Hands-on Workshop in R

APC module Productontwikkeling, UvA

Katrien Antonio, Thomas de Boer & Gemma van de Sande

Productontwikkeling R workshop | November 3, 2020

Prologue

Introduction

Course

https://github.com/katrienantonio/APC-workshop-Productontwikkeling

The course repo on GitHub, where you can find the data sets, lecture sheets, R scripts and R markdown files.

Us

https://katrienantonio.github.io/

katrien.antonio@kuleuven.be & thomas.de.boer@movir.nl & gemma.van.de.sande@asr.nl

(Katrien) Professor in insurance data science, teacher and coordinator of the APC module in Insurance Data Science

Checklist

☑ Do you have a fairly recent version of R?

```
version$version.string
## [1] "R version 4.0.3 (2020-10-10)"
```

☑ Do you have a fairly recent version of RStudio?

```
RStudio.Version()$version
## Requires an interactive session but should return something like "[1] ‘1.3.1093’"
```

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

or

☑ Have you created an account on RStudio Cloud (to avoid any local installation issues)?

Why this session?

The goals of this session 🚀

- develop practical **data handling and cleaning foundations**
- visualize and explore data
- **recap** covering the basics of linear regression models, generalized linear models and (a bit of) generalized additive models in actuarial science or econometrics
- learn by doing, get you started (in particular when you have limited experience in R).

> *"In short, we will cover things that we wish someone had taught us in our undergraduate programs."*
>
> This quote is from the Data science for economists course by Grant McDermott.

What's out there - the R universe

What is R?

> The R environment is an integrated suite of software facilities for data manipulation, calculation and graphical display.

A brief history:

- R is a dialect of the S language.
- R was written by **R**obert Gentleman and **R**oss Ihaka in 1992.
- The R source code was first released in 1995.
- In 1998, the Comprehensive R Archive Network CRAN was established.
- The first official release, R version 1.0.0, dates to 2000-02-29. Currently R 4.0.3 (October, 2020).
- R is open source via the GNU General Public License.

Explore the R architecture

- R is like a car's engine
- RStudio is like a car's dashboard, an integrated development environment (IDE) for R.

R: Engine	RStudio: Dashboard
	

How do I code in R?

Keep in mind:

- unlike other software like Excel, STATA, or SAS, R is an interpreted language
- no point and click in R!
- you have to program in R!

R packages extend the functionality of R by providing additional functions, and can be downloaded for free from the internet.

R: A new phone	R Packages: Apps you can download
	

How to install and load an R package?

Install the {ggplot2} package for data visualisation

```
install.packages("ggplot2")
```

