Insurance pricing analytics with R

Workshop on Pricing Analytics

Katrien Antonio, Roel Henckaerts and Roel Verbelen

Workshop pricing analytics | March, 2020

Prologue

Introduction

Workshop

https://github.com/katrienantonio/PE-pricing-analytics

The workshop repo on GitHub, where we upload presentation, code, data sets, etc.

Us

https://katrienantonio.github.io/

katrien.antonio@kuleuven.be & roel.henckaerts@kuleuven.be & roel.verbelen@gmail.com

(Katrien) Professor in insurance data science

(Roel H.) PhD student in insurance data science

(Roel V.) PhD, now with Finity Consulting in Sydney (Australia)

Checklist

☑ Do you have a fairly recent version of R?

```
version$version.string
```

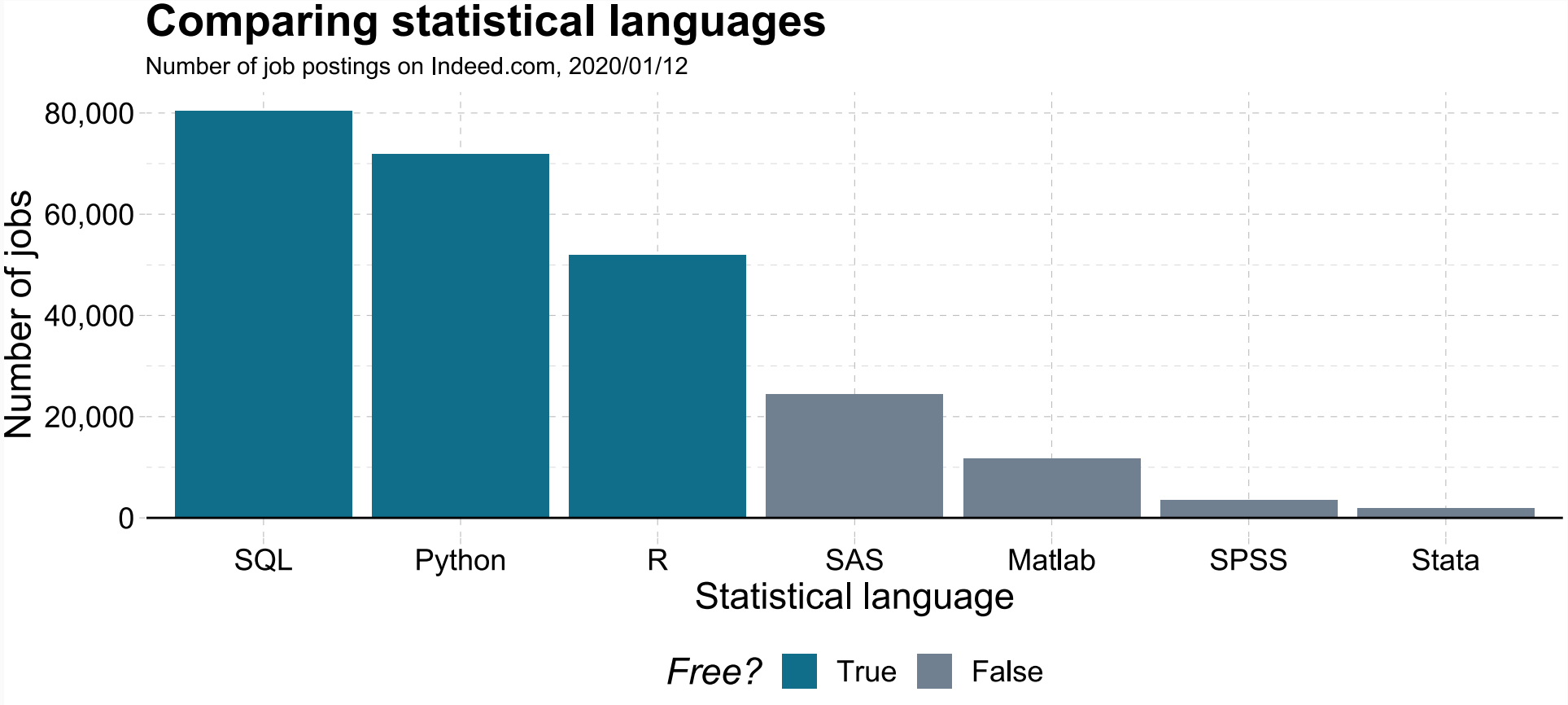
☑ Do you have a fairly recent version of RStudio?

```
RStudio.Version()$version
## Requires an interactive session but should return something like "[1] ‘1.2.5001’"
```

☑ Have you installed the R packages listed in the software requirements?

What's out there - the R universe

Why R and RStudio?



This graph is created from the search results obtained via www.indeed.com (on Jan 12, 2020), using Grant McDermott's code for `ggplot2`, see lecture 1 in his Data science for economists course.

Why R and RStudio? (cont.)

Data science positivism

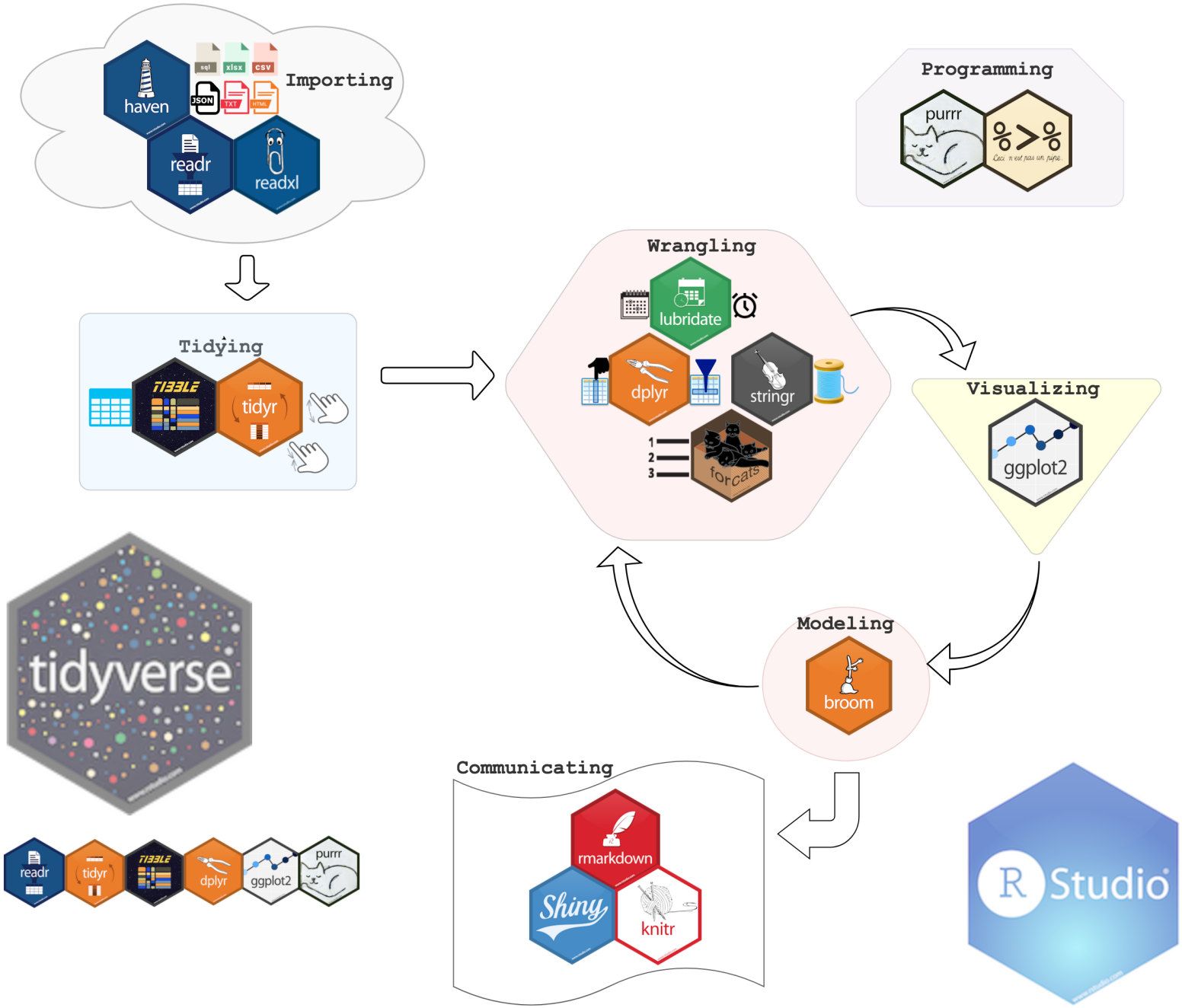
- Next to Python, R has become the *de facto* language for data science, with a cutting edge *machine learning toolbox*.
- See: The Popularity of Data Science Software
- R is open-source with a very active community of users spanning academia and industry.

Bridge to actuarial science, econometrics and other tools

- R has all of the statistics and econometrics support, and is amazingly adaptable as a “glue” language to other programming languages and APIs.
- R does not try to be everything to everyone. The RStudio IDE and ecosystem allow for further, seemless integration (with e.g. python, keras, tensorflow or C).
- Widely used in actuarial undergraduate programs

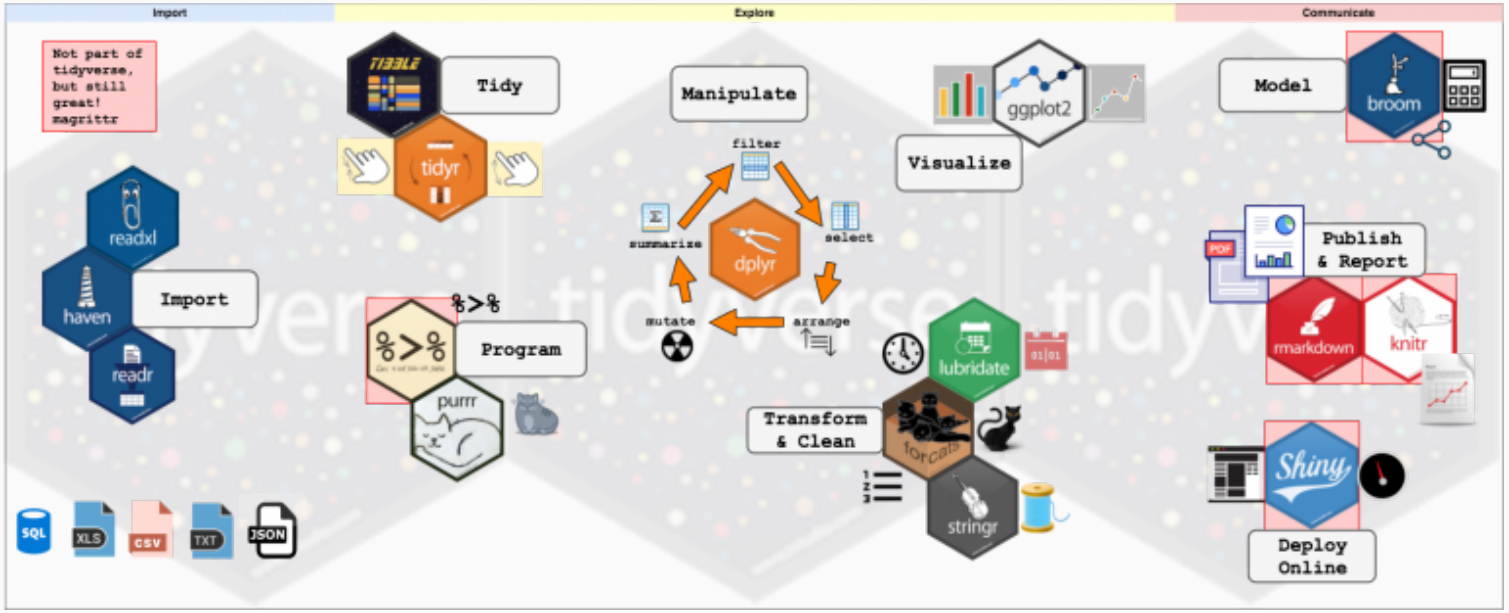
Disclaimer + Read more

- It's also the language that we know best.
- If you want to read more: R-vs-Python, when to use Python or R or Hadley Wickham on the future of R


Importing
haven
readr
readxl
Programming
purrr
Tidying
tidyr
Wrangling
lubridate
dplyr
stringr
forcats
Visualizing
ggplot2
Modeling
broom
Communicating
rmarkdown
Shiny
knitr
RStudio
tidyverse
readr
tidyr
dplyr
ggplot2
purrr

Welcome to the tidyverse!

> The **tidyverse** is an opinionated collection of R packages designed for data science. All packages share an underlying design philosophy, grammar, and data structures.



More on: tidyverse.

Install the packages with `install.packages("tidyverse")`. Then run `library(tidyverse)` to load the core tidyverse.

Principles of tidy data

Three interrelated rules from the R for data science book by Garrett Grolemund and Hadley Wickham:

1. Each variable must have its own column.
2. Each observation must have its own row.
3. Each value must have its own cell.

This figure is taken from Chapter 12 on Tidy data in R for data science.

Workflow of a data scientist

Here is a model of the **tools needed in a typical data science project**:

> Together, tidying and transforming are called **wrangling**, because getting your data in a form that's natural to work with often feels like a fight!
>
> Models are complementary tools to visualisation. Once you have made your questions sufficiently precise, you can use a model to answer them. Models are a fundamentally **mathematical or computational tool**, so they generally scale well. But every model makes **assumptions**, and by its very nature a model cannot question its own assumptions. That means **a model cannot fundamentally surprise you**.

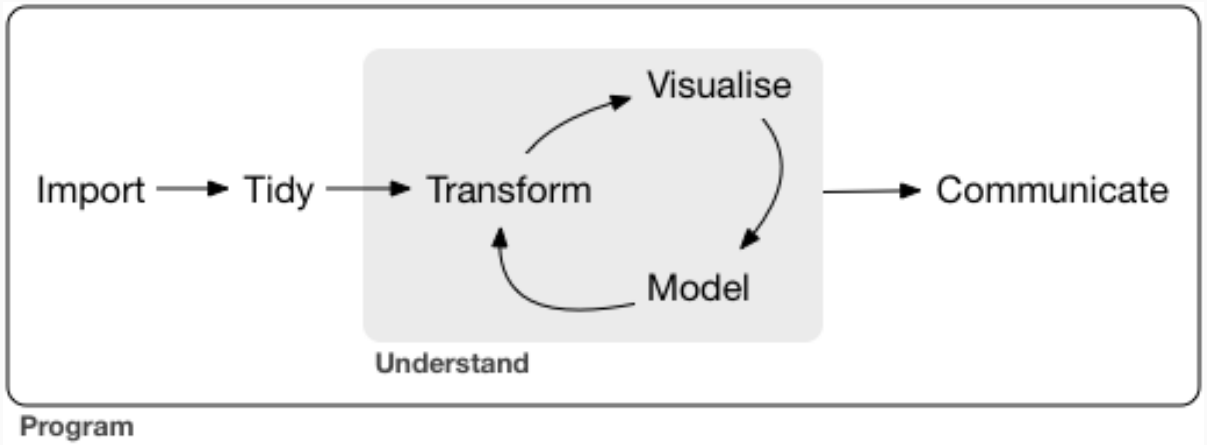


Figure and quote taken from Chapter 1 in R for data science.

Today's Outline

- Prologue
- What's out there: the R universe
  - why R and RStudio?
  - welcome to the tidyverse!
  - principles of tidy data
  - workflow of a data scientist
- Data wrangling and visualisation
  - tibbles
  - Pipe operator `%>%`
  - {dplyr} instructions
  - {ggplot2}
- Data set used in the workshop
  - exploratory data analysis, including maps with {sf}, {tmap}
- Model building
  - basic concepts of `glm` and `mgcv::gam` for GAMs
  - smooth effect of a continuous risk factor, a spatial smoother
  - smooth effect of an interaction between two continuous risk factors
- From GAM to GLM
  - binning the smooth spatial effect using {classInt}
  - binning a smoother of a continuous risk factor using {evtree}
  - putting it all together.

Preliminaries on data wrangling and visualisation

A tibble instead of a data.frame

Within the tidyverse `tibble` is a modern take on a `data.frame`:

- keep the features that have stood the test of time
- drop the features that used to be convenient but are now frustrating.

You can use:

- `tibble()` to create a new tibble
- `as_tibble()` transforms an object (e.g. a data frame) into a tibble.

Quick example: explore the differences!

```
mtcars
# install.packages("tidyverse")
library(tidyverse)
as_tibble(mtcars)
```

Chains with the pipe operator

In R, the pipe operator is `%>%`.

It takes the output of one statement and makes it the input of the next statement.

When describing it, you can think of it as a "THEN"; with this operator it becomes easy to chain a sequence of calculations.

For example, when you have an input data and want to call functions `foo` and `bar` in sequence, you can write `data %>% foo %>% bar`.

A first example:

- take the `diamonds` data (from the {ggplot2} package)
- then subset

```
diamonds %>% filter(cut == "Ideal")
```

Some excellent blog posts about this operator: Pipes in R tutorial for beginners and how to write this in base R.

Data manipulation verbs

The {dplyr} package holds many useful data manipulation verbs:

- `mutate()` adds new variables that are functions of existing variables
- `select()` picks variables based on their names
- `filter()` picks cases based on their values
- `summarise()` reduces multiple values down to a single summary
- `arrange()` changes the ordering of the rows.

These all combine naturally with `group_by()` which allows you to perform any operation “by group”.

A first example:

```
diamonds %>% mutate(price_per_carat = price/carat) %>% filter(price_per_carat > 1500)
```

or

```
diamonds %>% group_by(cut) %>% summarize(price = mean(price), carat = mean(carat))
```

Plots with ggplot2

The aim of the {ggplot2} package is to create elegant data visualisations using the **grammar of graphics**.

Here are the basic steps:

- begin a plot with the function `ggplot()` creating a coordinate system that you can add layers to
- the first argument of `ggplot()` is the dataset to use in the graph

A first example

```
library(ggplot2)
ggplot(data = mpg)
ggplot(mpg)
```

creates an empty graph.

You will now add layers to this graph!

Plots with ggplot2

You complete your graph by adding one or more **layers** to `ggplot()`.

For example:

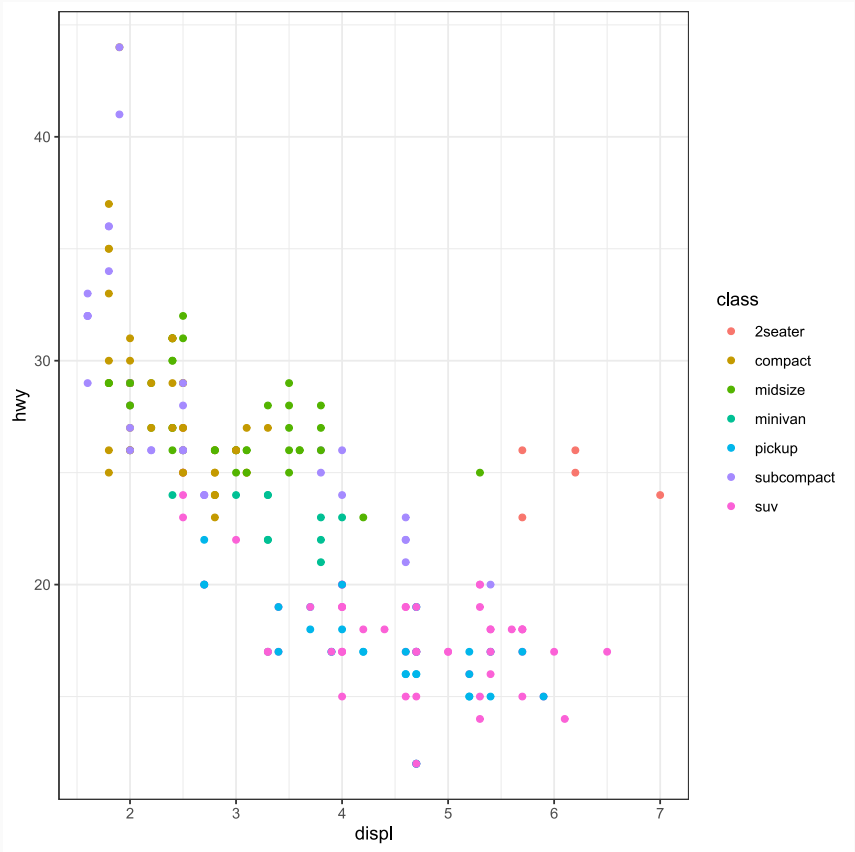
- `geom_point()` adds a layer of points to your plot, which creates a scatterplot
- `geom_smooth()` adds a smooth line
- `geom_bar` a bar plot

and many more, see ggplot2 documentation.

Each `geom` function in `ggplot2` takes an aesthetic mapping argument:

- maps variables in your dataset to visual properties
- always paired with `aes()` and the x and y arguments of `aes()` specify which variables to map to the x and y axes.

```
library(ggplot2)
ggplot(mpg, aes(displ, hwy, colour = class)) +
  geom_point() + theme_bw()
```



Extend the empty graph now with (here: global) aesthetic mapping argument `aes(displ, hwy, colour = class)`.

This implies: `displ` on the x-axis, `hwy` on the y-axis and `class` to differentiate the color of the plotting symbol.

With `geom_point` you add a layer of points to the empty graph.

`theme_bw()` changes the `ggtheme` to a simple black-and-white theme.

Data set used in the workshop

Data set used in this course - MTPL

We will use the Motor Third Party Liability data set. There are 163,231 policyholders in this data set.

The frequency of claiming (`nclaims`) and corresponding severity (`avg`, the amount paid on average per claim reported by a policyholder) are the **target variables** in this data set.

