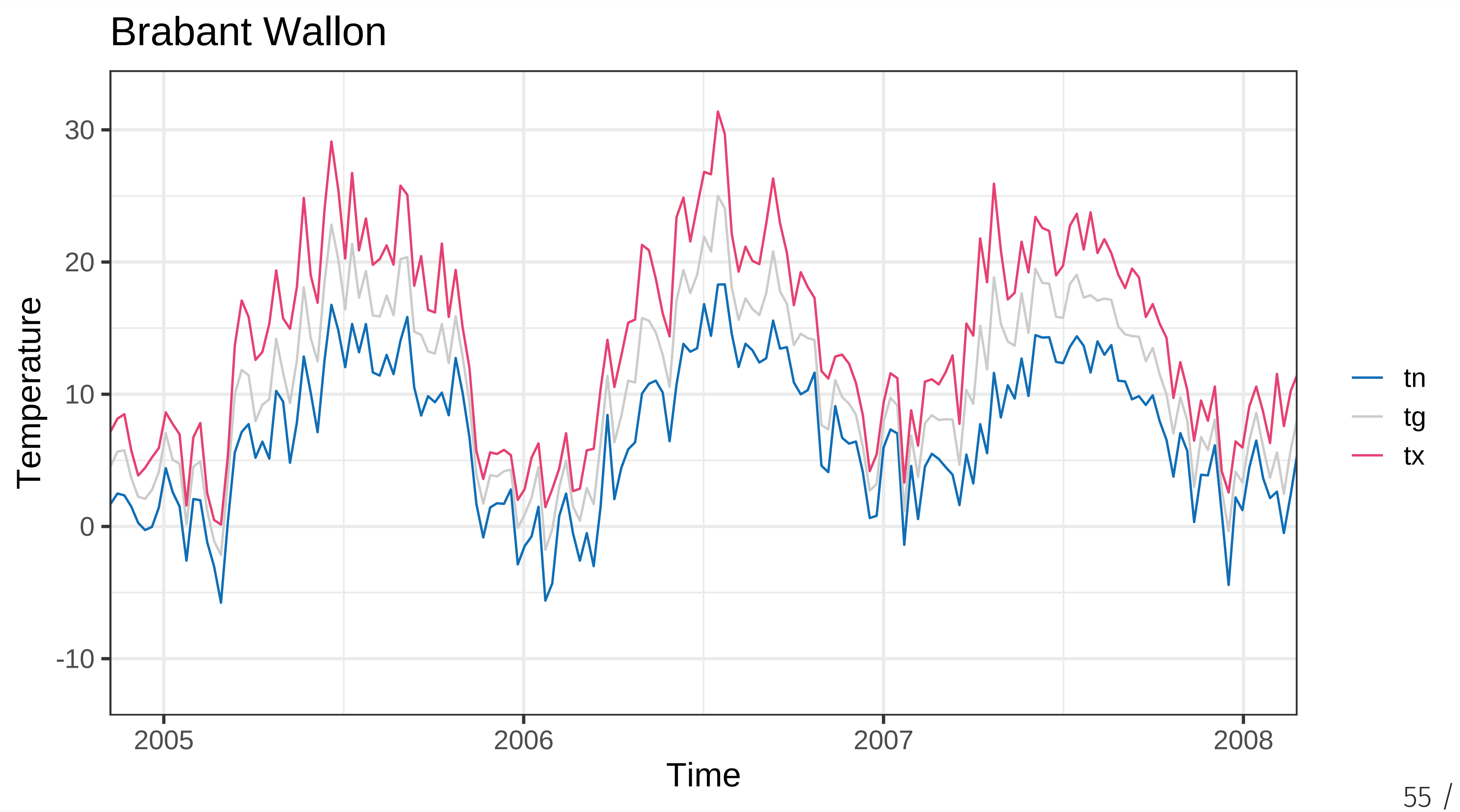
A: Zoom in on the years 2005-2007 and visualize the minimum, average and maximum temperature in Brabant Wallon.



In-sample fit

Your turn

A: Zoom in on the years 2005-2008: July 2006 heatwave extent in Europe



Thanks!

Slides created with the R package xaringan.

Course material available via

https://jensrobben.github.io/Workshop-LLN/