Load the installed package

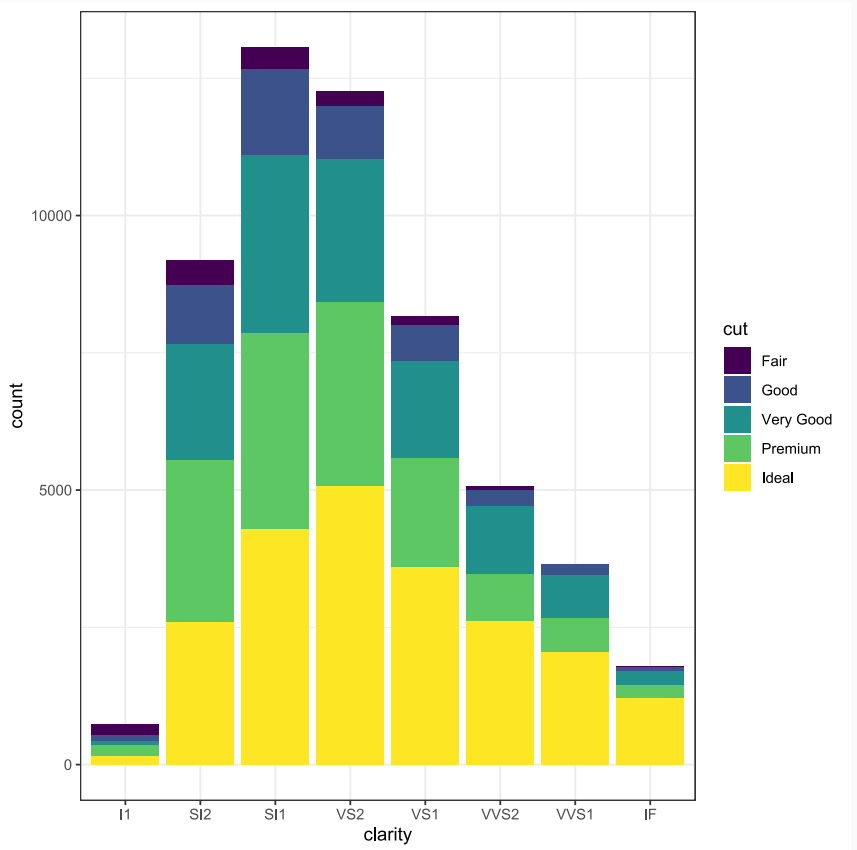
```
library(ggplot2)
```

And give it a try

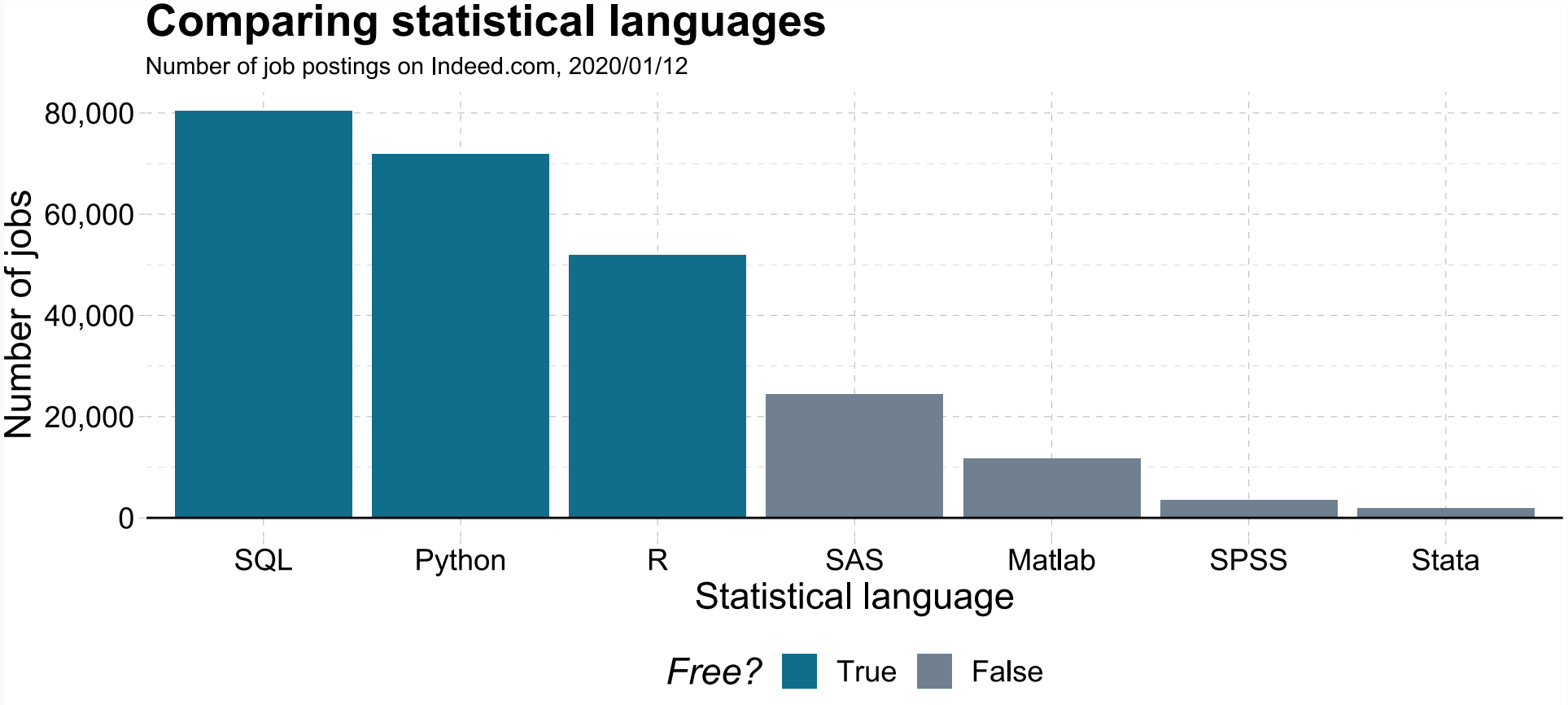
```
head(diamonds)
ggplot(diamonds, aes(clarity, fill = cut)) +
  geom_bar() + theme_bw()
```

Packages are developed and maintained by R users worldwide.

They are shared with the R community through CRAN: now 16,460 packages online (on November 2, 2020)!