Predictor variables are:

- the exposure-to-risk, the duration of the insurance coverage (max. 1 year)
- factor variables, e.g. gender, coverage, fuel
- continuous, numeric variables, e.g. age of the policyholder, age of the car
- spatial information: postal code (in Belgium) of the municipality where the policyholder resides.

More details in Henckaerts et al. (2018, Scandinavian Actuarial Journal) and Henckaerts et al. (2019, arxiv).

Data set used in this course - MTPL

You can load the data from a script in the `scripts` folder as follows:

```
# install.packages("rstudioapi")
dir ← dirname(rstudioapi::getActiveDocumentContext()$path)
setwd(dir)
mtpl_orig ← read.table('./data/P&Cdata.txt',
                                header = TRUE)
mtpl_orig ← as_tibble(mtpl_orig)
```

Some basic exploratory steps with this data follow on the next sheet.

Data set used in this course - MTPL

Note that the data `mtpl_orig` uses capitals for the variable names

```
mtpl_orig %>% slice(1:3) %>% select(-LONG, -LAT) %>% kable(format = 'html')
```

ID	NCLAIMS	AMOUNT	AVG	EXP	COVERAGE	FUEL	USE	FLEET	SEX	AGEPH	BM	AGEC	POWER	PC	TOWN
1	1	1618.001	1618.001	1	TPL	gasoline	private	N	male	50	5	12	77	1000	BRUSSEL
2	0	0.000	NA	1	PO	gasoline	private	N	female	64	5	3	66	1000	BRUSSEL
3	0	0.000	NA	1	TPL	diesel	private	N	male	60	0	10	70	1000	BRUSSEL

We change this to lower case variables, and rename `exp` to `expo`.

```
mtpl ← mtpl_orig %>%
  # rename all columns
  rename_all(function(.name) {
    .name %>%
      # replace all names with the lowercase versions
      tolower
    })
mtpl ← rename(mtpl, expo = exp)
```

Let's explore different ways to calculate the **empirical claim frequency**.

Inspect the following instructions and list pros and cons:

```
mean(mtpl$nclaims)
sum(mtpl$nclaims)/sum(mtpl$expo)
mtpl %>% summarize(emp_freq = sum(nclaims) / sum(expo))
```

```
## [1] 0.1239715
## [1] 0.1393352
```

emp_freq
0.1393352

What about the **empirical variance**?

```
m ← sum(mtpl$nclaims)/sum(mtpl$expo)
m
## [1] 0.1393352
var ← sum((mtpl$nclaims - m * mtpl$expo)^2)/sum(mtpl$e
var
## [1] 0.1517246
```

Here we use the expression for the variance:

$$\mathrm{Var}(X) = \mathrm{E}[(X - \mathrm{E}X)^2]$$

We empirically calculate the first expected value by taking exposure into account.

Moreover, we compare each realization of `nclaims` with its expected value, i.e. `m * expo` where `m` is the global empirical claim frequency.

```
dim(mtpl)
```

```
## [1] 163231     18
```

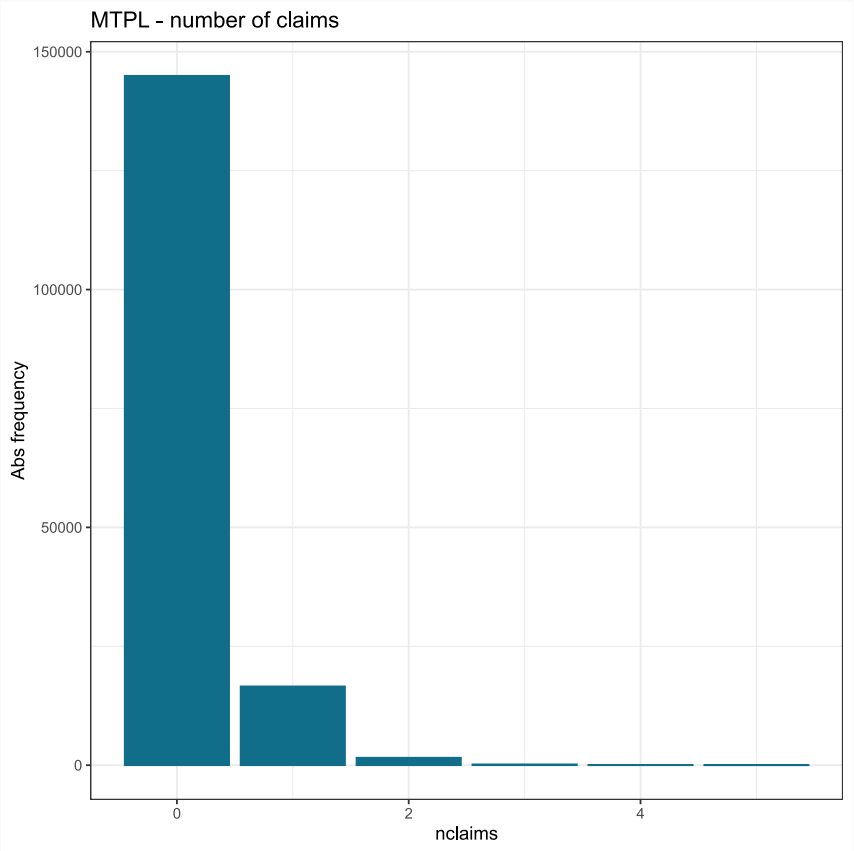
```
mtpl %>% summarize(emp_freq =
                     sum(nclaims) / sum(expo))
```

emp_freq
0.1393352

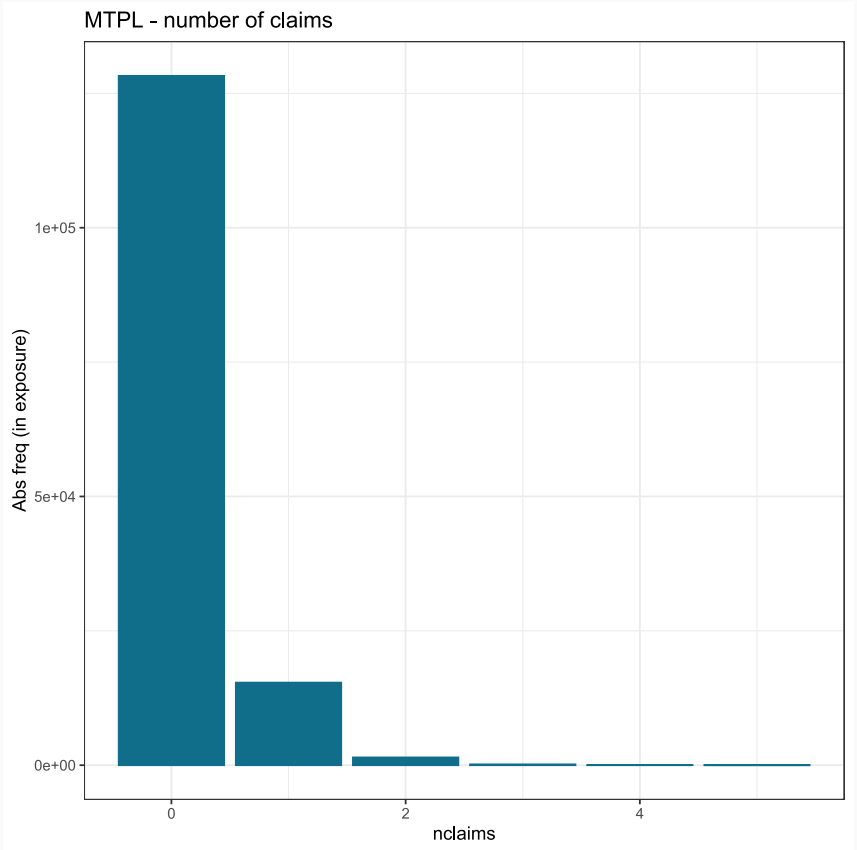
```
mtpl %>%
  group_by(sex) %>%
  summarize(emp_freq = sum(nclaims) / sum(expo))
```

sex	emp_freq
female	0.1484325
male	0.1361164

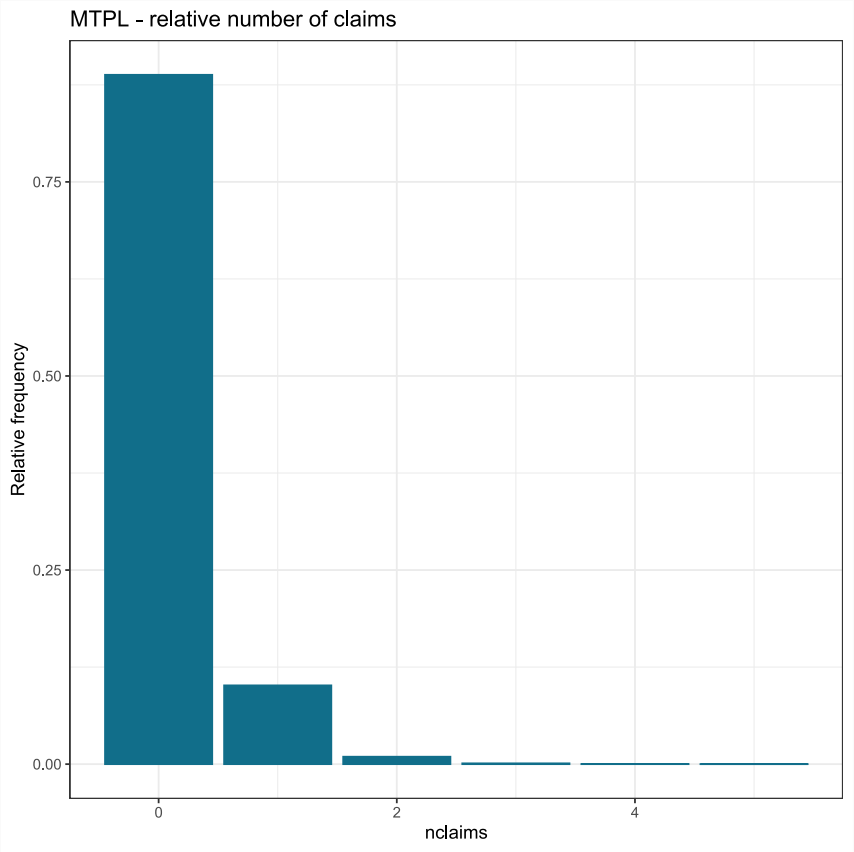
```
KULbg ← "#116E8A"
g ← ggplot(mtpl, aes(nclaims)) + theme_bw() +
     geom_bar(col = KULbg, fill = KULbg) +
     labs(y = "Abs frequency") +
     ggtitle("MTPL - number of claims")
g
```



```
KULbg ← "#116E8A"
g ← ggplot(mtpl, aes(nclaims)) + theme_bw() +
     geom_bar(aes(weight = expo), col = KULbg,
              fill = KULbg) +
     labs(y = "Abs freq (in exposure)") +
     ggtitle("MTPL - number of claims")
g
```



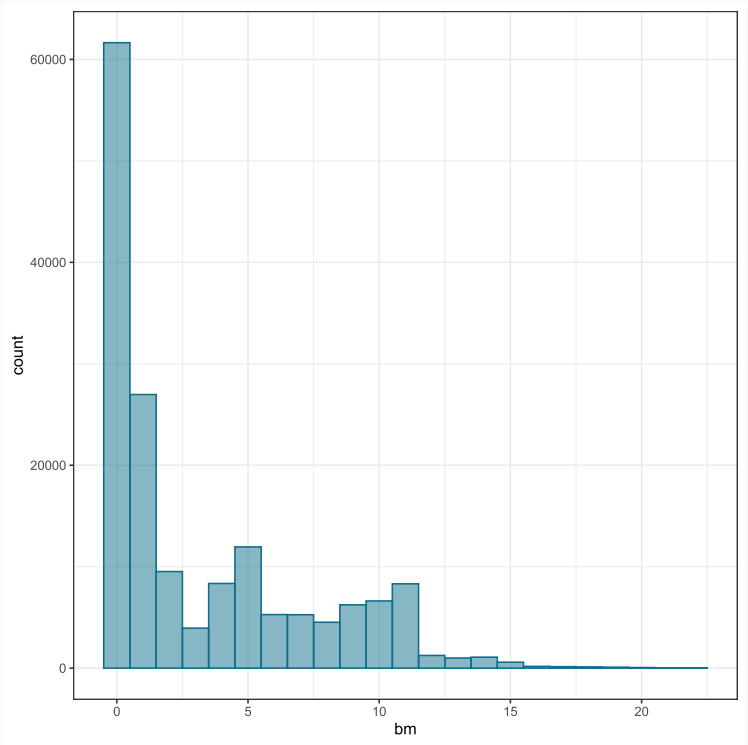

```
g ← ggplot(mtpl, aes(nclaims)) + theme_bw()
g + geom_bar(aes(y = (..count..)/sum(..count..)),
             col = KULbg, fill = KULbg) +
  labs(y = "Relative frequency") +
  ggtitle("MTPL - relative number of claims")
```



We now step from the barplot to a histogram (in {ggplot2}), see ggplot2 histogram.

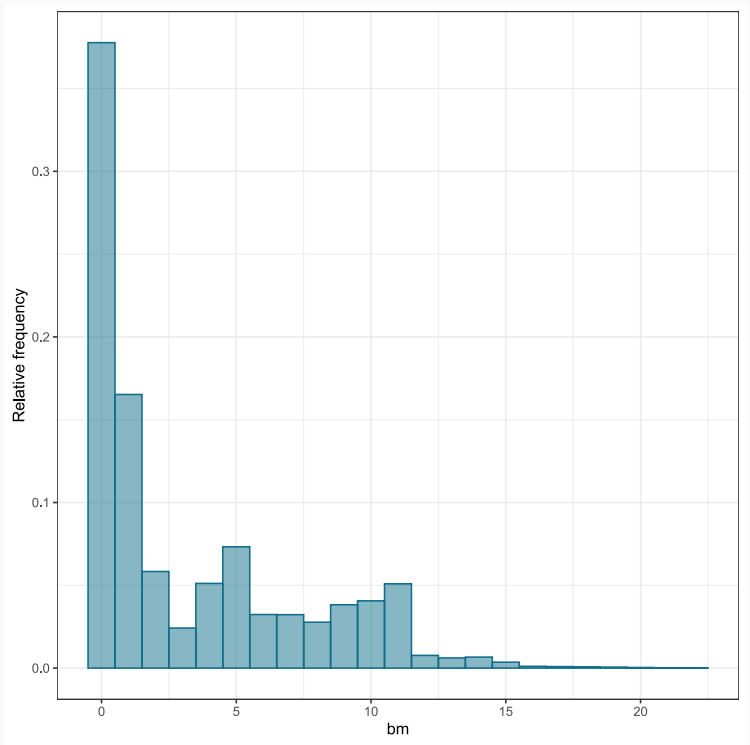
Here is the histogram of `bm` showing how the policyholders are distributed over levels in the bonus-malus scale:

```
g ← ggplot(mtpl, aes(bm)) + theme_bw()
g + geom_histogram(binwidth = 1, col = KULbg,
                   fill = KULbg, alpha = .5)
```



For the relative frequency histogram

```
g ← ggplot(mtpl, aes(bm)) + theme_bw()
g + geom_histogram(aes(y = (..count..)/sum(..count..)),
                   binwidth = 1, col = KULbg,
                   fill = KULbg, alpha = .5) +
  labs(y = "Relative frequency")
```



Your turn

To get warmed up, let's load the `mtpl` data and do some basic investigations into the variables. The idea is to get a feel for the data.

Your starting point are the instructions in the R script on data explorations from the course material.

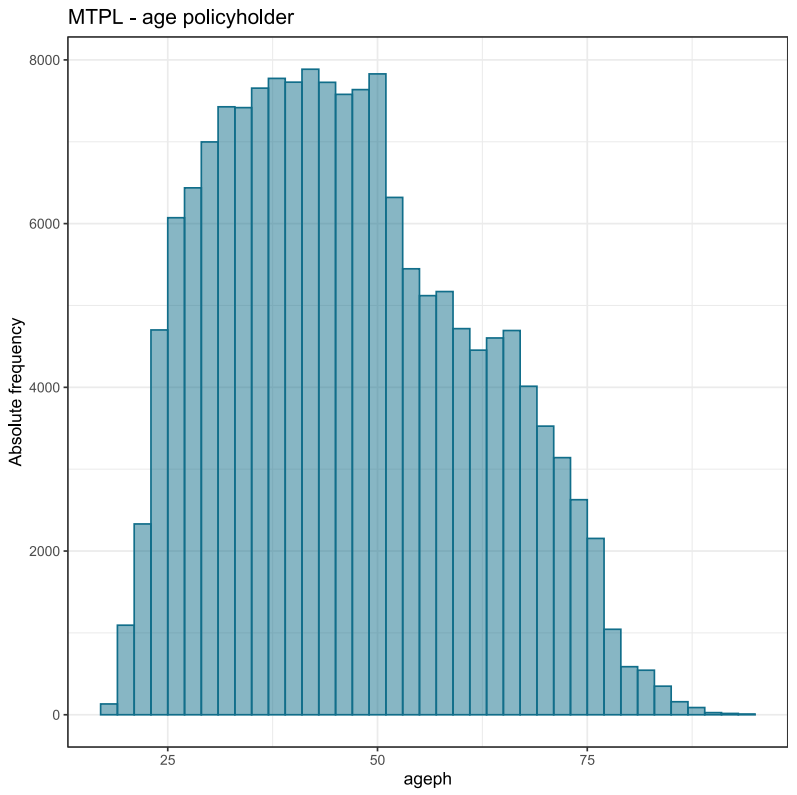
Q: you will work through the following exploratory steps.

1. Visualize the distribution of the `ageph` with a histogram.
2. For each age recorded in the data set `mtpl`: what is the total number of observations, the total exposure, and the corresponding total number of claims reported?
3. Calculate the empirical claim frequency, per unit of exposure, for each age and picture it. Discuss this figure.
4. Repeat the above for `bm`, the level occupied by the policyholder in the Belgian bonus-malus scale.

10:00

For **Q.1** a histogram of `ageph`

```
ggplot(data = mtpl, aes(ageph)) + theme_bw() +
      geom_histogram(binwidth = 2, col = KULbg,
                                   fill = KULbg, alpha =
      labs(y = "Absolute frequency") +
      ggtitle("MTPL - age policyholder")
```



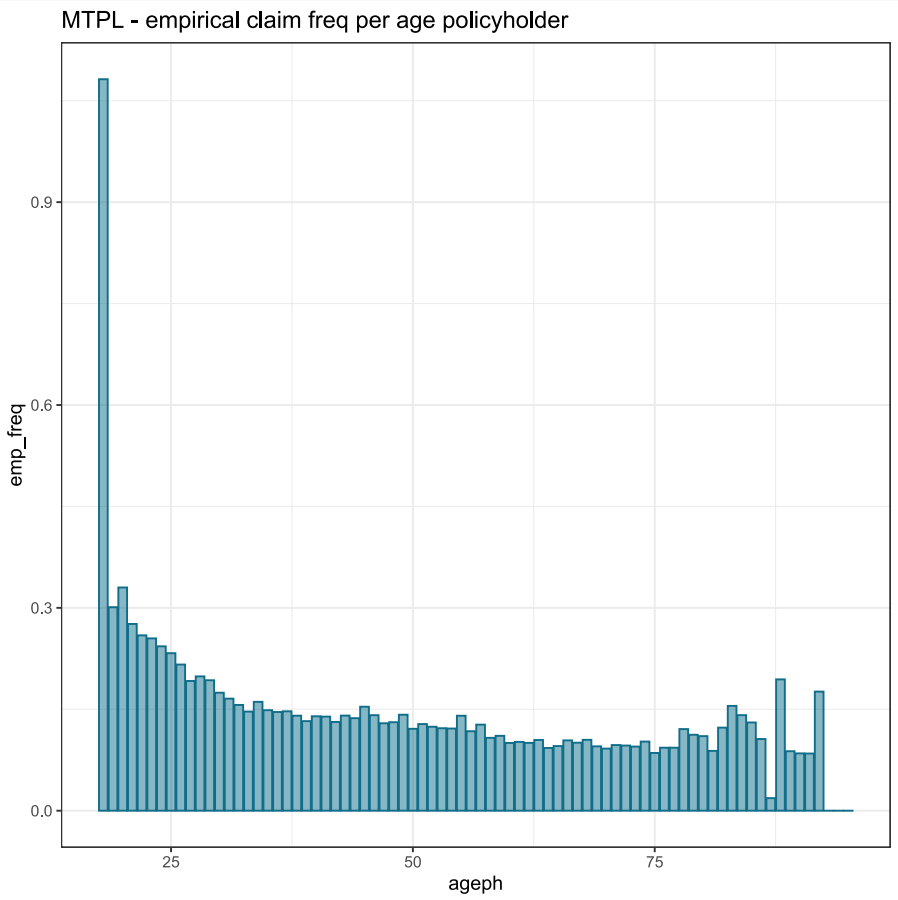
For **Q.2** for each `ageph` recorded

```
mtpl %>%
  group_by(ageph) %>%
  summarize(tot_claims = sum(nclaims),
            tot_expo = sum(expo), tot_obs = n())
```

ageph	tot_claims	tot_expo	tot_obs
18	5	4.621918	16
19	28	93.021918	116
20	113	342.284932	393
21	165	597.389041	701
22	202	778.827397	952
23	297	1165.358904	1379
24	426	1752.249315	2028
25	546	2343.504110	2673

For **Q.3** for each `ageph` recorded

```
freq_by_age ← mtpl %>%
  group_by(ageph) %>%
  summarize(emp_freq = sum(nclaims) / sum(expo))

ggplot(freq_by_age, aes(x = ageph, y = emp_freq)) +
                                        theme_bw() +
  geom_bar(stat = "identity", color = KULbg,
                          fill = KULbg, alpha = .5) +
  ggtitle("MTPL - empirical claim freq per
                                age policyholder")
```