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

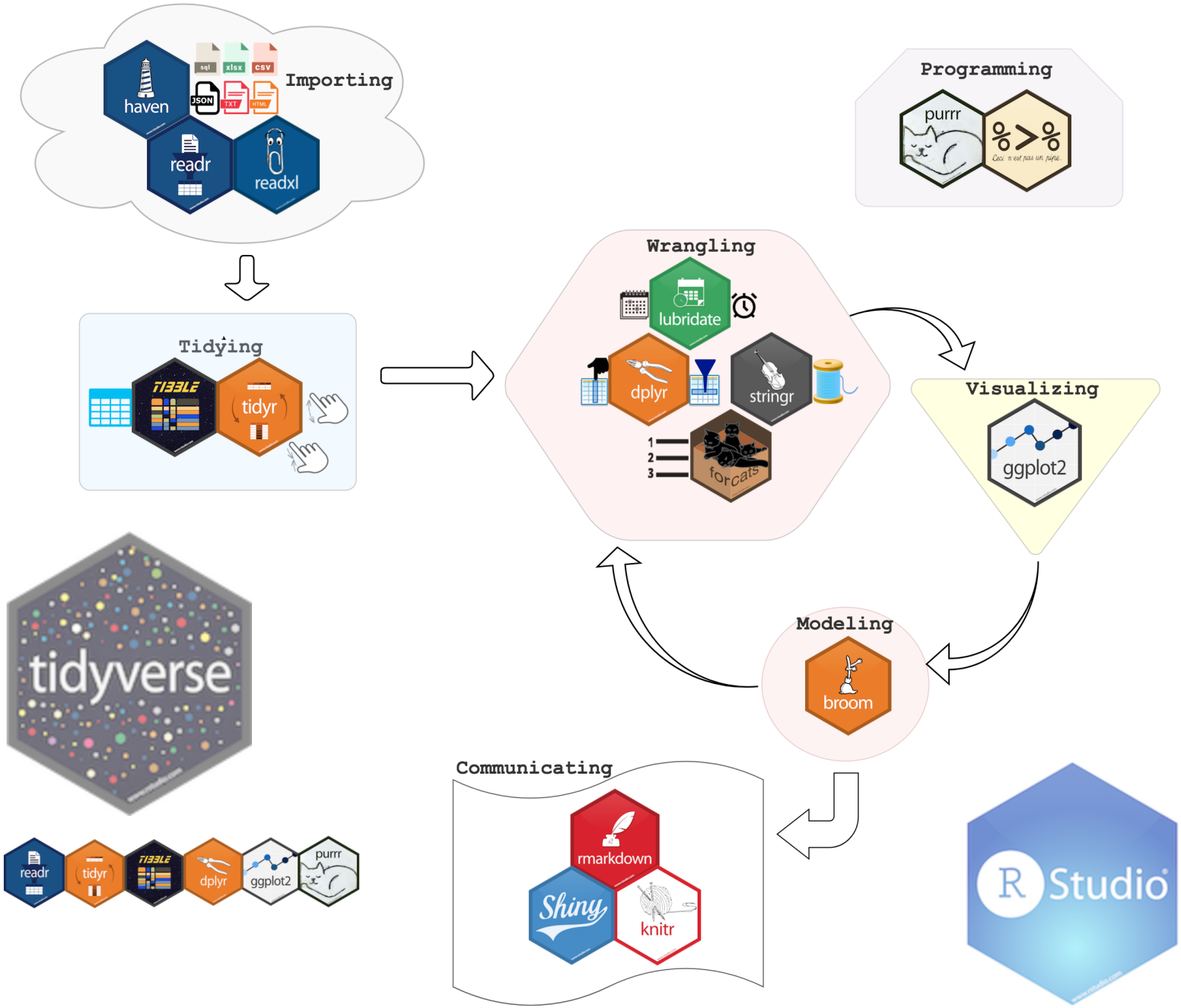
- Next to Python, R has become the *de facto* language for data science, with a cutting edge *machine learning toolbox*.
- See: The Popularity of Data Science Software
- R is open-source with a very active community of users spanning academia and industry.

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

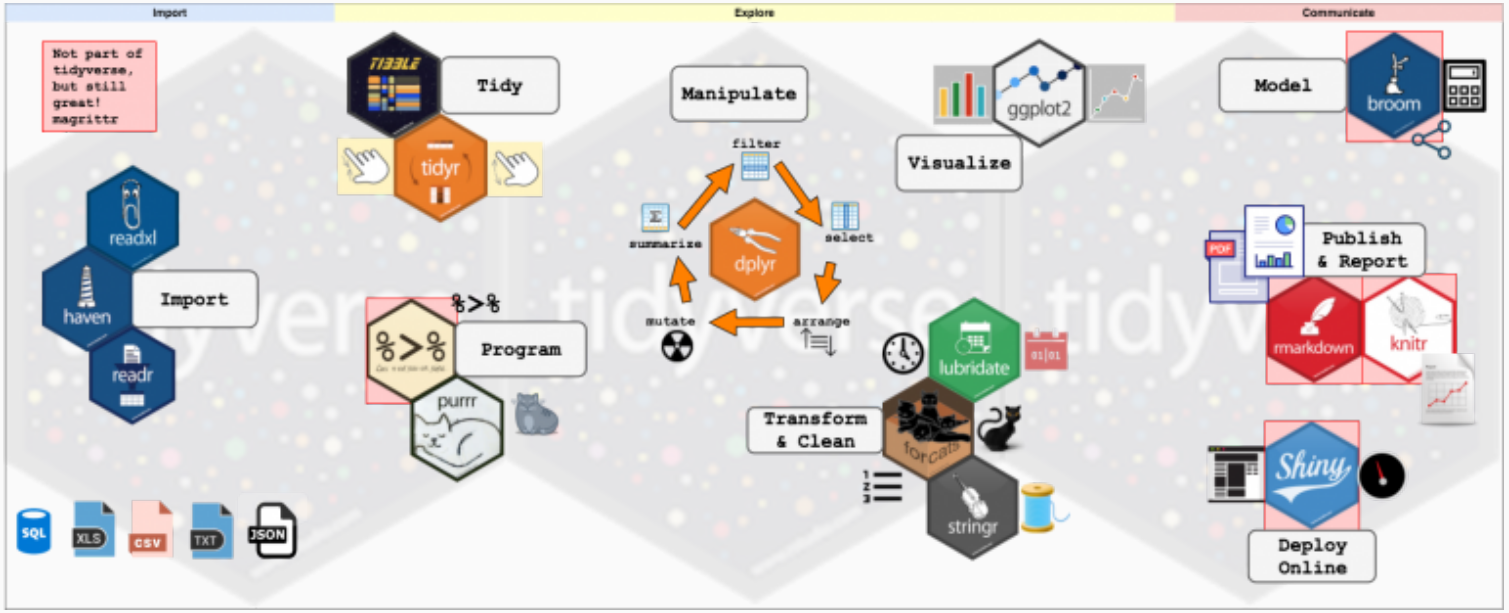
Disclaimer + Read more

- 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
tibble
tidyr
Wrangling
lubridate
dplyr
stringr
forcats
Visualizing
ggplot2
Modeling
broom
Communicating
rmarkdown
Shiny
knitr
tidyverse
readr
tidyr
tibble
dplyr
ggplot2
purrr
RStudio

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**.

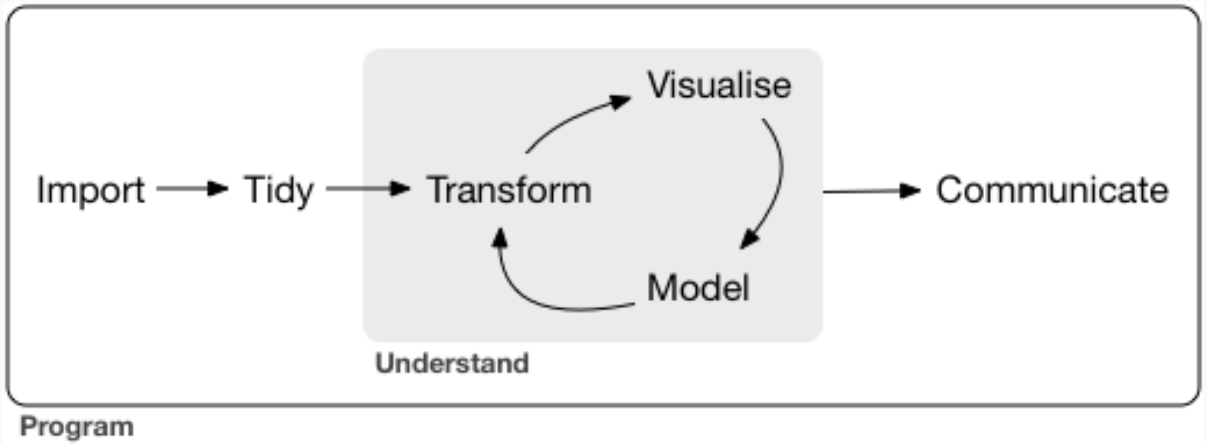


Figure and quote taken from Chapter 1 in R for data science.