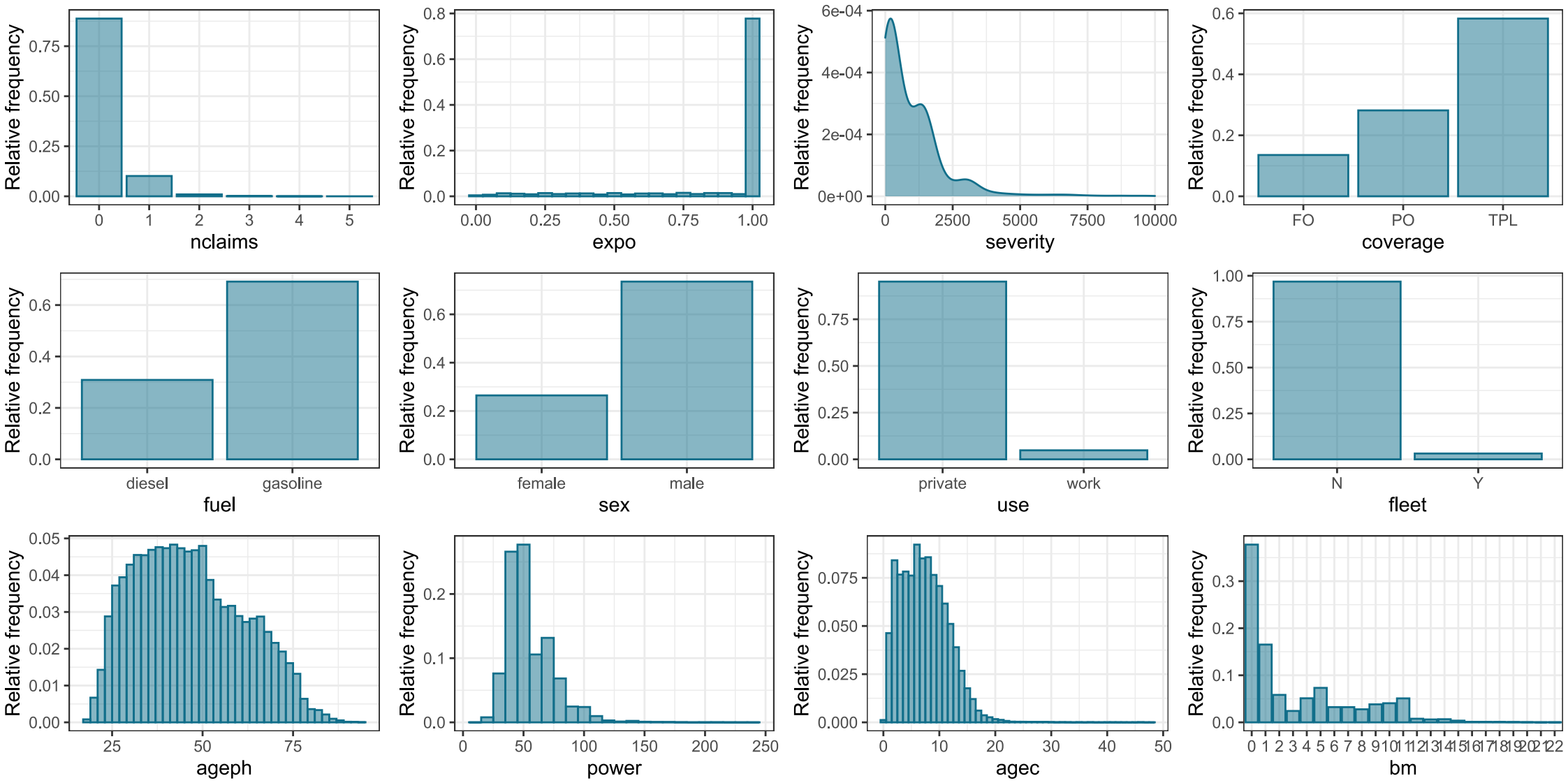


For **Q.4** recycle the above instructions and replace `ageph` with `bm`.

To reproduce the graphs from the paper, we write functions as a wrapper for the instructions used above.

```
# or, use the colors from our paper
col ← KULbg
fill ← KULbg
ylab ← "Relative frequency"

# wrapper functions
ggplot.bar ← function(DT, variable, xlab){
  ggplot(data = DT, aes(as.factor(variable))) + theme_bw() +
    geom_bar(aes(y = (..count..)/sum(..count..)), col = col, fill = fill, alpha = 0.5) + labs(x = xlab, y = ylab)
}

ggplot.hist ← function(DT, variable, xlab, binwidth){
  ggplot(data = DT, aes(variable)) + theme_bw() +
    geom_histogram(aes(y = (..count..)/sum(..count..)), binwidth = binwidth, col = col, fill = fill, alpha = 0.5) +
    labs(x = xlab, y = ylab)
}
```

Let's give these a try!


Relative frequency
nclaims
expo
severity
coverage
FO
PO
TPL
fuel
diesel
gasoline
sex
female
male
use
private
work
fleet
N
Y
ageph
power
agec
bm

Spatial data with {rgdal}

With the {rgdal} package we load the **shapefile** of Belgium at postal code level as follows:

```
library(rgdal)
readShapefile = function(){
  belgium_shape ← readOGR(dsn = path.expand(paste(getwd(), "./shape file Belgie postcodes", sep = "")),
                  layer = "npc96_region_Project1")
  belgium_shape ← spTransform(belgium_shape, CRS("+proj=longlat +datum=WGS84"))
  belgium_shape$id ← row.names(belgium_shape)
  return(belgium_shape)
}

belgium_shape = readShapefile()
## OGR data source with driver: ESRI Shapefile
## Source: "C:\Users\u0043788\Dropbox\Data science for non-life insurance\Computer labs\Pricing analytics in xaringan
## with 1146 features
## It has 6 fields
class(belgium_shape)
## [1] "SpatialPolygonsDataFrame"
## attr(,"package")
## [1] "sp"
```

Spatial data with {rgdal}

With the {rgdal} package we load the **shapefile** of Belgium at postal code level as follows.

Attributes or variables are stored as a data frame in the `@data` of the `SpatialPolygonsDataFrame`.

You can access it as follows.

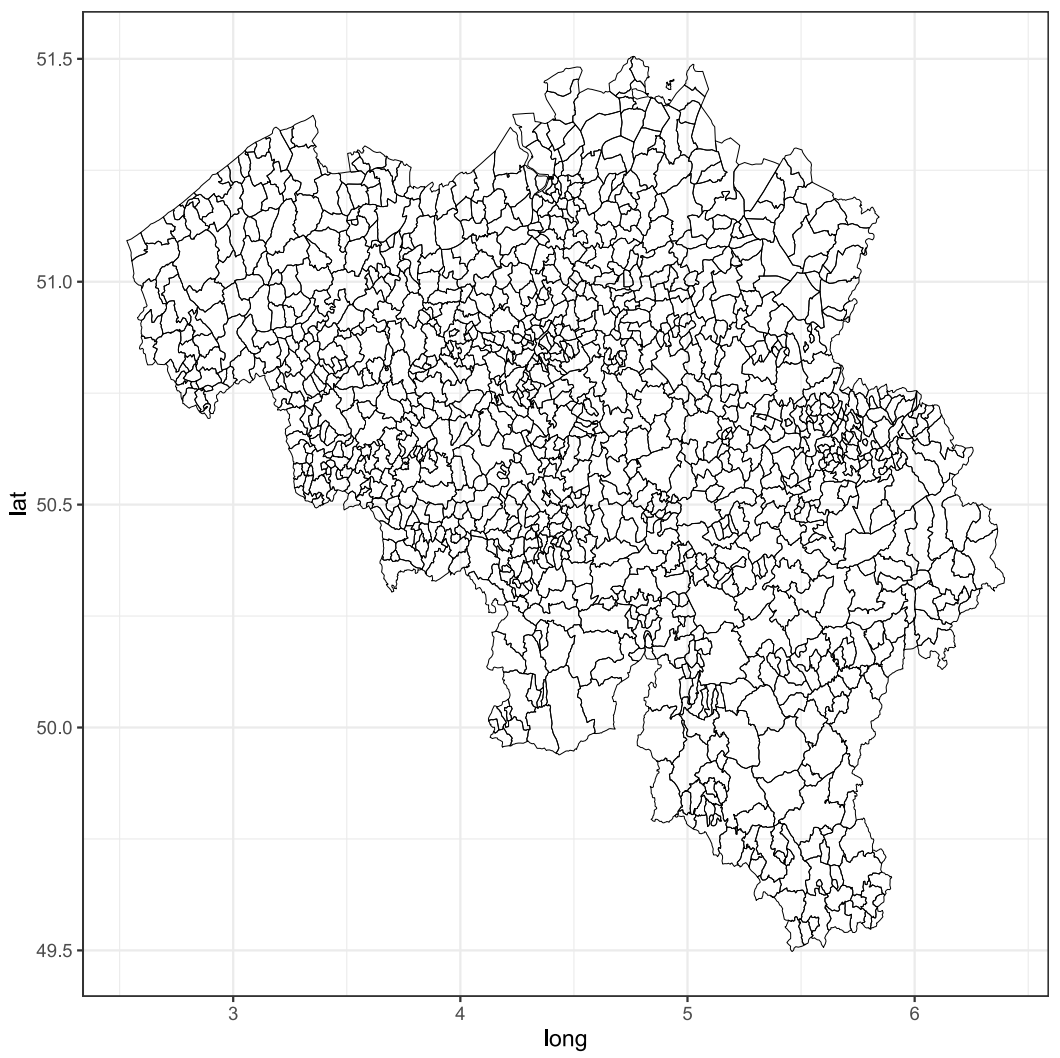
```
str(belgium_shape@data)
## 'data.frame':    1146 obs. of  7 variables:
##  $ POSTCODE  : int  1300 1301 1310 1315 1320 1325 1330 1331 1332 1340 ...
##  $ NAAM1     : Factor w/ 22 levels "ANDERLECHT","BRUSSEL",..: NA NA NA NA NA NA NA NA NA NA ...
##  $ NAAM2     : Factor w/ 1138 levels "'s Gravenvoeren",..: 1070 118 561 499 82 204 844 854 358 778 ...
##  $ POSTCODELA: Factor w/ 1146 levels "1000","1020",..: 23 24 25 26 27 28 29 30 31 32 ...
##  $ Shape_Leng: num  42814 15954 20276 43081 29648 ...
##  $ Shape_Area: num  32629722 9641834 15381287 38638405 38812269 ...
##  $ id        : chr  "0" "1" "2" "3" ...
```

Plotting just an empty map of Belgium, i.e. postcal code areas not filled with any color then goes as follows:

```
plot.eda.map ← ggplot(belgium_shape,
                      aes(long, lat,
                          group = group)) +
  geom_polygon(fill = NA, colour = "black",
               size = 0.1) +
  theme_bw()
plot.eda.map
```

Under the hood, the `ggplot` instruction transforms the shapefile into a data frame. This is what the `fortify` function does, e.g.

```
fortify(belgium_shape) %>% slice(1:3)
##       long      lat order  hole piece id group
## 1 4.550931 50.74536     1 FALSE     1  0   0.1
## 2 4.552253 50.74755     2 FALSE     1  0   0.1
## 3 4.555582 50.74903     3 FALSE     1  0   0.1
```

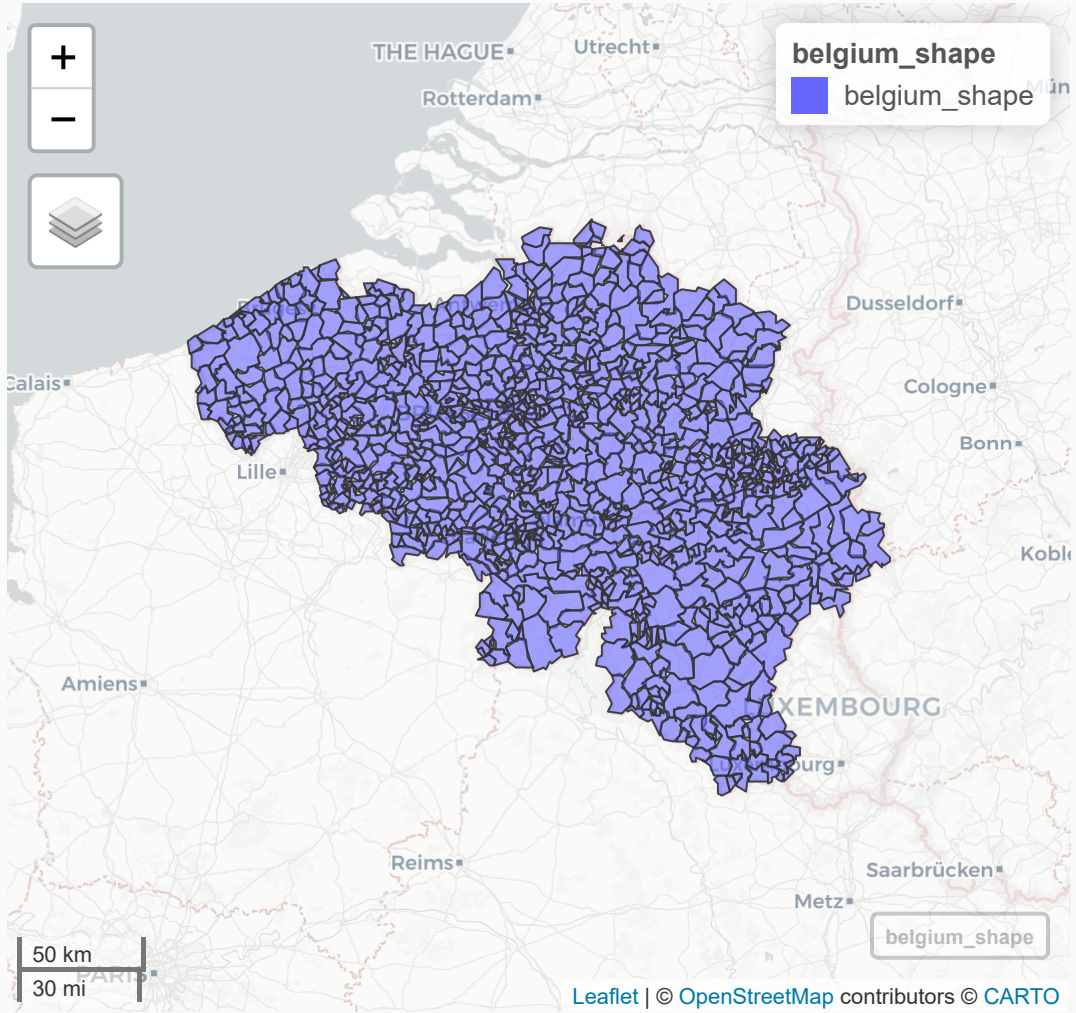


A quick, interactive look using `mapview` from the {mapview} package.

This package provides functions to very quickly and conveniently create interactive visualisations of spatial data.

```
mapview(belgium_shape)
```


```
mapview(belgium_shape)
```

Spatial data with {sf}

Alternatively, we can use the {sf} package to load the **shapefile** of Belgium at postal code level as follows:

```
library(sf)
belgium_shape_sf ← st_read('./shape file Belgie postcodes/npc96_region_Project1.shp', quiet = TRUE)
belgium_shape_sf ← st_transform(belgium_shape_sf, "+proj=longlat +datum=WGS84")
```

We inspect the resulting object:

```
class(belgium_shape_sf)
## [1] "sf"         "data.frame"
belgium_shape_sf %>% as_tibble() %>% select(-geometry) %>% slice(1:3) %>% kable(format = 'html')
```

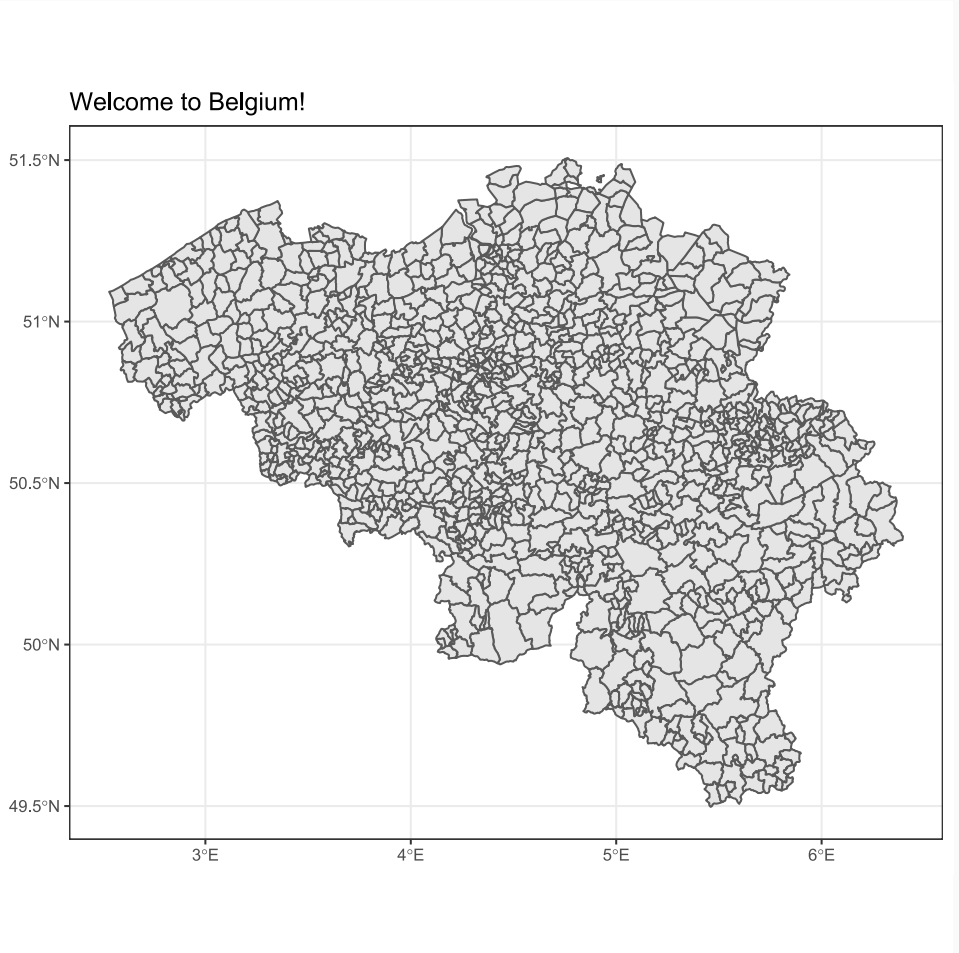
POSTCODE	NAAM1	NAAM2	POSTCODELA	Shape_Leng	Shape_Area
1300	NA	Wavre	1300	42814.06	32629722
1301	NA	Bierges	1301	15954.49	9641834
1310	NA	La Hulpe	1310	20276.46	15381287

With the packages {sp} and {rgdal}, spatial objects had to be converted to data frames (e.g. with `fortify()`) prior to plotting with {ggplot2}.

However, with {sf} the objects are already data frames and they can be plotted with the help of the new `geom_sf()` layer in {ggplot2}. This offers huge advantages!

```
ggplot(belgium_shape_sf) +
  geom_sf() +
  ggtitle("Welcome to Belgium!") +
  theme_bw()
```

Have a look at Tidy spatial data in R for more details.



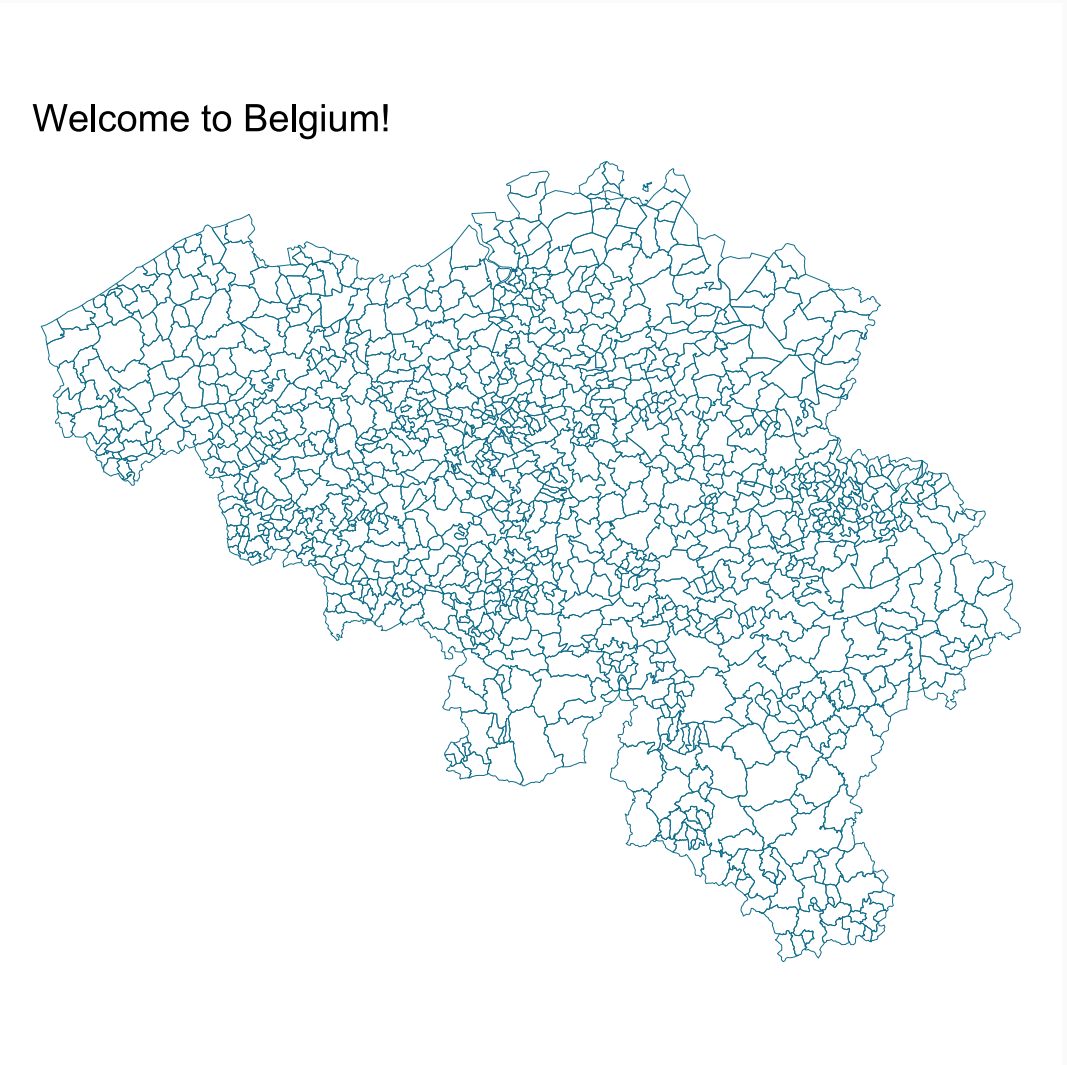
We can again build the interactive map with `mapview` from the object `belgium_shape_sf`. Give it a try!

Now we can also build nice static, thematic maps using the {tmap} package.

```
library(tmap)
# qtm(belgium_shape_sf) # does not work
# shapefile slightly corrupted!

# slightly smooth the shapefile
simple_shp ← st_simplify(belgium_shape_sf,
                         dTolerance = 0.00001)

# and plot
qtm(simple_shp)
```

We can also play with these maps

```
tm_shape(simple_shp) +
  tm_borders(col = KULbg, lwd = 0.5) +
  tm_layout(main.title = 'Welcome to Belgium!',
            legend.outside = TRUE, frame = FALSE)
```



Summary statistics at postal code level

Our goal is now to calculate summary statistics at postal code, followed by a visualisation of these summary stats using maps.

For example, what is the total exposure observed per postal code?

```
post_expo ← mtpl %>% group_by(pc) %>% summarize(num = n(), total_expo = sum(expo))

post_expo %>% slice(1:5) %>% kable(format = 'html')
```

pc	num	total_expo
1000	1171	961.3178
1030	928	738.3397
1040	433	357.2904
1050	643	524.6164
1060	344	279.9452

We recorded (non-zero) exposures for 583 of the 1,146 postal codes in Belgium.

As a first step, we take the {rgdal} approach and use the `SpatialPolygonsDataFrame` stored in `belgium_shape`.

We merge the summary stats at postal code level with the data variable stored in `belgium_shape@data`.

```
library(dplyr)
belgium_shape@data ← left_join(belgium_shape@data, post_expo, by = c("POSTCODE" = "pc"))
belgium_shape@data %>% slice(1:3) %>% kable(format = 'html')
```

POSTCODE	NAAM1	NAAM2	POSTCODELA	Shape_Leng	Shape_Area	id	num	total_expo
1300	NA	Wavre	1300	42814.06	32629722	0	477	383.87397
1301	NA	Bierges	1301	15954.49	9641834	1	NA	NA
1310	NA	La Hulpe	1310	20276.46	15381287	2	109	90.90411

and a quick check to verify our `left_join`

```
post_expo %>% filter(pc == 1301) %>% kable(format = 'html')
```

pc	num	total_expo

Next, we calculate the relative exposure per area unit.

```
belgium_shape@data$freq ← belgium_shape@data$total_expo/belgium_shape@data$Shape_Area
```

and we transform this continuous variable to a binned version using `cut()`

```
belgium_shape@data$freq_class ← cut(belgium_shape@data$freq,
                                    breaks = quantile(belgium_shape@data$freq, c(0,0.2,0.8,1), na.rm = TRUE),
                                    right = FALSE, include.lowest = TRUE, labels = c("low","average","high"))

belgium_shape@data %>% slice(1:3) %>% kable(format = 'html')
```

POSTCODE	NAAM1	NAAM2	POSTCODELA	Shape_Leng	Shape_Area	id	num	total_expo	freq	freq_class
1300	NA	Wavre	1300	42814.06	32629722	0	477	383.87397	1.18e-05	average
1301	NA	Bierges	1301	15954.49	9641834	1	NA	NA	NA	NA
1310	NA	La Hulpe	1310	20276.46	15381287	2	109	90.90411	5.90e-06	average

Finally, to use this with `ggplot` we need `fortify` to transform the original shape file to a data frame

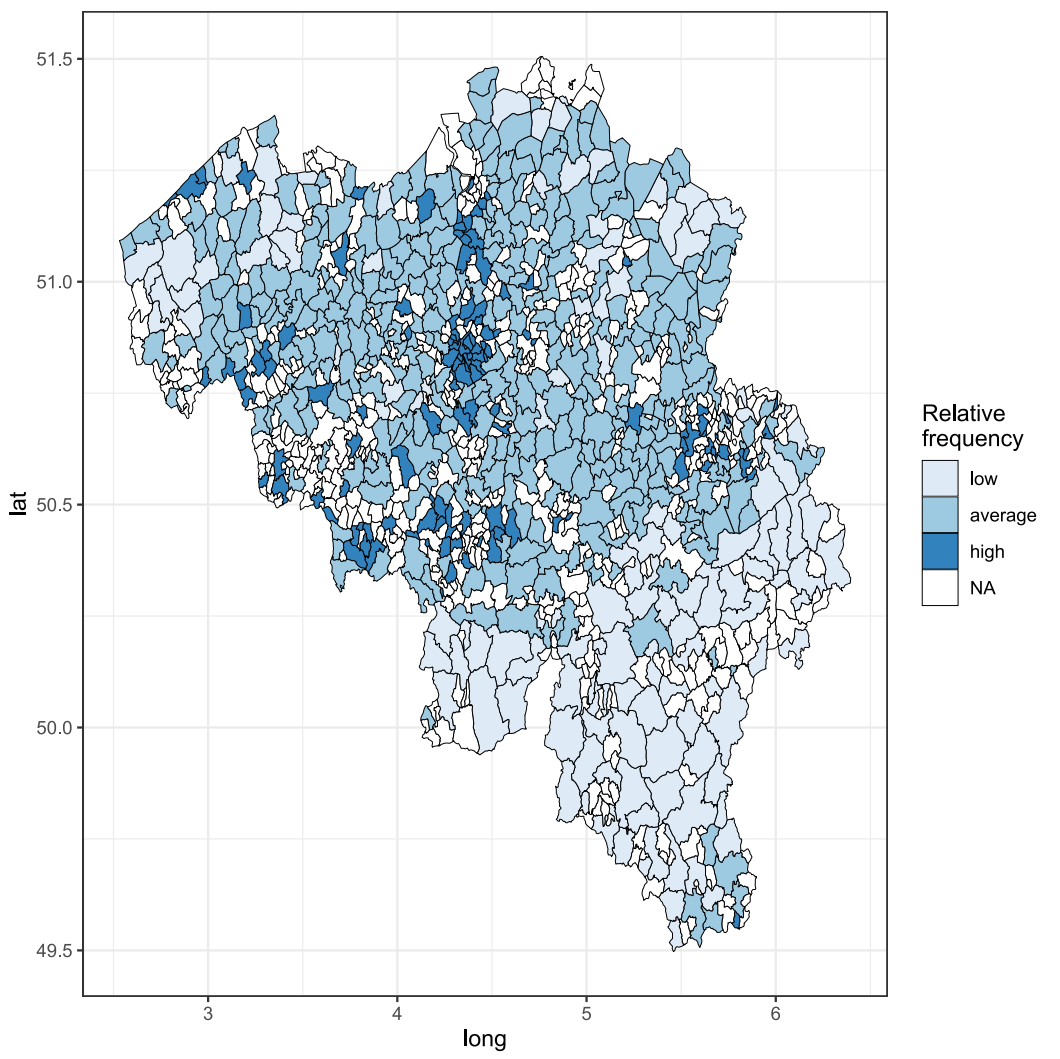
```
belgium_shape_f ← fortify(belgium_shape)
```

We merge the resulting data frame `belgium_shape_f` with `belgium_shape@data`

```
belgium_shape_f ← left_join(belgium_shape_f,
                            belgium_shape@data)
```

and we use the result to plot the map with `ggplot`

```
plot.eda.map ← ggplot(belgium_shape_f,
                      aes(long, lat, group = group)) +
  geom_polygon(aes(fill = belgium_shape_f$freq_class),
               colour = "black", size = 0.1)
plot.eda.map ← plot.eda.map + theme_bw() +
  labs(fill = "Relative\nfrequency") +
  scale_fill_brewer(palette = "Blues",
                    na.value = "white")
plot.eda.map
```



Your turn

Alternatively, you can merge the `belgium_shape_sf` object with the summary at postal code level, stored in `post_expo`.

Q: you will work through the following steps.

1. Return to the object of class `sf` and class data frame stored in `belgium_shape_sf`.
2. Merge this data frame with the `post_expo` data.
3. Calculate relative exposure per area unit, discretize this into 3 groups (low - average - high).
4. Colour each Belgian municipality according to NA - low - average - high, use `ggplot` and {tmap}.

10:00

Alternatively, we merge the `belgium_shape_sf` object with the constructed summary at postal code level, stored in `post_expo`.

```
belgium_shape_sf ← left_join(belgium_shape_sf,
                             post_expo,
                             by = c("POSTCODE" = "pc"))
```

We compute the relative exposure per area unit.

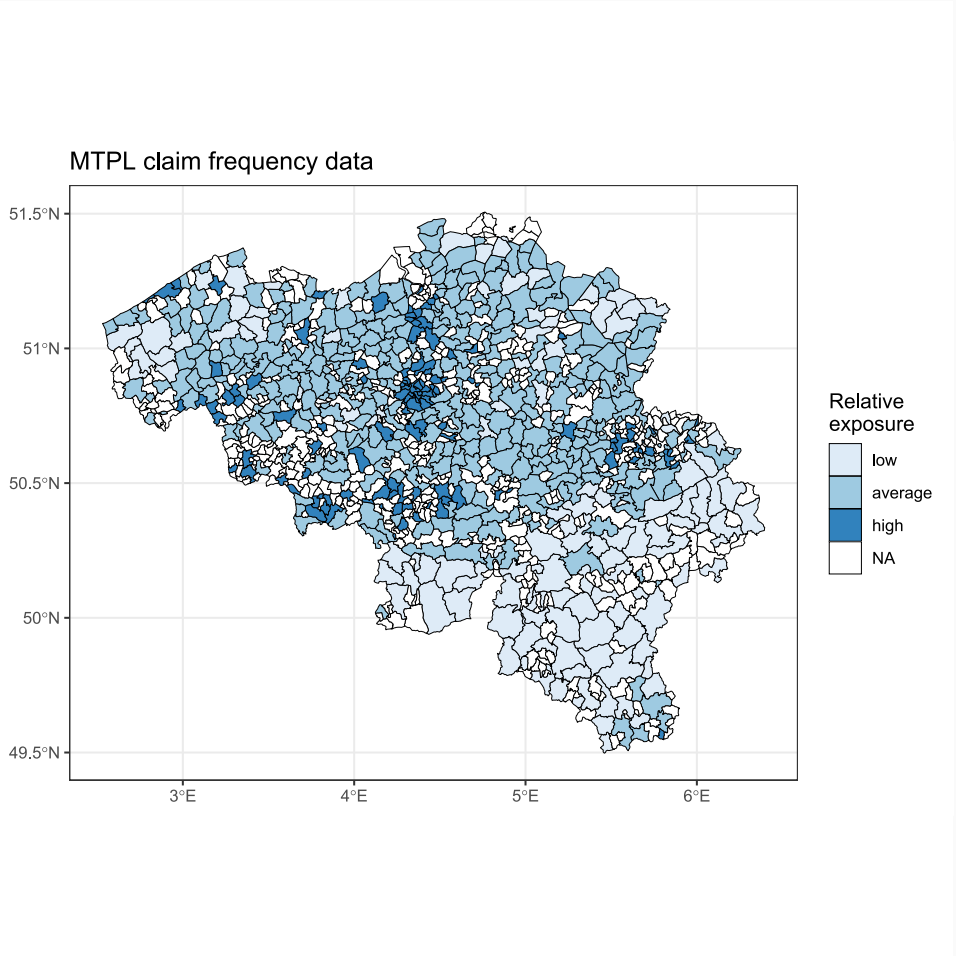
```
belgium_shape_sf$freq ←
  belgium_shape_sf$total_expo/belgium_shape_sf$Shape_Area
```

And we transform this continuous variable to a binned version using `cut()`

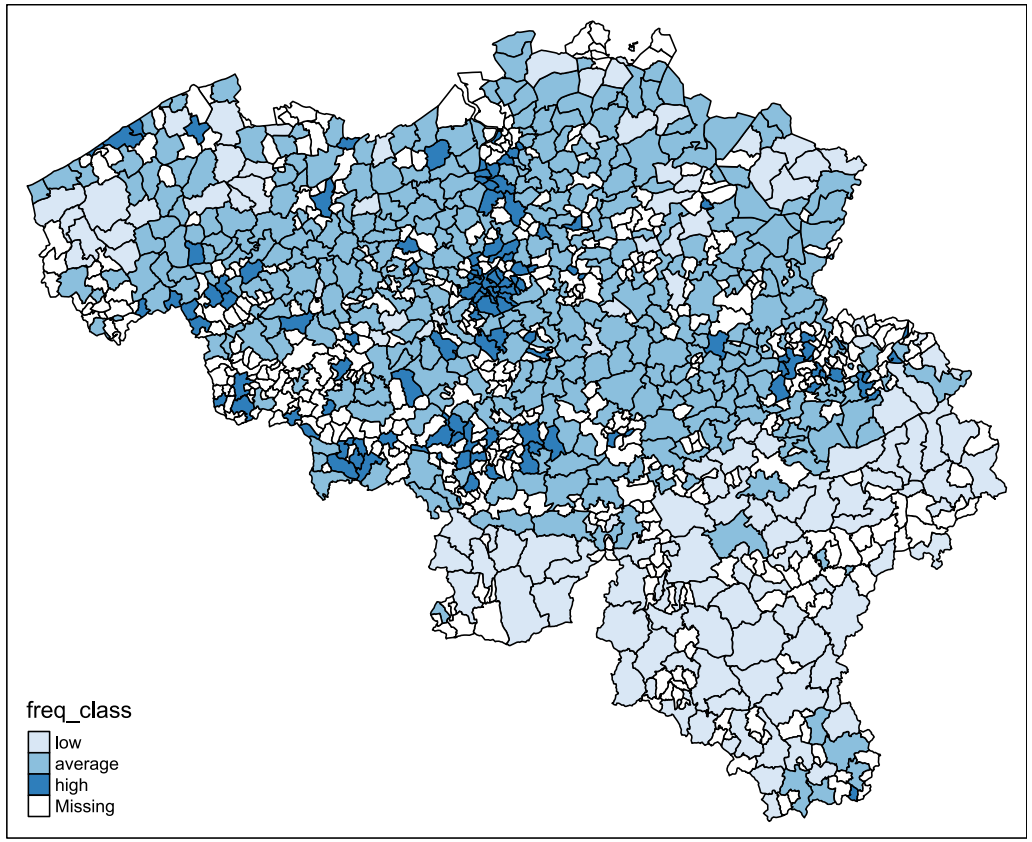
```
belgium_shape_sf$freq_class ← cut(belgium_shape_sf$freq,
                                  breaks = quantile(belgium_shape_sf$freq, c(0,0.2,0.8,1), na.rm = TRUE),
                                  right = FALSE, include.lowest = TRUE,
                                  labels = c("low", "average", "high"))
```

The resulting map can then easily be displayed with {ggplot} or with {tmap} instructions, starting from the `belgium_shape_sf` object.

```
ggplot(belgium_shape_sf) +
  geom_sf(aes(fill = freq_class),
          colour = "black", size = 0.1) +
  ggtitle("MTPL claim frequency data") +
  labs(fill = "Relative\nexposure") +
  scale_fill_brewer(palette = "Blues",
                    na.value = "white") +
  theme_bw()
```

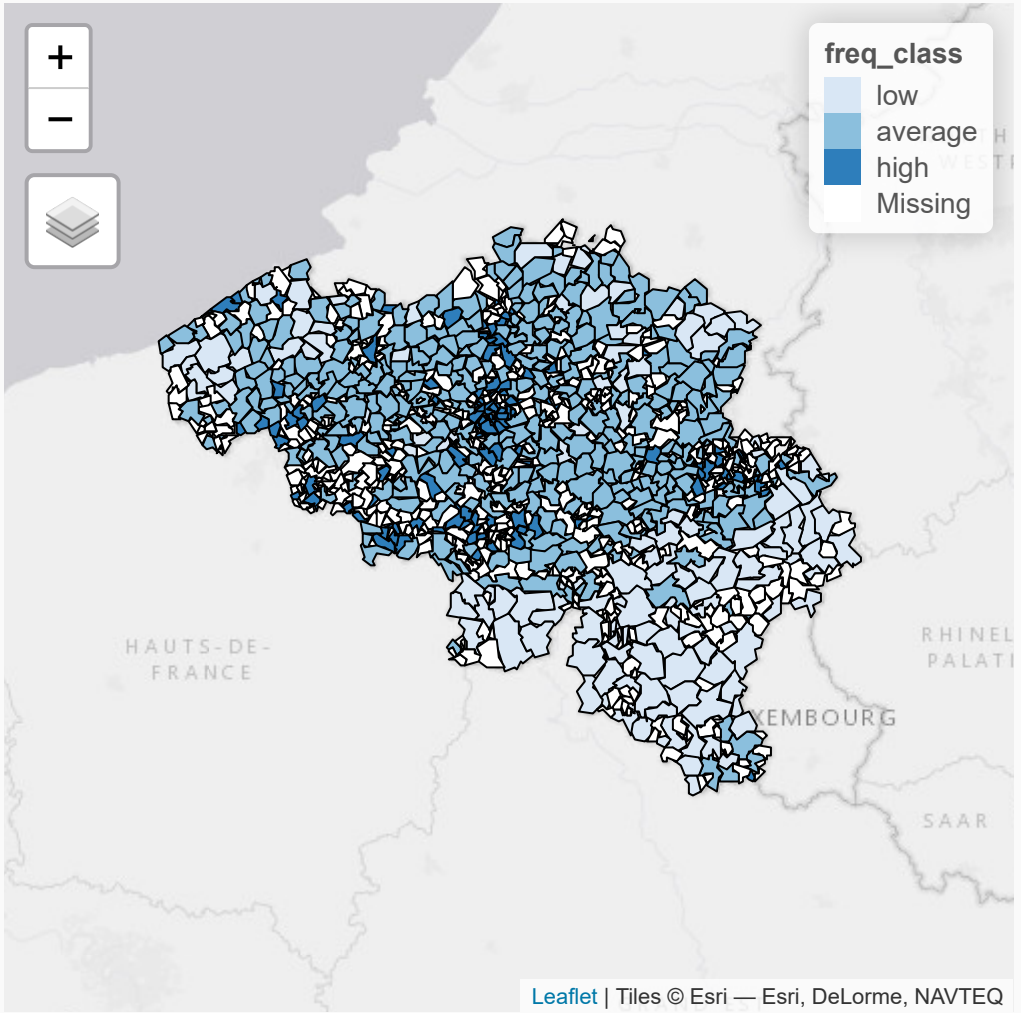


```r
library(tmap)

# slightly smooth the shapefile
belgium_shape_sf ← st_simplify(belgium_shape_sf,
                          dTolerance = 0.00001)

# and plot
tm_shape(belgium_shape_sf) +
  tm_borders(col = "black") +
  tm_fill(col = "freq_class",
          style = "cont", palette = "Blues",
          colorNA = "white")
```



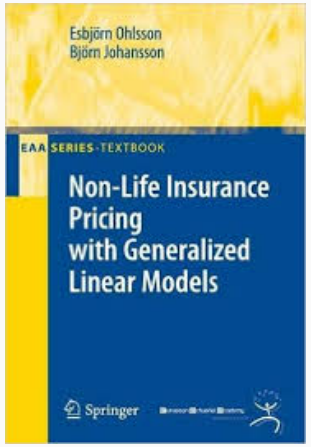
Finally, for interactive maps we can use e.g.