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} for data visualisation
  - what else is there?
- Data sets used in the workshop
  - MTPL data
  - Aemas housing data
  - AOV data, country and company level
- Building linear and generalized linear models in R
  - tidy model output with {broom}
  - linear regression with `lm`
  - fitting and building GLMs with `glm`

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:

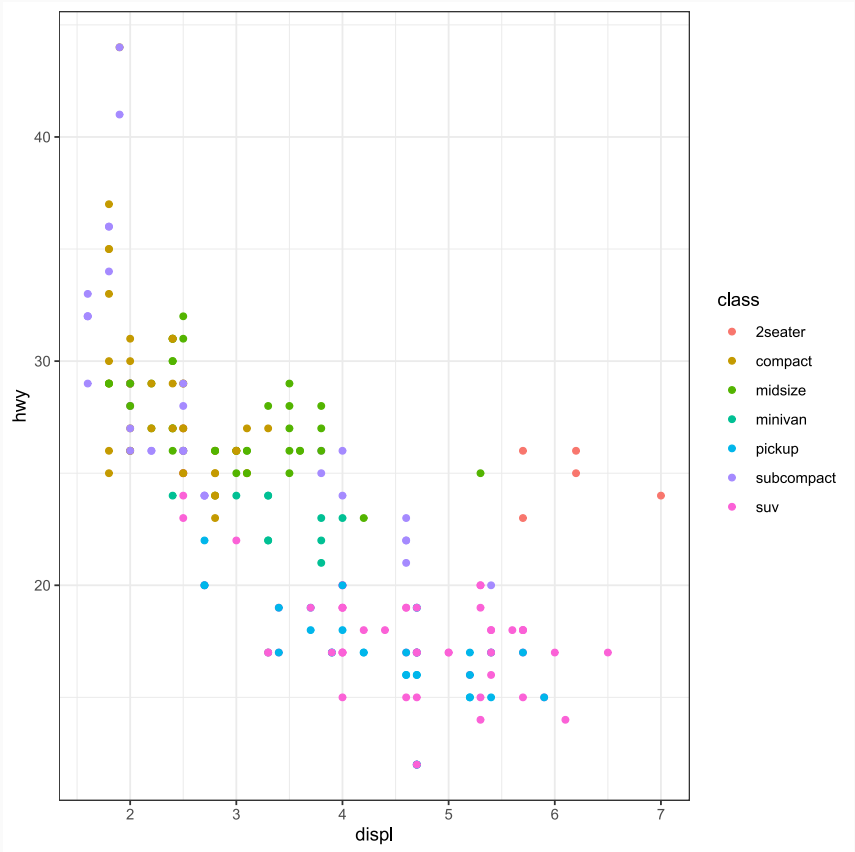
- `geom_point()` adds a layer of points to your plot, which creates a scatterplot
- `geom_smooth()` adds a smooth line
- `geom_bar` a bar plot

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.

What else is there?

Recall

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

There are **(multiple) alternative ways** to do what the packages and functions in the tidyverse do.

For instance:

- base R
- the {data.table} package

You can read more about comparisons on e.g. how to write this in base R or Base R, the tidyverse, and data.table: a comparison of R dialects to wrangle your data.

Data sets used in the session

Data sets used in this session - MTPL

We illustrate some first data handling steps on 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 sets used in this session - MTPL

You can load the data from a .R script in the course material:

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

If you work in an R notebook or R markdown file, you can also go for:

```
# install.packages("here")
library(here)
dir ← here::here()
setwd(dir)
mtpl_orig ← read.table('./data/PC_data.txt',
                                header = TRUE)
mtpl_orig ← as_tibble(mtpl_orig)
```

The last instruction transforms `mtpl_orig` into a `tibble`.

These instructions are recommended because they avoid referring to a specific working directory on your computer, e.g.

```
setwd("C:\\Users\\u0043788\\Dropbox\\AOV module APC")
mtpl_orig ← read.table('PC_data.txt', header = TRUE)
```

If you organize your data analysis or project in a folder on your computer that holds all relevant files, then the above instructions allow you (and your colleagues) to get started right away.

The only thing you need is an organized file structure.

The {rstudioapi} package is developed for RStudio. The {here} library is more general.

Do note that when working in a .Rmd, {here} will use the folder that markdown lives in as the working directory, but if you are working in a script (.R) the working directory is the top level of the project file.

First of all, we explore the structure of `mtpl_orig` with `str()`.

```
str(mtpl_orig)
## tibble [163,231 x 18] (S3: tbl_df/tbl/data.frame)
##  $ ID      : int [1:163231] 1 2 3 4 5 6 7 8 9 10 ...
##  $ NCLAIMS : int [1:163231] 1 0 0 0 1 0 1 0 0 0 ...
##  $ AMOUNT  : num [1:163231] 1618 0 0 0 156 ...
##  $ AVG     : num [1:163231] 1618 NA NA NA 156 ...
##  $ EXP     : num [1:163231] 1 1 1 1 0.0466 ...
##  $ COVERAGE: chr [1:163231] "TPL" "PO" "TPL" "TPL" ...
##  $ FUEL    : chr [1:163231] "gasoline" "gasoline" "diesel" "gasoline" ...
##  $ USE     : chr [1:163231] "private" "private" "private" "private" ...
##  $ FLEET   : chr [1:163231] "N" "N" "N" "N" ...
##  $ SEX     : chr [1:163231] "male" "female" "male" "male" ...
##  $ AGEPH   : int [1:163231] 50 64 60 77 28 26 26 58 59 34 ...
##  $ BM      : int [1:163231] 5 5 0 0 9 11 11 11 0 7 ...
##  $ AGEC    : int [1:163231] 12 3 10 15 7 12 8 14 3 6 ...
##  $ POWER   : int [1:163231] 77 66 70 57 70 70 55 47 98 74 ...
##  $ PC      : int [1:163231] 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 ...
##  $ TOWN    : chr [1:163231] "BRUSSEL" "BRUSSEL" "BRUSSEL" "BRUSSEL" ...
##  $ LONG    : num [1:163231] 4.36 4.36 4.36 4.36 4.36 ...
##  $ LAT     : num [1:163231] 50.8 50.8 50.8 50.8 50.8 ...
```

Or with `head()`...

```
head(mtpl_orig) %>% kable(format = 'html')
```

ID	NCLAIMS	AMOUNT	AVG	EXP	COVERAGE	FUEL	USE	FLEET	SEX	AGEPH	BM	AGEC	POWER	PC	TOW
1	1	1618.0010	1618.0010	1.0000000	TPL	gasoline	private	N	male	50	5	12	77	1000	BRU
2	0	0.0000	NA	1.0000000	PO	gasoline	private	N	female	64	5	3	66	1000	BRU
3	0	0.0000	NA	1.0000000	TPL	diesel	private	N	male	60	0	10	70	1000	BRU
4	0	0.0000	NA	1.0000000	TPL	gasoline	private	N	male	77	0	15	57	1000	BRU
5	1	155.9746	155.9746	0.0465753	TPL	gasoline	private	N	female	28	9	7	70	1000	BRU
6	0	0.0000	NA	1.0000000	TPL	gasoline	private	N	male	26	11	12	70	1000	BRU

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)
```

Check `rename_all()` in {dplyr}.