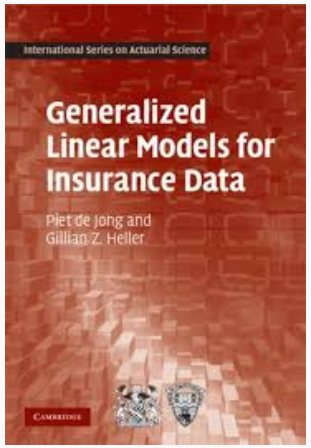
```
tmap_leaflet(last_map())
```



Model building with GLMs and GAMs

Linear and Generalized Linear Models





With **linear regression models** `lm(.)`

- model specification

$$Y = x^{'}\beta + \epsilon.$$

- ϵ is normally distributed with mean 0 and common variance σ^2, thus: Y is normal with mean $x^{'}\beta$ and variance σ^2

With **generalized linear regression models** `glm(.)`

- model specification

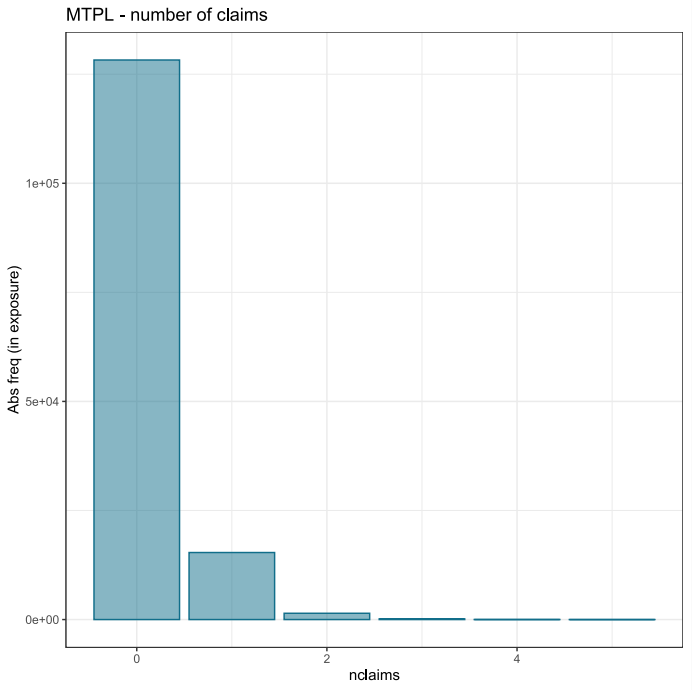
$$g(E[Y]) = x^{'}\beta.$$

- $g(.)$ is the link function
- Y follows a distribution from the exponential family.

Generalized Linear Models (GLMs)

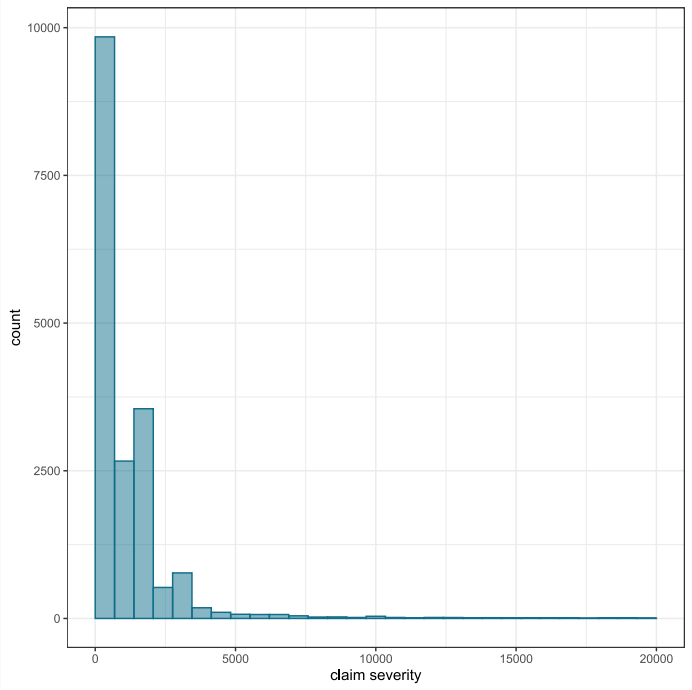
We return to the `mtpl` data set.

Target variable `nclaims` (frequency)



Suitable distributions: Poisson, Negative Binomial.

... and `avg` (severity).



Suitable distributions: log-normal, gamma.

A Poisson GLM

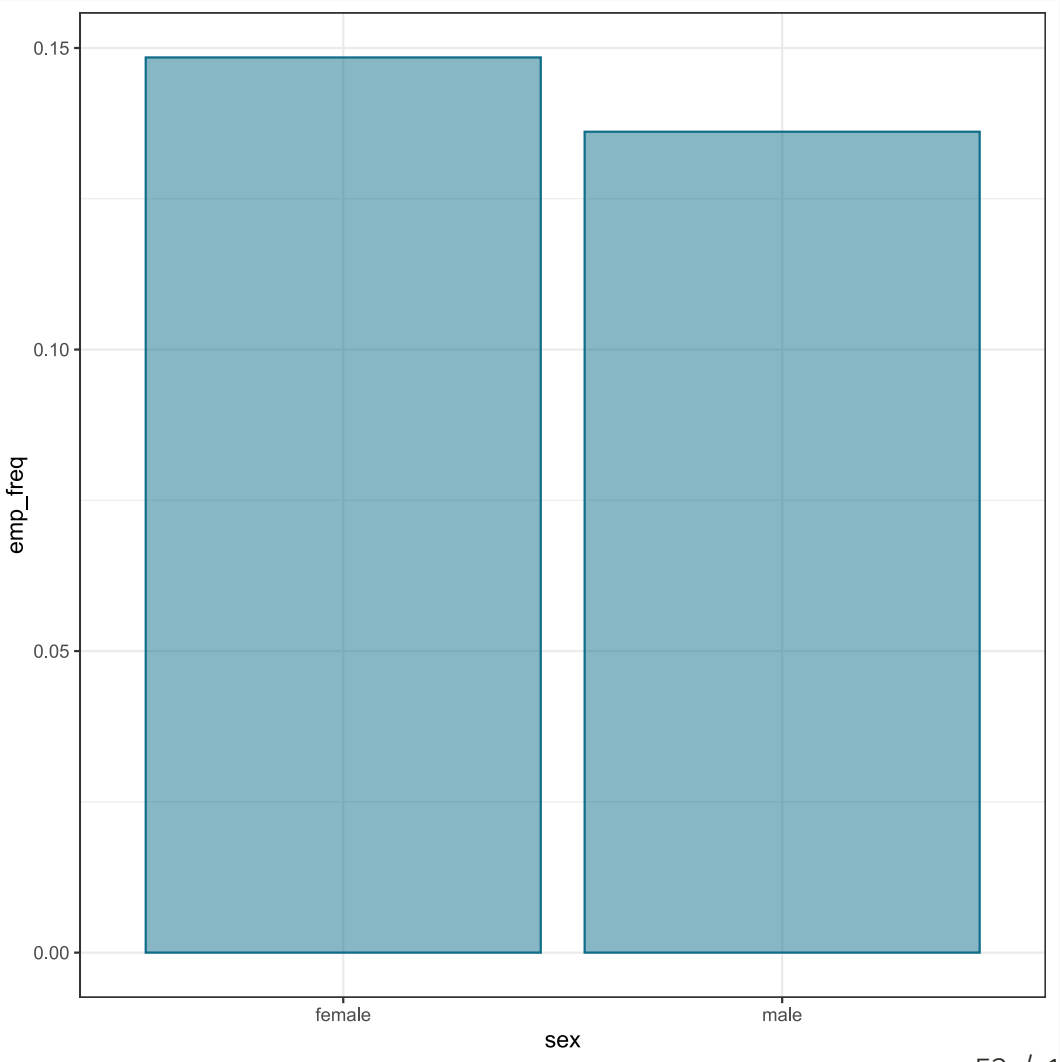
A brief recap..

```
freq_by_gender ← mtpl %>%
  group_by(sex) %>%
  summarize(emp_freq = sum(nclaims) / sum(expo))
```

sex	emp_freq
female	0.1484325
male	0.1361164

Let's picture the empirical gender-specific claim frequency...

```
ggplot(freq_by_gender, aes(x = sex, y = emp_freq)) +
  geom_bar(col = KULbg, fill = KULbg, alpha = .5)
```



A Poisson GLM (cont.)

```
freq_glm_1 ← glm(nclaims ~ sex, offset = log(expo),
                 family = poisson(link = "log"),
                 data = mtpl)
```

Fit a Poisson GLM, with logarithmic link function.

This implies:

Y ~ Poisson, with

$$\log(E[Y]) = x^{'}\beta,$$

or,

$$E[Y] = \exp\left(x^{'}\beta\right).$$

Fit this model on `data = mtpl`.

A Poisson GLM (cont.)

```
freq_glm_1 ← glm(nclaims ~ sex, offset = log(expo),
                 family = poisson(link = "log"),
                 data = mtpl)
```

Use `nclaims` as Y.

Use `gender` as the only (factor) variable in the linear predictor.

Include `log(exp)` as an offset term in the linear predictor.

Then,

$$x'\beta = \log(\mathbf{expo}) + \beta_0 + \beta_1 \mathbb{I}(\mathbf{male}).$$

Put otherwise,

$$E[Y] = \mathbf{expo} \cdot \exp\left(\beta_0 + \beta_1 \mathbb{I}(\mathbf{male})\right),$$

where **expo** refers to `expo` the exposure variable.

```
freq_glm_1 ← glm(nclaims ~ sex, offset = log(expo),
                 family = poisson(link = "log"),
                 data = mtpl)
```

```
freq_glm_1 %>% broom::tidy()
```

term	estimate	std.error	statistic	p.value
(Intercept)	-1.9076251	0.0133227	-143.186324	0
sexmale	-0.0866198	0.0156837	-5.522931	0

Mind the specification of `type.predict` when using `augment` with a GLM!

```
freq_glm_1 %>% broom::augment(type.predict =
                                "response")
```

nclaims	sex	.fitted	.se.fit
1	male	0.1361164	0.0011264
0	female	0.1484325	0.0019775

The `predict` function of a GLM object offers 3 options: `"link"`, `"response"` or `"terms"`.

The same options hold when `augment()` is applied to a GLM object.

Let's see how the fitted values at `"response"` level are constructed:

```
exp(coef(freq_glm_1)[1])
## (Intercept)
##   0.1484325
exp(coef(freq_glm_1)[1] + coef(freq_glm_1)[2])
## (Intercept)
##   0.1361164
```

Do you recognize these numbers?

Last step:

try `freq_glm_1 %>% glance()` or `summary(freq_glm_1)` for deviances.

Your turn

You will further explore GLMs in R with the `glm(.)` function.

Q: continue with the `freq_glm_1` object that was created, you will now explicitly call the `predict()` function on this object.

1. Verify the arguments of `predict.glm` using `? predict.glm`.

2. The help reveals the following structure `predict(.object, .newdata, type = (" ... "))` where `.object` is the fitted GLM object, `.newdata` is (optionally) a data frame to look for the features used in the model, and `type` is `"link"`, `"response"` or `"terms"`.
   Use `predict` with `freq_glm_1` and a newly created data frame.
   Explore the different options for `type`, and their connections.

3. Fit a gamma GLM for `avg` (the claim severity) with log link.
   Use `sex` as the only variable in the model. What do you conclude?

Q.1 You can access the documentation via `? predict.glm`.

Q.2 You create new data frames (or tibbles) as follows

```
male_driver ← data.frame(exp = 1, sex = "male")
female_driver ← data.frame(exp = 1, sex = "female")
```

Next, you apply `predict` with the GLM object `freq_glm_1` and one of these data frames, e.g.

```
predict(freq_glm_1, newdata = male_driver,
                        type = "response")
```

```
##         1
## 0.1361164
```

Q.2 Next, you apply `predict` with the GLM object `freq_glm_1` and one of these data frames, e.g.

```
predict(freq_glm_1, newdata = male_driver,
                            type = "response")
```

```
##         1
## 0.1361164
```

At the level of the linear predictor:

```
predict(freq_glm_1, newdata = male_driver,
                            type = "link")
```

```
##         1
## -1.994245
```

```
exp(predict(freq_glm_1, newdata = male_driver,
                            type = "link"))
```

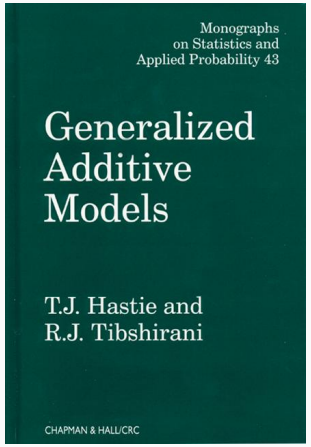
```
##         1
## 0.1361164
```

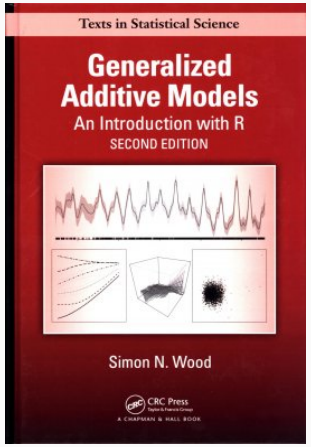
Q.3 For the gamma regression model

```
sev_glm_1 ← glm(avg ~ sex, family = Gamma(link = "log"), data = mtpl)
sev_glm_1
```

```
##
## Call:  glm(formula = avg ~ sex, family = Gamma(link = "log"), data = mtpl)
##
## Coefficients:
## (Intercept)      sexmale
##      7.5730      -0.2581
##
## Degrees of Freedom: 18294 Total (i.e. Null);  18293 Residual
##   (144936 observations deleted due to missingness)
## Null Deviance:        46690
## Residual Deviance: 46440     AIC: 299700
```

Generalized Additive Models (GAMs)





With **GLMs** `glm(.)`

- transformation of the mean modelled with a linear predictor

$$x'\beta$$

- not well suited for continuous risk factors that relate to the response in a non-linear way.

With **Generalized Additive Models (GAMs)**

- the predictor allows for smooth effects of continuous risk factors and spatial covariates, next to the linear terms, e.g.

$$x'\beta + \sum_j f_j(x_j) + f(\texttt{lat}, \texttt{long})$$

- predictor is still additive
- preferred R package is {mgcv} by Simon Wood.

More on GAMs

So, a GAM is a GLM where the linear predictor depends on **smooth functions** of covariates.

Consider a GAM with the following predictor:

$$x^{'}\beta + f_j(x_j).$$

GAMs use **basis functions** to estimate the smooth effect $f_j(.)$

$$f_j(x_j) = \sum_{m=1}^{M} \beta_{jm} b_{jm}(x_j),$$

where the $b_{jm}(x)$ are known basis functions and β_{jm} are coefficients that have to be estimated.

GAMs avoid overfitting by adding a **wiggliness penalty** to the likelihood

$$\int \left(f_j(x)^{''}\right)^2 = \beta_j^t \mathbf{S}_j \beta_j.$$

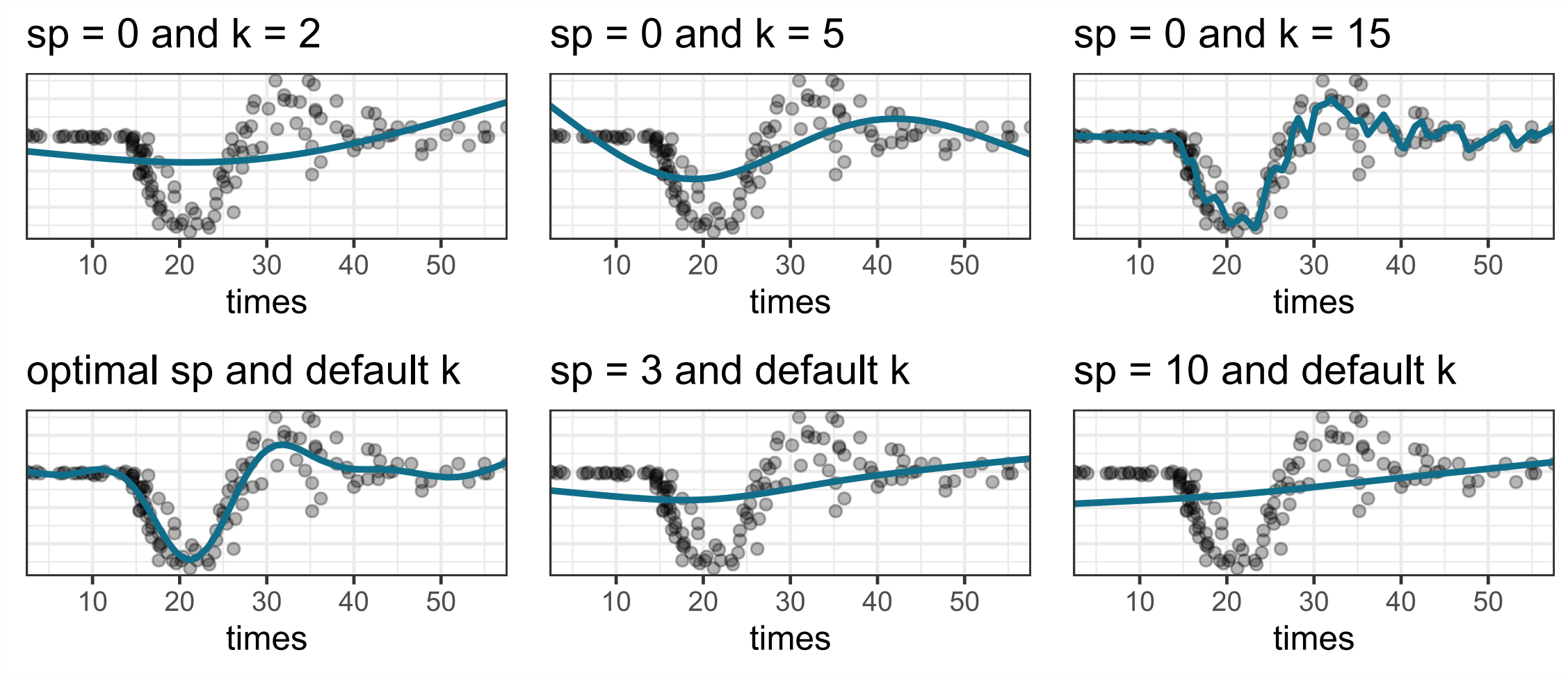
GAMs then balance goodness-of-fit and wiggliness via

$$\log \mathcal{L}(\beta, \beta_j) - \lambda_j \cdot \beta_j^t \mathbf{S}_j \beta_j,$$

with λ_j the **smoothing parameter**.

The smoothing parameter λ_j controls the trade-off between fit & smoothness.

Let's run some experiments to illustrate the effect of the smoothing parameter (`sp = .`), the number (`k = .`) and type of basis functions (`bs = .`). We use the `mcycle` data from {MASS}.



Fitting a GAM with {mgcv}

Fitting a GAM with {mgcv} - the Mixed GAM Computation Vehicle with automatic smoothness estimation - essentially goes as follows:

```
model ← gam(accel ~ s(times, sp = 1.2,
                      k = 5, bs = "cr"),
            family = gaussian, data = mcycle)
```

Include a smooth effect of `times` via `s(times)`.

`sp = .` specifies a value for the smoothing parameter.

`k = .` fixes the number of basis functions.

`bs = "cr"` indicates which type of basis functions should be used. Here `"cr"` refers to the cubic spline basis.

More options and details via https://stat.ethz.ch/R-manual/R-devel/library/mgcv/html/smooth.terms.html.

Fitting a GAM with {mgcv}

Fitting a GAM with {mgcv} - the Mixed GAM Computation Vehicle with automatic smoothness estimation - essentially goes as follows:

```
model ← gam(accel ~ s(times, sp = 1.2,
                      k = 5, bs = "cr"),
            family = gaussian, data = mcycle)
```

or like this

```
model ← gam(accel ~ s(times, bs = "cr"),
            method = "REML",
            family = gaussian, data = mcycle)
```

Include a smooth effect of `times` via `s(times)`.

`sp = .` specifies a value for the smoothing parameter.

`k = .` fixes the number of basis functions.

`bs = "cr"` indicates which type of basis functions should be used. Here `"cr"` refers to the cubic spline basis.

More options and details via https://stat.ethz.ch/R-manual/R-devel/library/mgcv/html/smooth.terms.html.

`method = "REML"` will determine an optimal value for the smoothing parameter, using the REML method.

More details via https://www.rdocumentation.org/packages/mgcv/versions/1.8-31/topics/gam.

Fitting a GAM with {mgcv} (cont.)

We then inspect the fitted model via

```
print(model)
```

and access the smoothing parameter with

```
model$sp
```

A visualization of the fitted smoothers is available as

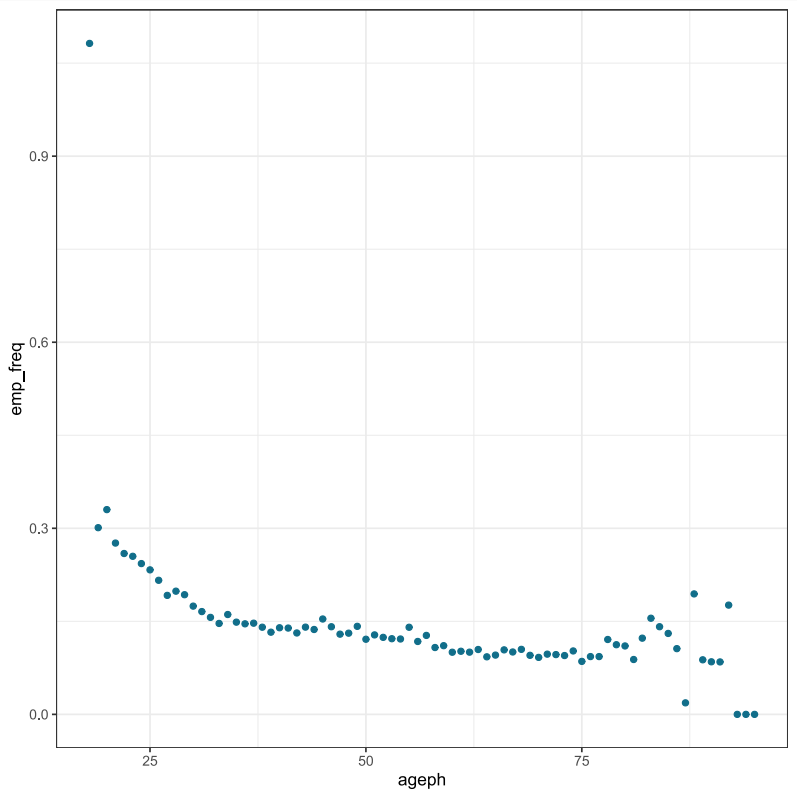
```
plot(model, pages = 1, scheme = 0)
plot(model, pages = 1, scheme = 1)
```

where `pages` indicates the number of pages over which to spread the output (with `0` as the default), `scheme = .` specifies the built-in plotting scheme that should be used.

More details via https://www.rdocumentation.org/packages/mgcv/versions/1.8-31/topics/plot.gam.

A Poisson GAM

We continue working with `mtpl` and now focus on `ageph`.



We will now explore **four different model specifications**:

1. `ageph` as linear effect in `glm`
2. `ageph` as factor variable in `glm`
3. `ageph` split manually into bins using `cut`, then used as factor in `glm`
4. a smooth effect of `ageph` in `mgcv::gam`.

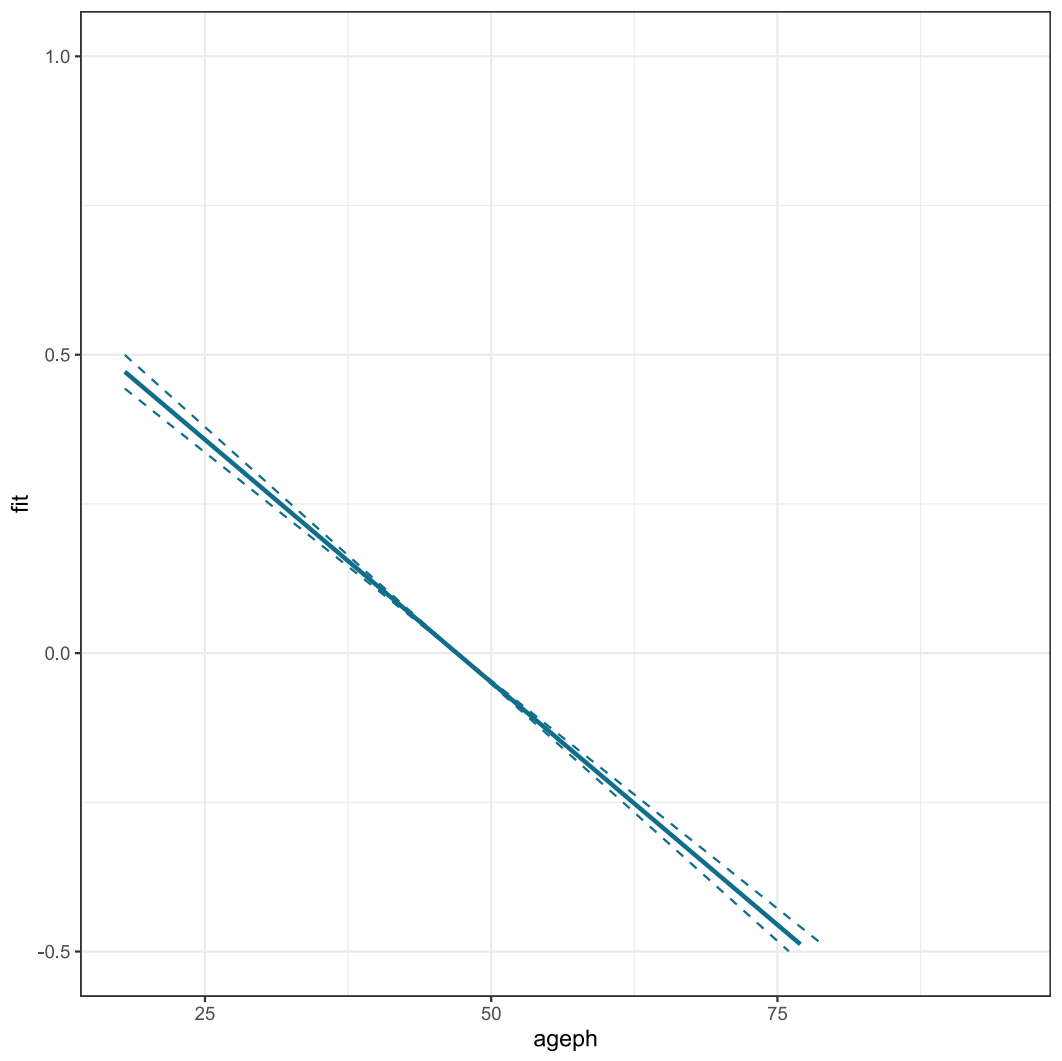
Let's go!

Grid of observed `ageph` values

```
a ← min(mtpl$ageph):max(mtpl$ageph)
```

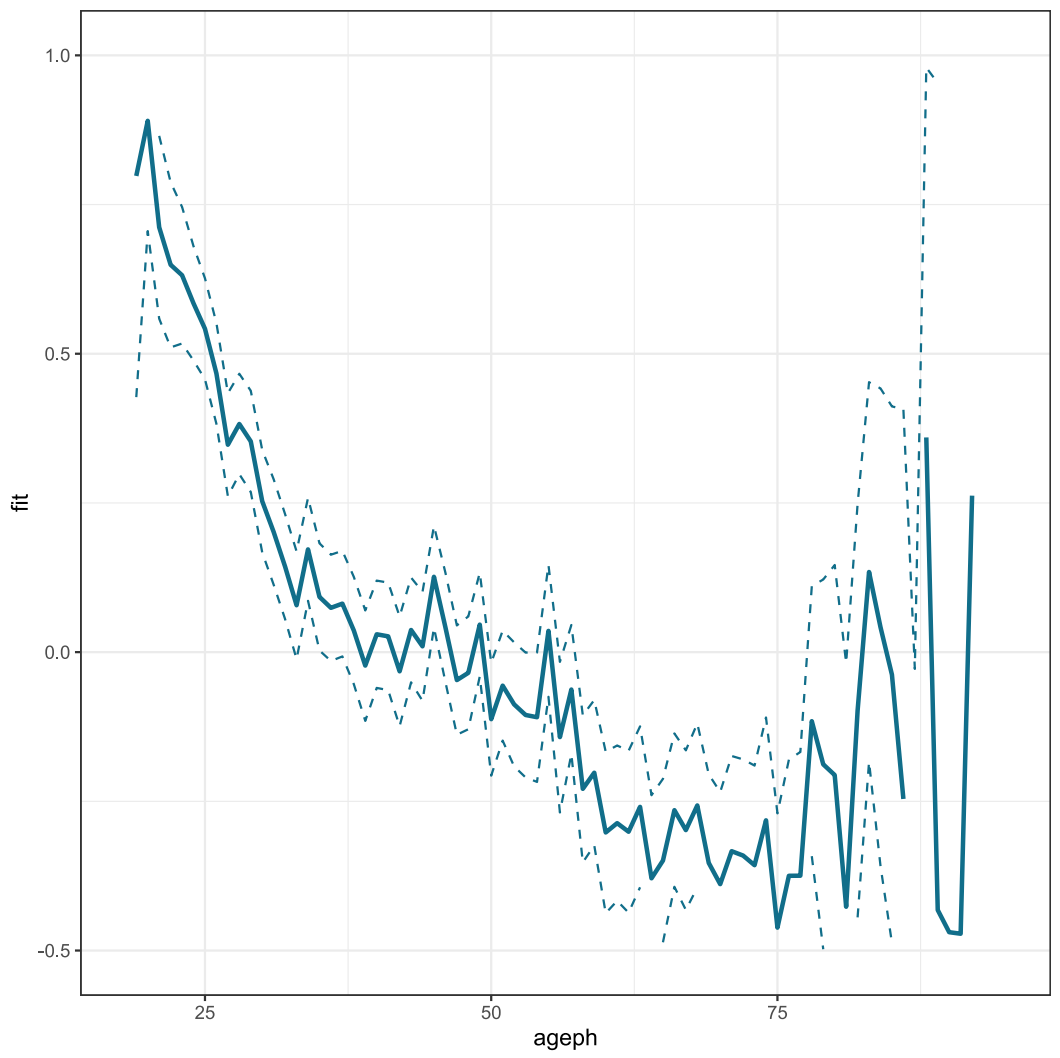
Model 1: linear effect of `ageph`

```
freq_glm_age ← glm(nclaims ~ ageph,
                   offset = log(expo),
                   data = mtpl,
                   family = poisson(link = "log"))
pred_glm_age ← predict(freq_glm_age,
          newdata = data.frame(ageph = a, expo = 1),
          type = "terms", se.fit = TRUE)
b_glm_age ← pred_glm_age$fit
l_glm_age ← pred_glm_age$fit
                 - qnorm(0.975)*pred_glm_age$se.fit
u_glm_age ← pred_glm_age$fit
                 + qnorm(0.975)*pred_glm_age$se.fit
df ← data.frame(a, b_glm_age, l_glm_age, u_glm_age)
```



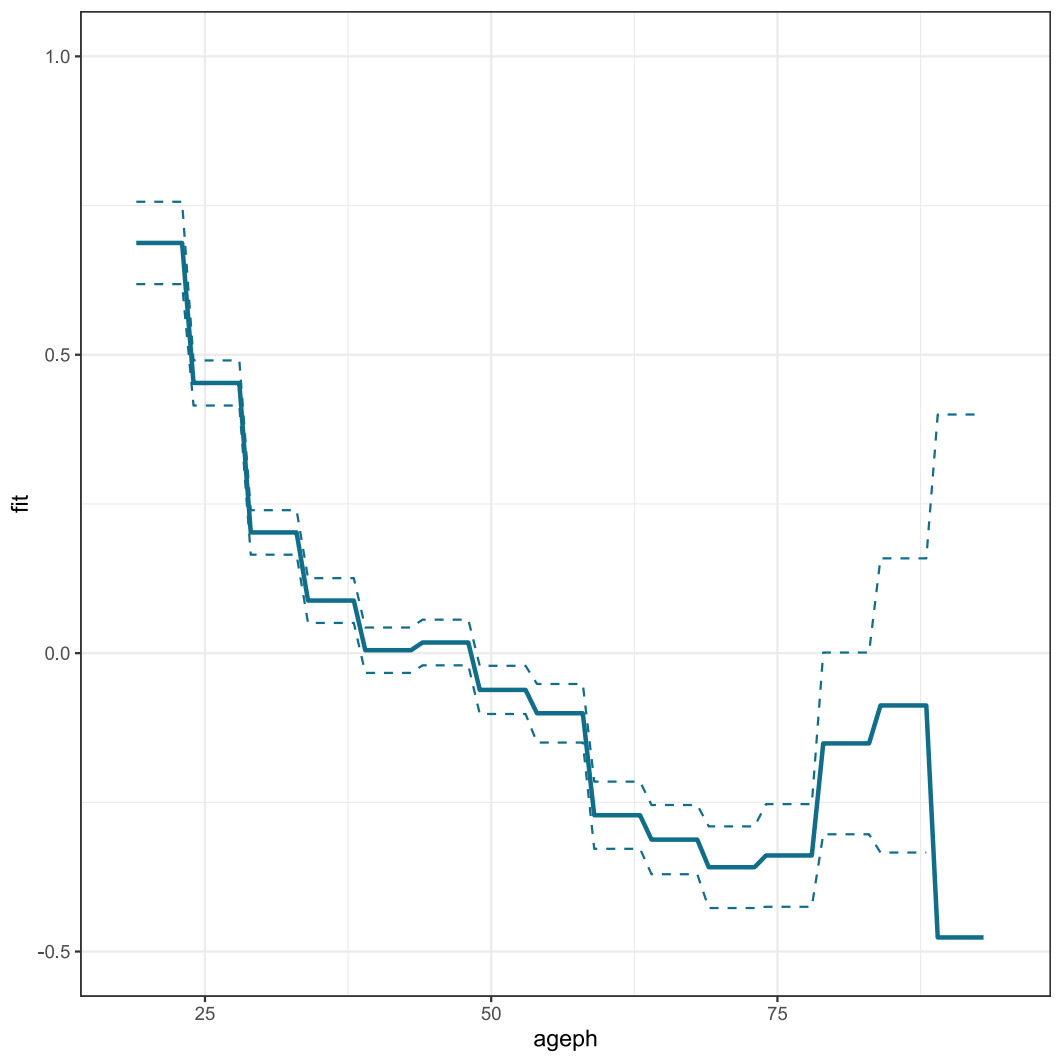
Model 2: `ageph` as factor variable in `glm`

```
freq_glm_age_f ← glm(nclaims ~ as.factor(ageph),
                     offset = log(expo),
                     data = mtpl,
                     family = poisson(link = "log"))
pred_glm_age_f ← predict(freq_glm_age_f,
           newdata = data.frame(ageph = a, expo = 1),
           type = "terms", se.fit = TRUE)
b_glm_age_f ← pred_glm_age_f$fit
l_glm_age_f ← pred_glm_age_f$fit
                   - qnorm(0.975)*pred_glm_age_f$se.fit
u_glm_age_f ← pred_glm_age_f$fit
                   + qnorm(0.975)*pred_glm_age_f$se.fit
df ← data.frame(a, b_glm_age_f,
                     l_glm_age_f, u_glm_age_f)
```



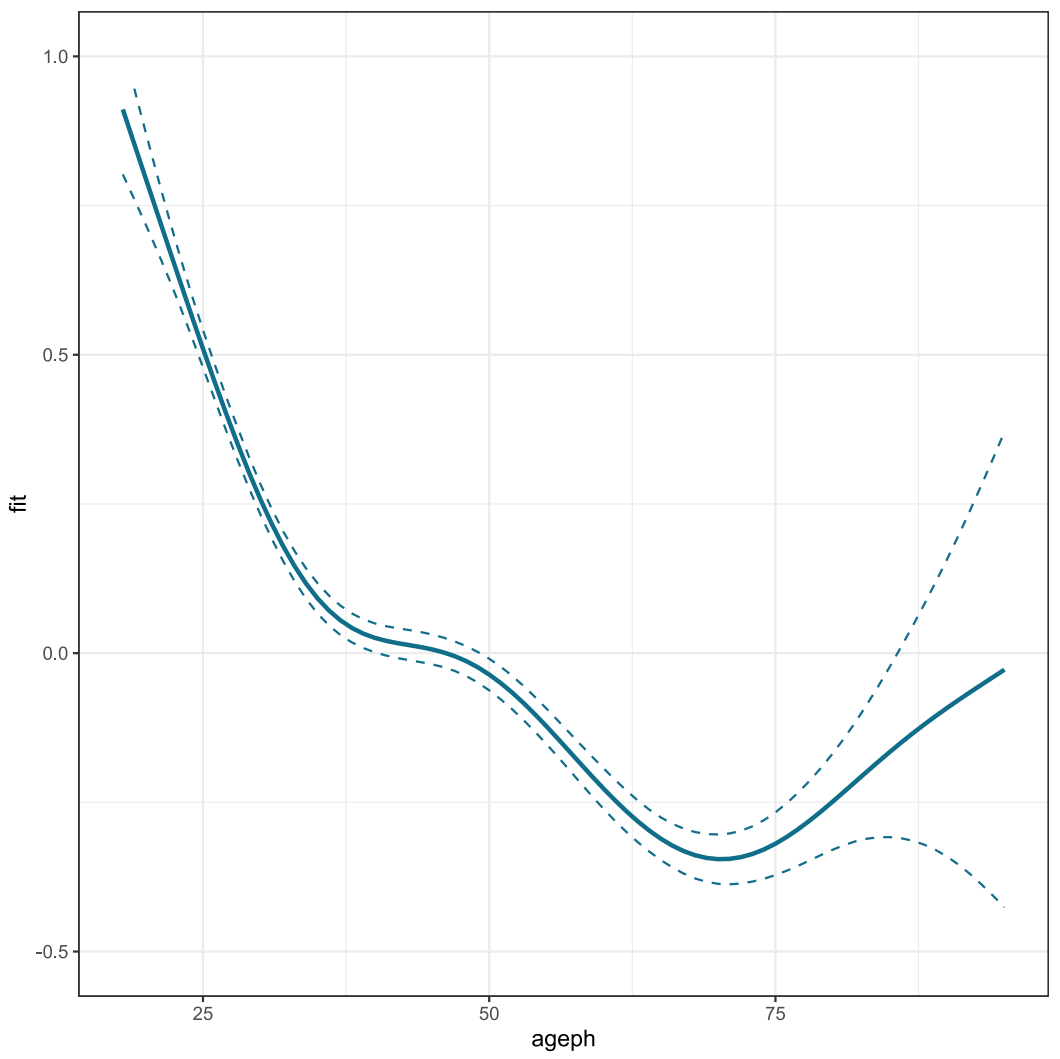
Model 3: `ageph` split into 5-year bins and then used in `glm`

```
level ← seq(min(mtpl$ageph), max(mtpl$ageph), by = 5)
freq_glm_age_c ← glm(nclaims ~ cut(ageph, level),
                     offset = log(expo),
                     data = mtpl,
                     family = poisson(link = "log"))
pred_glm_age_c ← predict(freq_glm_age_c,
          newdata = data.frame(ageph = a, expo = 1),
          type = "terms", se.fit = TRUE)
b_glm_age_c ← pred_glm_age_c$fit
l_glm_age_c ← pred_glm_age_c$fit
                   - qnorm(0.975)*pred_glm_age_c$se.fit
u_glm_age_c ← pred_glm_age_c$fit
                   + qnorm(0.975)*pred_glm_age_c$se.fit
df ← data.frame(a, b_glm_age_c,
                     l_glm_age_c, u_glm_age_c)
```



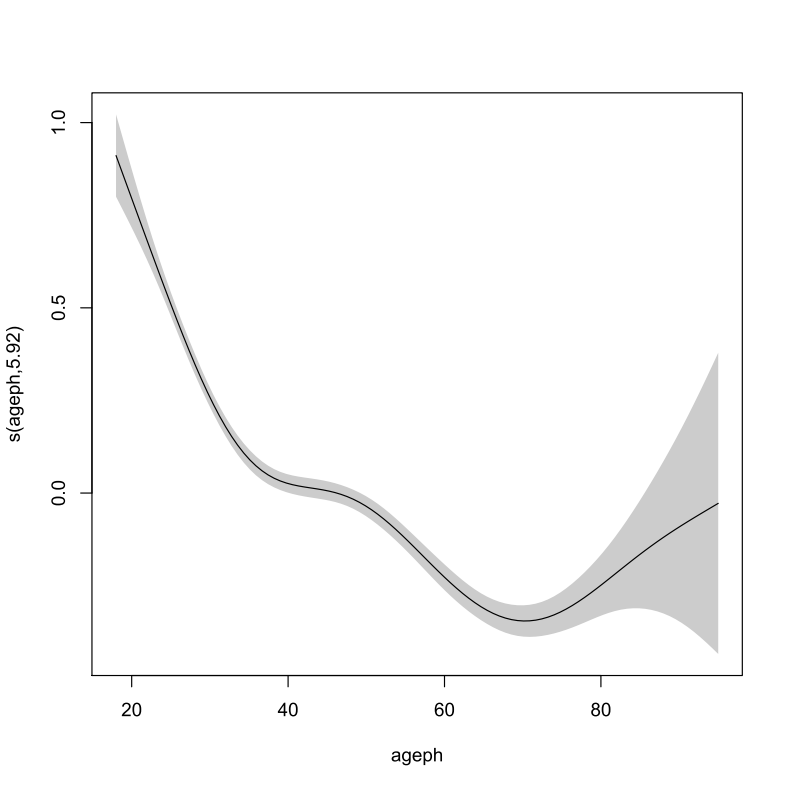
Model 4: smooth effect of `ageph` in `mgcv::gam`

```
library(mgcv)
freq_gam_age ← gam(nclaims ~ s(ageph),
                   offset = log(expo),
                   data = mtpl,
                   family = poisson(link = "log"))
pred_gam_age ← predict(freq_gam_age,
          newdata = data.frame(ageph = a, expo = 1),
          type = "terms", se.fit = TRUE)
b_gam_age ← pred_gam_age$fit
l_gam_age ← pred_gam_age$fit -
                  qnorm(0.975)*pred_gam_age$se.fit
u_gam_age ← pred_gam_age$fit +
                  qnorm(0.975)*pred_gam_age$se.fit
df ← data.frame(a, b_gam_age,
                   l_gam_age, u_gam_age)
```



Model 4 (revisited): picture smooth effect of `ageph` in `mgcv::gam` with built-in `plot`.

```
library(mgcv)
freq_gam ← gam(nclaims ~ s(ageph), offset = log(expo), family = poisson(link = "log"), data = mtpl)
plot(freq_gam, scheme = 1)
```



Your turn

You will further explore GAMs in R with the `gam(.)` function from the {mgcv} package.

Q: you will combine insights from building `glm` as well as `gam` objects by working through the following coding steps.

1. Fit a `gam` including some factor variables as well as a smooth effect of `ageph` and `bm`. Visualize the fitted smooth effects.

2. Specify risk profiles of drivers. Calculate their expected annual claim frequency from the constructed `gam`.

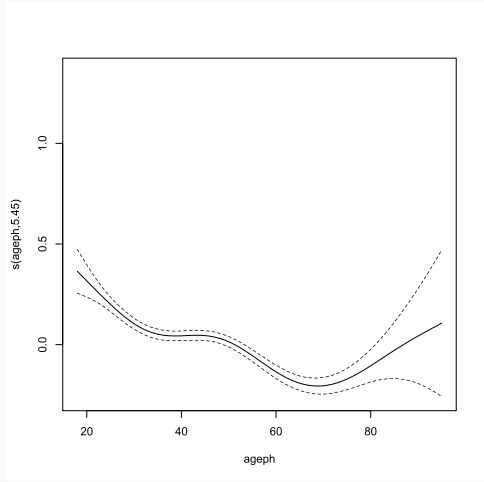
3. Explain (in words) which profiles would represent high vs low risk according to the constructed model.