```
dim(mtpl)
```

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

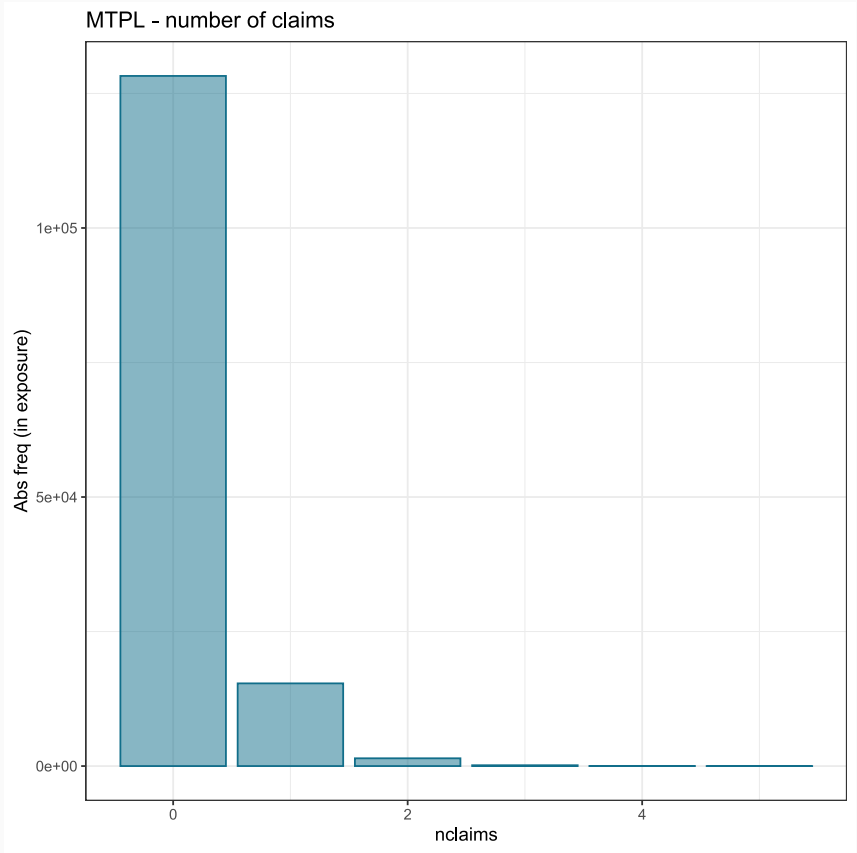
```
mtpl %>% summarize(emp_freq =
                     sum(nclaims) / sum(expo))
```

emp_freq
0.1393352

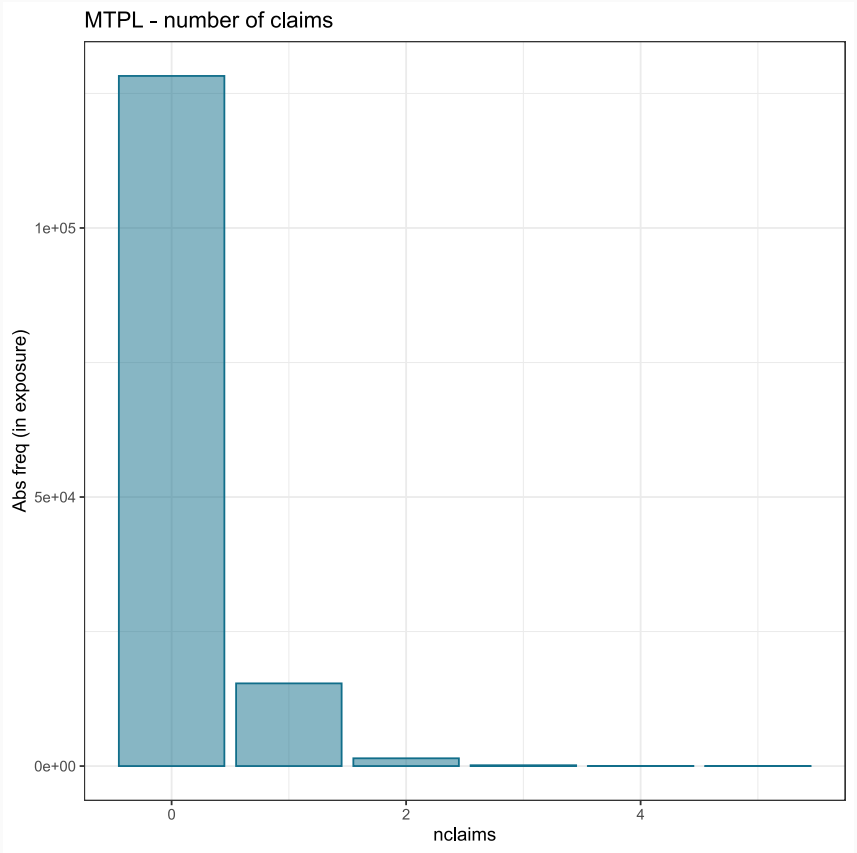
```
mtpl %>%
  group_by(sex) %>%
  summarize(emp_freq = sum(nclaims) / sum(expo))
```

sex	emp_freq
female	0.1484325
male	0.1361164

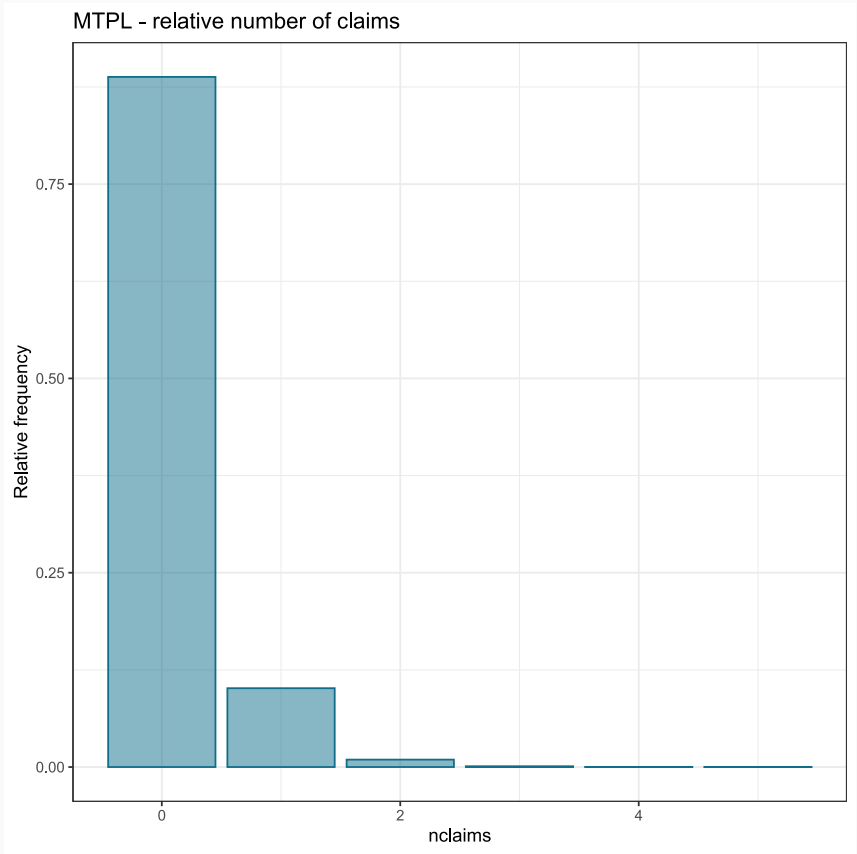
```
g ← ggplot(mtpl, aes(nclaims)) + theme_bw() +
    geom_bar(aes(weight = expo), col = KULbg,
             fill = KULbg, alpha = 0.5) +
    labs(y = "Abs freq (in exposure)") +
    ggtitle("MTPL - number of claims")
g
```