Q.1 examine the following `gam` fit

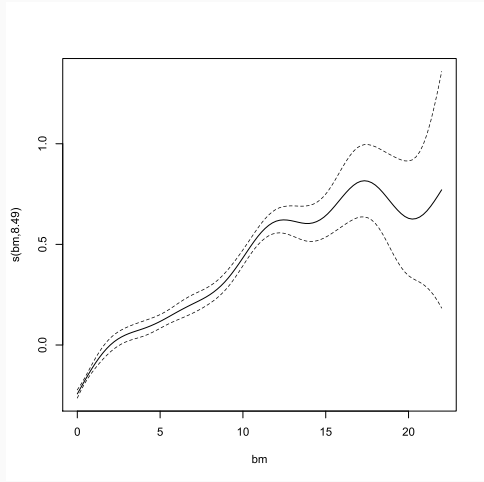
```
freq_gam_2 ← gam(nclaims ~ sex + fuel + use +
                           s(ageph) + s(bm),
                 offset = log(expo),
                 family = poisson(link = "log"),
                 data = mtpl)
```

```
summary(freq_gam_2)
##
## Family: poisson
## Link function: log
##
## Formula:
## nclaims ~ sex + fuel + use + s(ageph) + s(bm)
##
## Parametric coefficients:
##              Estimate Std. Error  z value Pr(>|z|)
## (Intercept)  -1.917801   0.018124 -105.818   <2e-16
## sexmale       0.009177   0.016043    0.572   0.5673
## fuelgasoline -0.152756   0.015100  -10.116   <2e-16
## usework      -0.055156   0.033092   -1.667   0.0956
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.
##
```

```
plot(freq_gam_2, select = 1)
```




```
plot(freq_gam_2, select = 2)
```



Q.2 define some risk profiles

```
drivers ← data.frame(expo = c(1, 1, 1),
                     sex = c("female", "female", "fema
                     fuel = c("diesel", "diesel", "die
                     use = c("private", "private", "pr
                     ageph = c(18, 45, 65), bm = c(20,
drivers
```

expo	sex	fuel	use	ageph	bm
1	female	diesel	private	18	20
1	female	diesel	private	45	5
1	female	diesel	private	65	0

Now, you predict the annual expected claim frequency for these profiles.

```
predict(freq_gam_2, newdata = drivers,
        type = "response")
```

x
0.3969310
0.1727430
0.0951124

Smooth functions of several variables

We now focus on estimating smoothers of two input variables.

We first put focus on estimating $f(x, y)$ via thin plate splines.

This is an example of a so-called isotropic smooth, used when x and y are at the same scale.

These represent $f(x, y)$ as $\sum_{n=1}^{N} \gamma_{jn} \cdot \tilde{b}_{jn}(x, y)$.

To avoid overfitting a wiggliness penalty is added to the likelihood:

$$\int\int \left(\frac{\partial^2 f_j}{(\partial x)^2}\right)^2 + 2\left(\frac{\partial^2 f_j}{\partial x \partial y}\right)^2 + \left(\frac{\partial^2 f_j}{(\partial y)^2}\right)^2 dxdy$$
$$= \gamma_j^t \boldsymbol{T}_j \gamma_j.$$

In {mgcv} we fit the spatial effect using a thin plate spline of `lat` and `long`.

```
freq_gam_spatial ← gam(nclaims ~ s(long, lat,
                                   bs = "tp"),
                       offset = log(expo),
                       family =
                         poisson(link = "log"),
                       data = mtpl)
freq_gam_spatial$sp
## s(long,lat)
##    0.858939
```

The output shows that one smoothing parameter is used.

The built-in plotting function `plot(freq_gam_spatial, scheme = 2)` reveals the structure of Belgium (a bit).

Smooth functions of several variables (cont.)

Tensor product smooths are typically used when variables are measured on different scale.

We start by choosing marginal bases and penalties, as if constructing 1-D smooths of x and y:

$$f(x) = \sum \alpha_i a_i(x), \qquad f(z) = \sum \beta_j b_j(z),$$

with the usual penalties.

The tensor product basis construction then gives:

$$f(x, y) = \sum \sum \beta_{ij} b_j(z) a_i(x),$$

with double penalties and **two smoothing parameters**.

We include an interaction effect (via `ti(., .)`) next to the individual, univariate effects.

```
freq_gam_inter ← gam(nclaims ~ s(ageph) + s(power) +
                       ti(ageph, power, bs = "tp"),
                       offset = log(expo),
                       family = poisson(link = "log"),
                       data = mtpl)
freq_gam_inter$sp
##         s(ageph)         s(power) ti(ageph,power)1 t
##       1.73015996       0.08216707     654.63175948
```

The built-in plotting function `plot(freq_gam_inter, scheme = 2, select = 3)` reveals the structure of the fitted interaction effect.

Visualizing the fitted spatial effect

To improve the visualisation of the fitted spatial effect, we extract the fitted `s(lat, long)` for each postal code stored in the `sf` object `belgium_shape_sf`.

First, we retrieve the coordinates of the postal code centroids, using `st_centroid(.)` applied to an `sf` object.

```
post_dt ← st_centroid(belgium_shape_sf)
post_dt$long ← do.call(rbind, post_dt$geometry)[,1]
post_dt$lat ← do.call(rbind, post_dt$geometry)[,2]
```

Next, we evaluate the spatial smoother stored in `freq_gam_spatial` in each of the postal codes, using the `predict(.)` function of the GAM.

```
pred ← predict(freq_gam_spatial, newdata = post_dt,
               type = "terms", terms = "s(long,lat)")
```

```
dt_pred ← data.frame(pc = post_dt$POSTCODE,
                     long = post_dt$long,
                     lat = post_dt$lat, pred)
names(dt_pred)[4] ← "fit_spatial"
```

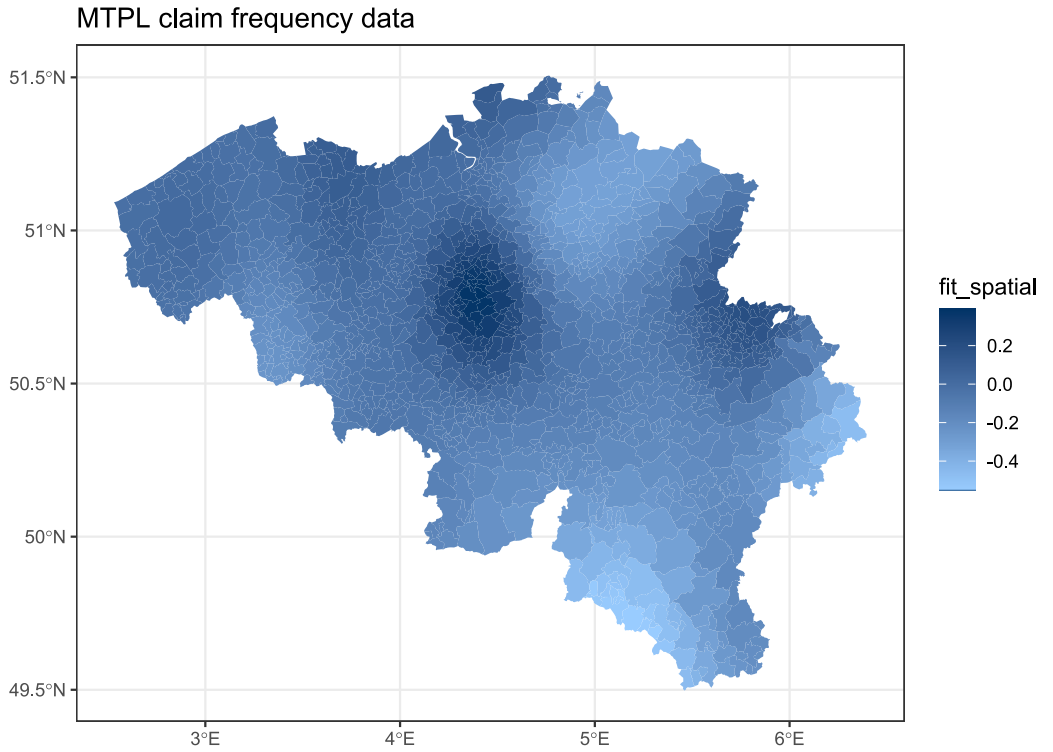
Finally, we merge `dt_pred` and `belgium_shape_sf`.

```
belgium_shape_sf ← left_join(belgium_shape_sf,
                             dt_pred,
                             by = c("POSTCODE" =
                                    "pc"))
```

We plot the fitted spatial effect on a map, using the tools discussed earlier on.

Mind the `scale_fill_gradient(.)` to be used with a continuous variable for `fill = .`.

```
ggplot(belgium_shape_sf) +
  geom_sf(aes(fill = fit_spatial), colour = NA) +
  ggtitle("MTPL claim frequency data") +
  scale_fill_gradient(low = "#99CCFF",
                      high = "#003366") +
  theme_bw()
```

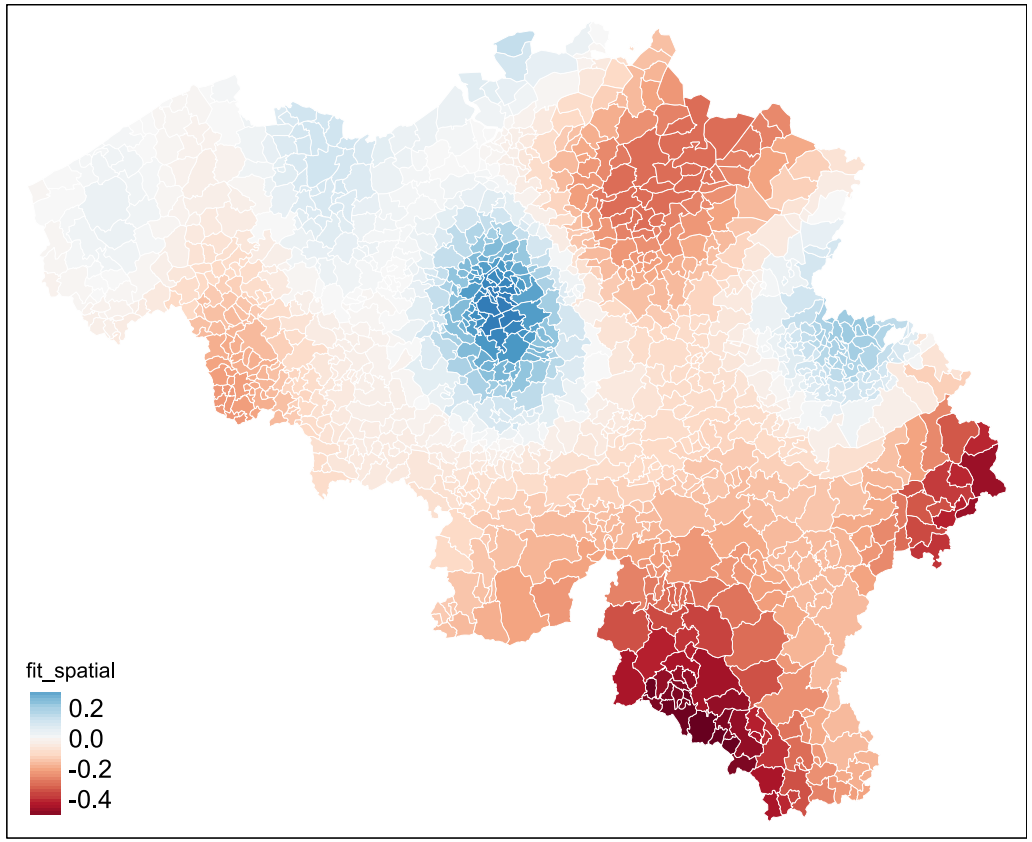


Using the tools from the {tmap} package

```
tm_shape(belgium_shape_sf) +
  tm_borders(col = 'white', lwd = .1) +
  tm_fill("fit_spatial", style = "cont",
          palette = "RdBu", legend.reverse = TRUE,
          auto.palette.mapping = TRUE) +
  tm_layout(legend.title.size = 1.0,
            legend.text.size = 1.0)
```

For use later on we remove the `fit_spatial` variable from `belgium_shape_sf`:

```
library(dplyr)
belgium_shape_sf ← belgium_shape_sf %>%
                   dplyr::select(-fit_spatial)
```



Putting it all together

We are now ready to fit the preferred GAM from Henckaerts et al. (SAJ, 2018).

The proposed selection of variables is based on an **exhaustive search** over all possible combinations of variables.

```
freq_gam ← gam(nclaims  ~
                  coverage + fuel +
                  s(ageph) + s(bm) +
                  s(power) + s(long, lat) +
                  ti(ageph, power, bs = "tp"),
                  offset = log(expo),
                  data = mtpl,
                  family = poisson(link = "log"))
```

```
summary(freq_gam)
##
## Family: poisson
## Link function: log
##
## Formula:
## nclaims ~ coverage + fuel + s(ageph) + s(bm) + s(pov
##     lat) + ti(ageph, power, bs = "tp")
##
## Parametric coefficients:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -1.9745911  0.0216710 -91.117  < 2e-16
## coveragePO    0.0008931  0.0239860   0.037     0.97
## coverageTPL   0.1221058  0.0221623   5.510  3.6e-08
## fuelgasoline -0.1817652  0.0158127 -11.495  < 2e-16
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.
##
## Approximate significance of smooth terms:
##                   edf Ref.df Chi.sq  p-value
## s(ageph)        5.709  6.754 295.74  < 2e-16 ***
## s(bm)           6.458  7.373 991.37  < 2e-16 ***
```

From GAM to GLM using clustering and tree methods

Binning the spatial effect

We continue working with `post_dt`, a data set with the 1 146 Belgian postal codes, their corresponding `lat` and `long` (of the center of the postal code).

We want to use the `predict` function of the GAM stored in `freq_gam` on the data set `post_dt`.

Hereto, we need to extend `post_dt` with values for the other covariates used by `freq_gam`.

```
post_dt$coverage ← mtpl$coverage[1]
post_dt$fuel ← mtpl$fuel[1]
post_dt[c("bm", "ageph", "power", "expo")] ←
  c(mtpl$bm[1], mtpl$ageph[1], mtpl$power[1],
    mtpl$expo[1])
```

We will use $\hat{f}(\mathbf{long}, \mathbf{lat})$ from the `freq_gam` object.

We create a new data set `dt_pred`, with the postal codes, the coordinates of their centroids and the fitted spatial effect.

```
pred ← predict(freq_gam, newdata = post_dt,
                type = "terms", terms = "s(long,lat)")
dt_pred ← tibble(pc = post_dt$POSTCODE,
                 long = post_dt$long,
                 lat = post_dt$lat, pred)
names(dt_pred)[4] ← "fit_spatial"
```

```
dt_pred ← dplyr::arrange(dt_pred, pc)
```

Binning the spatial effect (cont.)

We use the fitted spatial effect $s_i := \hat{f}\left(\texttt{long}_i, \texttt{lat}_i\right)$ for every $i = 1, \ldots, 1\,146$.

We split or bin using one of the following: (also see the {classint} package):

- equal intervals: k bins of equal length $\frac{\max\left(s_i\right) - \min\left(s_i\right)}{k}$
- quantile binning: each bin contains $\approx \frac{1\,146}{k}$ municipalities
- complete linkage (i.e. hierarchical clustering)
- Fisher's natural breaks.

The goal is to construct homogeneous class intervals (or bins).

With Fisher's natural breaks (i.e. K-means clustering):

- minimize the sum of squared distances between observations $s_u^{(l)}$ and the bin means

$$\sum_{l=1}^{K} \sum_{u=1}^{n_l} (s_u^{(l)} - \bar{s}^{(l)})^2,$$

with K the number of bins.

In R we use the `classIntervals` function

```
num_bins ← 5
library(classInt)
classint_fisher ← classIntervals(
                    dt_pred$fit_spatial,
                    num_bins,
                    style = "fisher")

classint_fisher$brks

min(dt_pred$fit_spatial)
max(dt_pred$fit_spatial)
```

We fix the number of bins upfront and store it as `num_bins`.

This is a tuning parameter, and we discuss a tuning strategy later on.

```
num_bins ← 5
library(classInt)
classint_fisher ← classIntervals(
                    dt_pred$fit_spatial,
                    num_bins,
                    style = "fisher")

classint_fisher$brks

min(dt_pred$fit_spatial)
max(dt_pred$fit_spatial)
```

In R we use the `classInt::classIntervals` function.

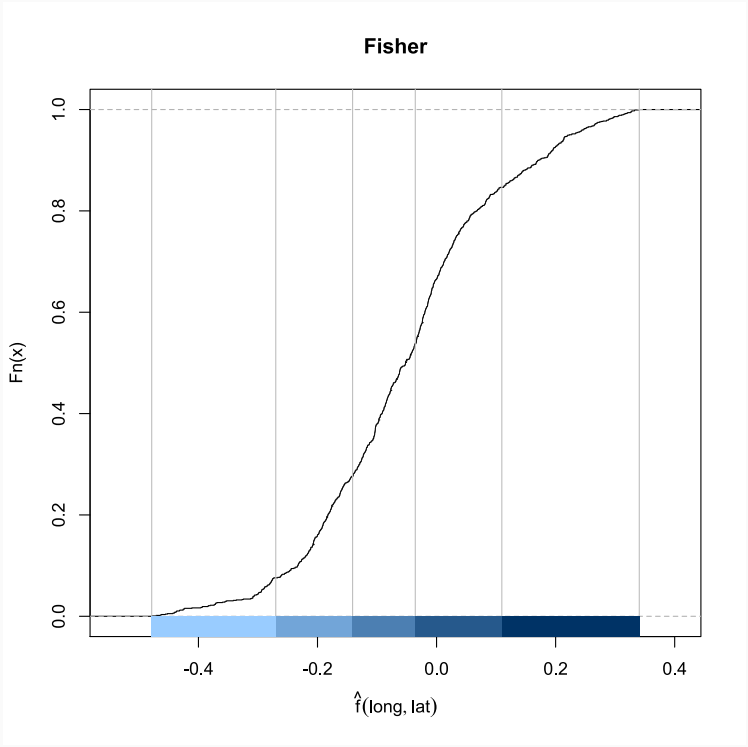
We specify the fitted values of the spatial smoother, stored in `dt_pred$fit_spatial`, the number of bins and the clustering technique.

We inspect the constructed bins, stored in `classint_fisher`:

```
classint_fisher$brks
## [1] -0.47852103 -0.27012356 -0.14106325 -0.03575633
min(dt_pred$fit_spatial)
## [1] -0.478521
max(dt_pred$fit_spatial)
## [1] 0.3405182
```

We picture the binned spatial effect using the `plot` function that comes with the {classInt} package.

```
crp ← colorRampPalette(c("#99CCFF", "#003366"))
plot(classint_fisher, crp(num_bins),
     xlab = expression(hat(f)(long,lat)),
     main = "Fisher")
```



We merge the `fit_spatial` effect with the `belgium_shape_sf` object, per postal code.

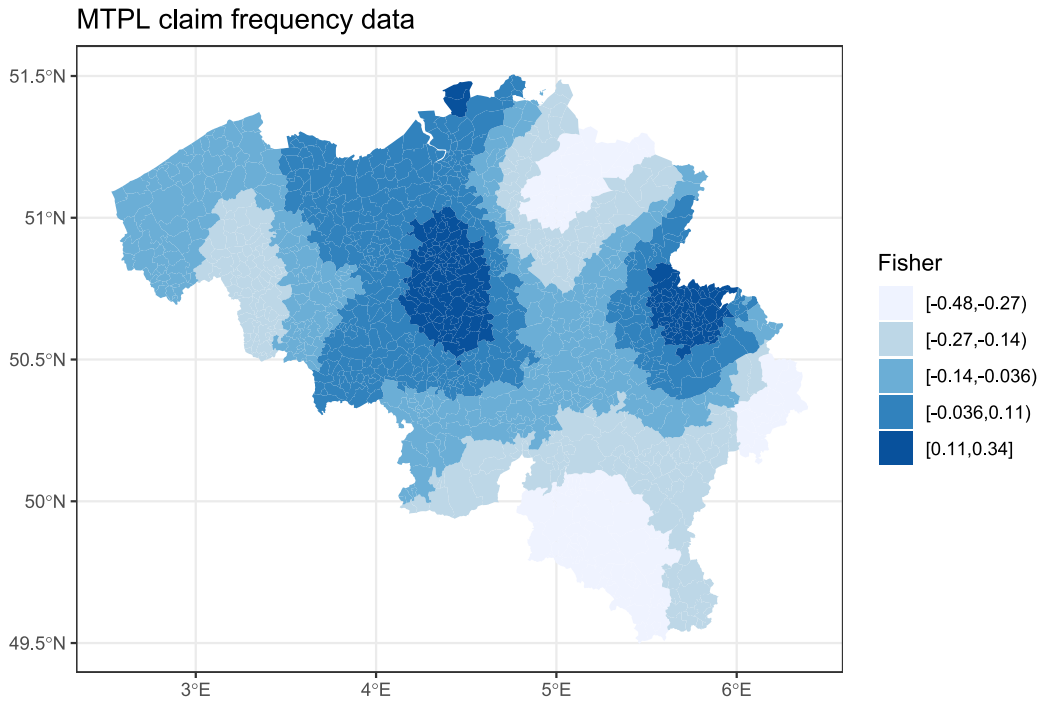
```
belgium_shape_sf ← left_join(belgium_shape_sf,
                             dt_pred,
                             by = c("POSTCODE" =
                                    "pc"))
```

Using the `cut(.)` function we bin the `fit_spatial` effect using the break points stored in `classint_fisher`.

```
belgium_shape_sf$class_fisher ←
  cut(belgium_shape_sf$fit_spatial,
      breaks = classint_fisher$brks,
      right = FALSE, include.lowest = TRUE,
      dig.lab = 2)
```

As a final step, we visualize the constructed clusters on the map of Belgium.

```
ggplot(belgium_shape_sf) + theme_bw() +
  labs(fill = "Fisher") +
  geom_sf(aes(fill = class_fisher), colour = NA) +
  ggtitle("MTPL claim frequency data") +
  scale_fill_brewer(palette = "Blues",
                    na.value = "white") +
  theme_bw()
```



Your turn

You will further explore the clustering methods in the `classIntervals(.)` function from the {classInt} package.

Q: rerun the coding steps discussed above

1. use a different `style = .` argument. Search the `? classIntervals` for other options.
2. visualize the resulting clusters on the cdf of the fitted spatial smoother.
3. visualize the resulting clusters on the map of Belgium.

Tuning the number of spatial clusters

We now construct a new data set where the spatial effect is replaced by a binned version.

The binnend effect is stored as `mtpl_geo$geo`.

```
library(dplyr)
mtpl_geo ← mtpl %>% dplyr::select(nclaims, expo, coverage, fuel, ageph, bm, power, pc)
mtpl_geo ← left_join(mtpl_geo, dt_pred)
mtpl_geo$geo ← as.factor(cut(mtpl_geo$fit_spatial,
                             breaks = classint_fisher$brks, right = FALSE,
                             include.lowest = TRUE, dig.lab = 2))
head(mtpl_geo$geo)
## [1] [0.11,0.34] [0.11,0.34] [0.11,0.34] [0.11,0.34] [0.11,0.34] [0.11,0.34]
## 5 Levels: [-0.48,-0.27) [-0.27,-0.14) [-0.14,-0.036) ... [0.11,0.34]
```

On this new data set, we fit the GAM, with the factor variable `geo` replacing the spatial smoother `s(long, lat)`.

To tune the number of spatial bins, we store the AIC/BIC of `freq_gam_geo`.

We repeat the calculation over a grid of values for `num_bins`.

The optimal number of bins is chosen by minimizing the AIC/BIC.

```
freq_gam_geo ← gam(nclaims ~ coverage + fuel + s(ageph) + s(bm) + s(power) +
                   ti(ageph, power, bs = "tp") + geo,
                   offset = log(expo) , data = mtpl_geo, family = poisson(link = "log"))
summary(freq_gam_geo)
##
## Family: poisson
## Link function: log
##
## Formula:
## nclaims ~ coverage + fuel + s(ageph) + s(bm) + s(power) + ti(ageph,
##     power, bs = "tp") + geo
##
## Parametric coefficients:
##                   Estimate Std. Error z value Pr(>|z|)
## (Intercept)       -2.271692   0.055518 -40.918  < 2e-16 ***
## coveragePO        -0.004255   0.023915  -0.178  0.85879
## coverageTPL        0.118676   0.022002   5.394 6.89e-08 ***
## fuelgasoline      -0.179961   0.015758 -11.420  < 2e-16 ***
## geo[-0.27,-0.14)   0.122503   0.055900   2.191  0.02842 *
```

Binning the univariate smooth effects

When binning the continuous risk factors, we aim to construct consecutive intervals.

The clustering methods used to bin the spatial effect do not guarantee consecutive intervals.

We replace them with tree-based methods.

The tree will minimize the following MSE:

$$\frac{\sum_{i=\min(\mathbf{ageph})}^{\max(\mathbf{ageph})} w_{\mathbf{ageph}_i} (\hat{f}\left(\mathbf{ageph}_i\right) - \hat{f}^b(\mathbf{ageph}_i))^2}{\sum_{i=\min(\mathbf{ageph})}^{\max(\mathbf{ageph})} w_{\mathbf{ageph}_i}}.$$

Here, $\hat{f}^b(\mathbf{ageph}_i)$ is a binned approximation of $\hat{f}\left(\mathbf{ageph}_i\right)$, obtained by fitting a constant per constructed bin.

To balance goodness-of-fit and complexity, the `evtree` minimizes

$$n \cdot \text{MSE} + \alpha \cdot \text{complexity},$$

where α is a tuning parameter.

Binning the univariate smooth effects (cont.)

Starting from the GAM stored in `freq_gam_geo` we extract a data set with the unique values of a continous risk factor, the number of records observed for each value, and the fitted smoother evaluated in these values.

```
getGAMdata_single = function(model, term, var, varname){
  pred ← predict(model, type = "terms", terms = term)
  dt_pred ← tibble("x" = var, pred)
  dt_pred ← arrange(dt_pred, x)
  names(dt_pred) = c("x", "s")
  dt_unique ← unique(dt_pred)
  dt_exp ← dt_pred %>% group_by(x) %>% summarize(tot = n())
  dt_exp ← dt_exp[c("x", "tot")]
  GAM_data ← left_join(dt_unique, dt_exp)
  names(GAM_data) ← c(varname, "s", "tot")
  GAM_data ← GAM_data[which(GAM_data$tot ≠ 0), ]
  return(GAM_data)
}

freq_gam_ageph ← getGAMdata_single(freq_gam_geo, "s(ageph)", mtpl_geo$ageph, "ageph")
```

Binning the univariate smooth effects (cont.)

We put focus on $\hat{f}(\mathbf{ageph})$ and use (evolutionary) trees to bin the fitted effect, by making splits on `ageph` only.

```
library(evtree)
source("./scripts/evtree.R")
```

```
ctrl.freq ← evtree.control(
              minbucket = 0.05*nrow(mtpl),
              alpha = 550, maxdepth = 5)
```

Notice the `weights` argument in the `evtree` function, using the number of policyholders with a particular `ageph` as a weight.

```
evtree_freq_ageph ← evtree(s ~ ageph,
                           data = freq_gam_ageph,
                           weights = tot,
                           control = ctrl.freq)
evtree_freq_ageph
```

```
##
## Model formula:
## s ~ ageph
##
## Fitted party:
## [1] root
## |   [2] ageph < 56
## |   |   [3] ageph < 33
## |   |   |   [4] ageph < 29
## |   |   |   |   [5] ageph < 26: *
## |   |   |   |   [6] ageph ≥ 26: *
## |   |   |   [7] ageph ≥ 29: *
## |   |   [8] ageph ≥ 33
## |   |   |   [9] ageph < 51: *
## |   |   |   [10] ageph ≥ 51: *
## |   [11] ageph ≥ 56
## |   |   [12] ageph < 61: *
## |   |   [13] ageph ≥ 61
## |   |   |   [14] ageph < 73: *
## |   |   |   [15] ageph ≥ 73: *
##
## Number of inner nodes:    7
```

Binning the univariate smooth effects (cont.)

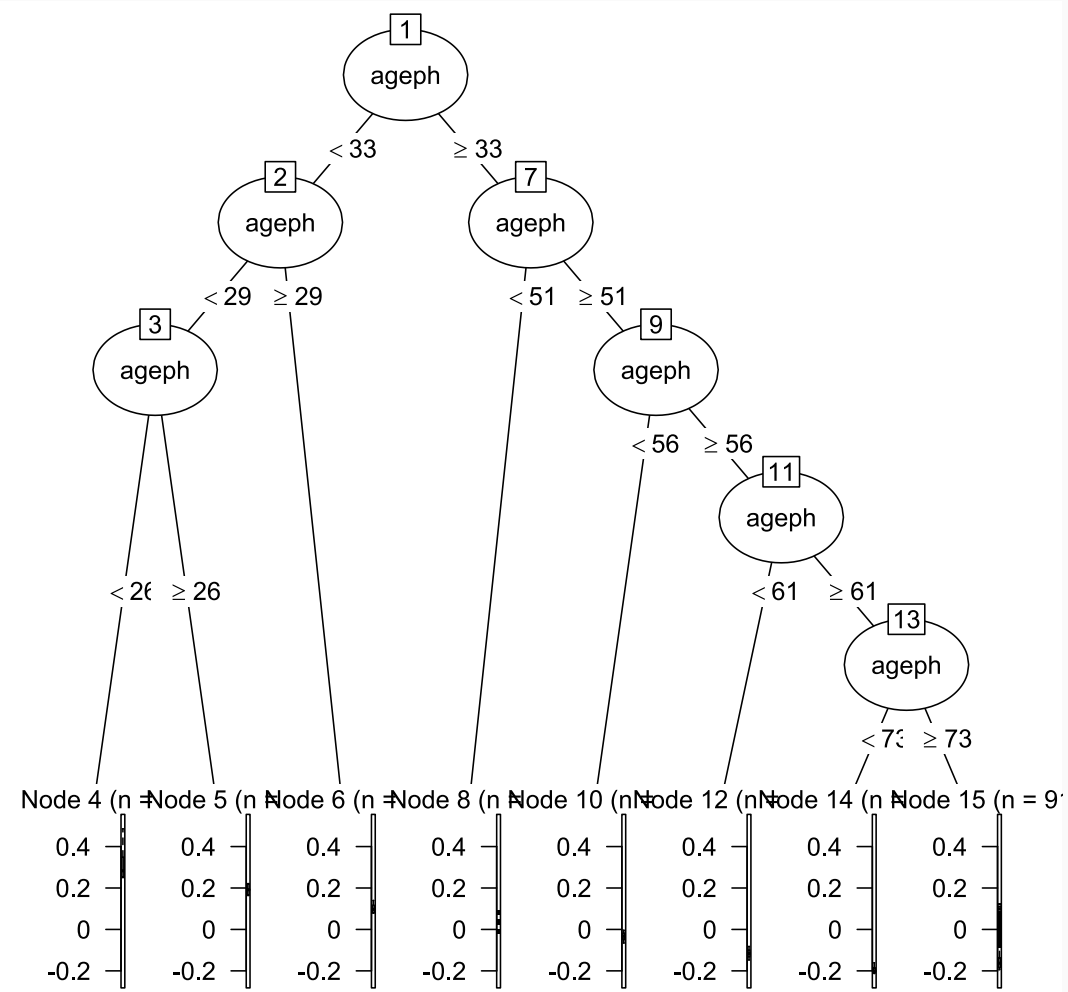
We put focus on $\hat{f}(\mathbf{ageph})$ and use (evolutionary) trees to bin the fitted effect, by making splits on `ageph` only.

```
library(evtree)
source("./scripts/evtree.R")
```

```
ctrl.freq ← evtree.control(
              minbucket = 0.05*nrow(mtpl),
              alpha = 550, maxdepth = 5)
```

Notice the `weights` argument in the `evtree` function, using the number of policyholders with a particular `ageph` as a weight.

```
evtree_freq_ageph ← evtree(s ~ ageph,
                           data = freq_gam_ageph,
                           weights = tot,
                           control = ctrl.freq)
plot(evtree_freq_ageph)
```



Your turn

You will now examine the splits created for the other smooth effects of continuous risk factors.

Q: you will work through the following exploratory steps.

1. Split the smooth effects of `bm` and `power` using `evtree`.
2. Visualize the constructed trees.
3. What would (conceptually) be your strategy to split the interaction effect `ti(ageph, power, bs = "tp")`.

10:00

To extract the split points from the tree model stored in `evtree_freq_ageph` we use the following helper function

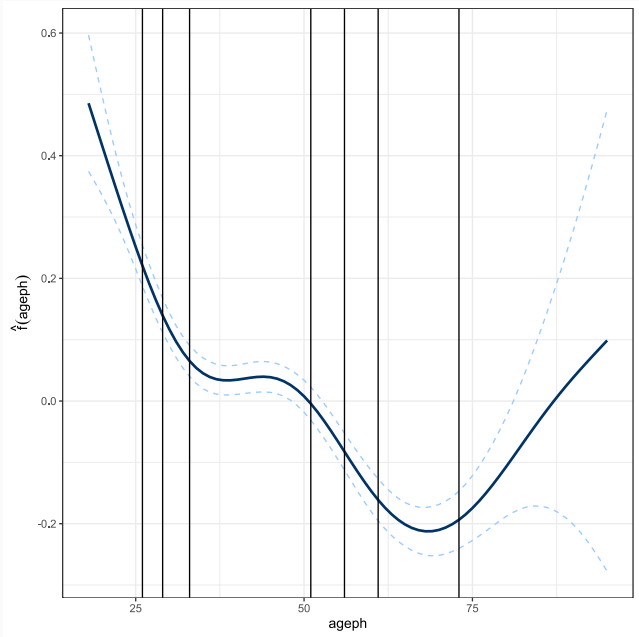
```
splits_evtree = function(evtreemodel, GAMvar, DTvar){
  preds ← predict(evtreemodel, type = "node")
  nodes ← data.frame("x" = GAMvar, "nodes" = preds)
  nodes$change ← c(0, pmin(1, diff(nodes$nodes)))
  splits_evtree ← unique(c(min(DTvar),
                  nodes$x[which(nodes$change==1)],
                  max(DTvar)))
  return(splits_evtree)
}
```

```
freq_splits_ageph ← splits_evtree(
                            evtree_freq_ageph,
                            freq_gam_ageph$ageph,
                            mtpl$ageph)
freq_splits_ageph
## [1] 18 26 29 33 51 56 61 73 95
```

To compare the fitted smooth effect with the constructed bins of `ageph` we use

```
ggplot.gam ← function(model, variable, gam_term,
                      xlabel, ylabel){
  pred ← predict(model, type = "terms", se = TRUE)
  col_index ← which(colnames(pred$fit)==gam_term)
  x ← variable
  b ← pred$fit[, col_index]
  l ← pred$fit[, col_index] -
         qnorm(0.975) * pred$se.fit[, col_index]
  u ← pred$fit[, col_index] +
         qnorm(0.975) * pred$se.fit[, col_index]
  df ← unique(data.frame(x, b, l, u))
  p ← ggplot(df, aes(x = x))
  p ← p + geom_line(aes(y = b), size = 1,
                    col = "#003366")
  p ← p + geom_line(aes(y = l), size = 0.5,
                    linetype = 2, col = "#99CCFF")
  p ← p + geom_line(aes(y = u), size = 0.5,
                    linetype = 2, col = "#99CCFF")
  p ← p + xlab(xlabel) + ylab(ylabel) + theme_bw()
  p
}
```

```
plot_freq_bin_ageph ← ggplot.gam(freq_gam_geo,
                                 mtpl_geo$ageph,
                                 "s(ageph)", "ageph",
                                 expression(hat(f)(age

plot_freq_bin_ageph ← plot_freq_bin_ageph +
  geom_vline(xintercept =
               freq_splits_ageph[2:(length(freq_splits_
plot_freq_bin_ageph
```



Putting it all together

```
mtpl_bin ← mtpl_geo[c("nclaims", "expo", "coverage", "fuel", "geo")]
mtpl_bin$ageph ← cut(mtpl_geo$ageph, freq_splits_ageph, right = FALSE, include.lowest = TRUE)
summary(mtpl_bin$ageph)
## [18,26) [26,29) [29,33) [33,51) [51,56) [56,61) [61,73) [73,95]
##    8258    9265   13939   69055   15674   12591   25323    9126
```

```
summary(mtpl_bin$coverage)
##    FO    PO   TPL
## 22107 45988 95136
summary(mtpl_bin$fuel)
##   diesel gasoline
##    50398   112833
summary(mtpl_bin$geo)
##  [-0.48,-0.27)  [-0.27,-0.14) [-0.14,-0.036)  [-0.036,0.11)    [0.11,0.34]
##           4122          21809          34241          71374          31685
```

```
mtpl_bin$ageph ← relevel(mtpl_bin$ageph, ref = "[33,51)")
mtpl_bin$geo ← relevel(mtpl_bin$geo, ref = "[-0.036,0.11)")
mtpl_bin$coverage ← relevel(mtpl_bin$coverage, ref = "TPL")
mtpl_bin$fuel ← relevel(mtpl_bin$fuel, ref = "gasoline")
```

```r
freq_glm ← gam(nclaims ~ coverage + fuel + ageph + geo, offset = log(expo),
                         data = mtpl_bin, family = poisson(link = "log"))

summary(freq_glm)
##
## Family: poisson
## Link function: log
##
## Formula:
## nclaims ~ coverage + fuel + ageph + geo
##
## Parametric coefficients:
##                   Estimate Std. Error  z value Pr(>|z|)
## (Intercept)       -1.95021    0.01555 -125.444  < 2e-16 ***
## coverageFO        -0.13001    0.02159   -6.022 1.73e-09 ***
## coveragePO        -0.15759    0.01666   -9.458  < 2e-16 ***
## fueldiesel         0.17609    0.01503   11.714  < 2e-16 ***
## ageph[18,26)       0.61163    0.02620   23.340  < 2e-16 ***
## ageph[26,29)       0.37585    0.02715   13.846  < 2e-16 ***
## ageph[29,33)       0.19884    0.02457    8.092 5.88e-16 ***
## ageph[51,56)      -0.09561    0.02599   -3.678 0.000235 ***
## ageph[56,61)      -0.21004    0.02995   -7.013 2.33e-12 ***
## ageph[61,73)      -0.32697    0.02355  -13.887  < 2e-16 ***
## ageph[73,95]      -0.29090    0.03590   -8.102 5.39e-16 ***
## geo[-0.48,-0.27)  -0.36796    0.05303   -6.938 3.97e-12 ***
## geo[-0.27,-0.14)  -0.23224    0.02323   -9.996  < 2e-16 ***
```

```r
freq_glm ← gam(nclaims ~ coverage + fuel + ageph + geo, offset = log(expo),
                         data = mtpl_bin, family = poisson(link = "log"))

anova(freq_glm)
##
## Family: poisson
## Link function: log
##
## Formula:
## nclaims ~ coverage + fuel + ageph + geo
##
## Parametric Terms:
##          df Chi.sq p-value
## coverage  2  103.7  <2e-16
## fuel      1  137.2  <2e-16
## ageph     7 1326.1  <2e-16
## geo       4  587.9  <2e-16
```

Your turn

You will now extend the GLM obtained so far with factor effects of `bm` and `power`.

Q: you will work through the following exploratory steps.

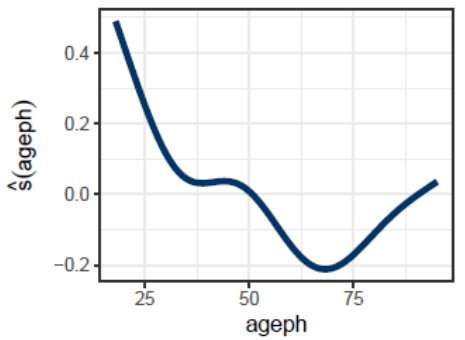
1. Extract the split points constructed for the smooth effects of `bm` and `power`.
2. Split the continuous variables `bm` and `power` using these split points.
3. Extend the GLM with the constructed factor variables. Pick an appropriate reference level.

10:00

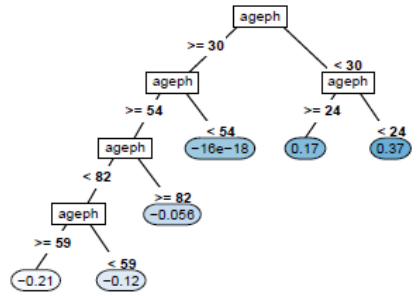
That's a wrap!

A data driven binning strategy for the construction of insurance tariff classes

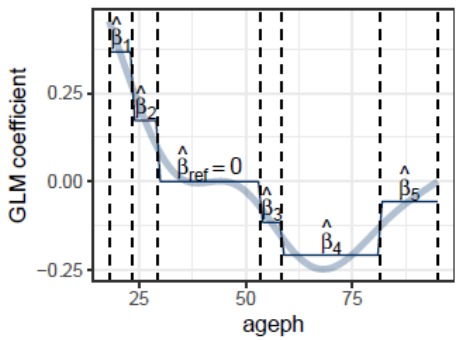
by Henckaerts, Antonio, Clijsters & Verbelen (2018).



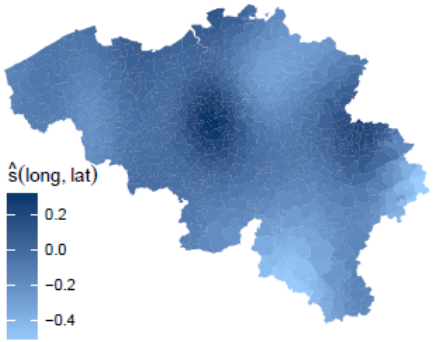
(1a) Smooth continuous effect



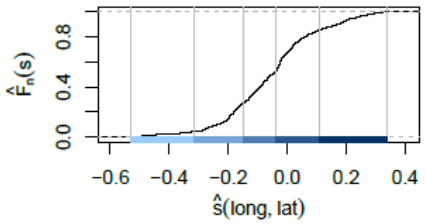
(1b) Supervised decision tree



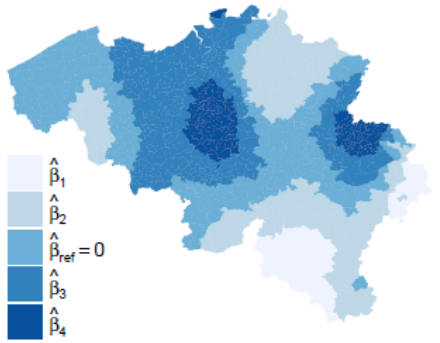
(1c) Binned continuous effect



(2a) Smooth spatial effect



(2b) Unsupervised clustering



(2c) Binned spatial effect

What else is there?

Sparse regression with multi-type feature modelling

by Devriendt, Antonio et al. (2018)

- automatic feature selection and binning of risk factors via **regularization**
- R package {smurf} on CRAN
- end product is a GLM!

Boosting insights in insurance tariff plans with tree-based machine learning

by Henckaerts, Côté, Antonio & Verbelen (2020), North American Actuarial Journal.

- GLMs, GAMs, decision trees, random forests and gradient boosting machines
- R packages extended to Poisson and gamma deviance
- tuning strategy, interpretability tools and managerial insights
- supplementary material on GitHub.

Thanks!

Slides created with the R package xaringan.

Course material available via

https://github.com/katrienantonio/PE-pricing-analytics