```
g ← ggplot(mtpl, aes(nclaims)) + theme_bw() +
    geom_bar(aes(weight = expo), col = KULbg,
             fill = KULbg, alpha = 0.5) +
    labs(y = "Abs freq (in exposure)") +
    ggtitle("MTPL - number of claims")
g
```



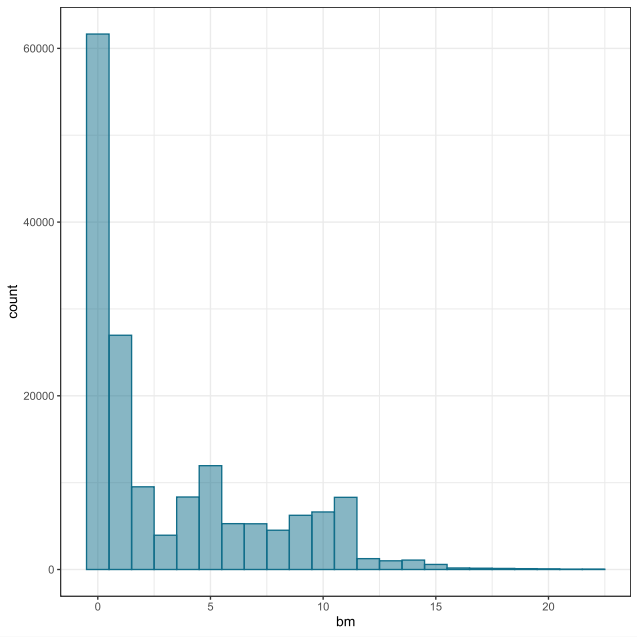

```
g ← ggplot(mtpl, aes(nclaims)) + theme_bw()
g + geom_bar(aes(y = (..count..)/sum(..count..)),
    col = KULbg, fill = KULbg, alpha = 0.5) +
  labs(y = "Relative frequency") +
  ggtitle("MTPL - relative number of claims")
```



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

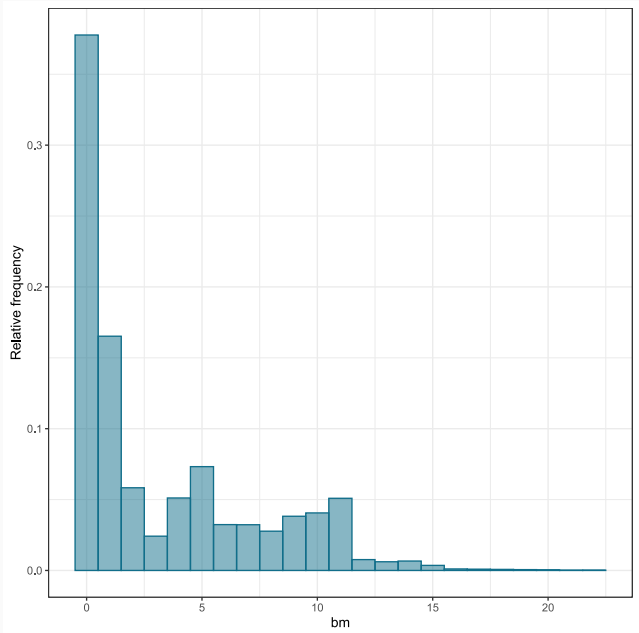
Here is the histogram of `bm` showing how the policyholders are distributed over levels in the bonus-malus scale:

```
g ← ggplot(mtpl, aes(bm)) + theme_bw()
g + geom_histogram(binwidth = 1, col = KULbg,
                   fill = KULbg, alpha = 0.5)
```



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 = 0.5) +
  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.

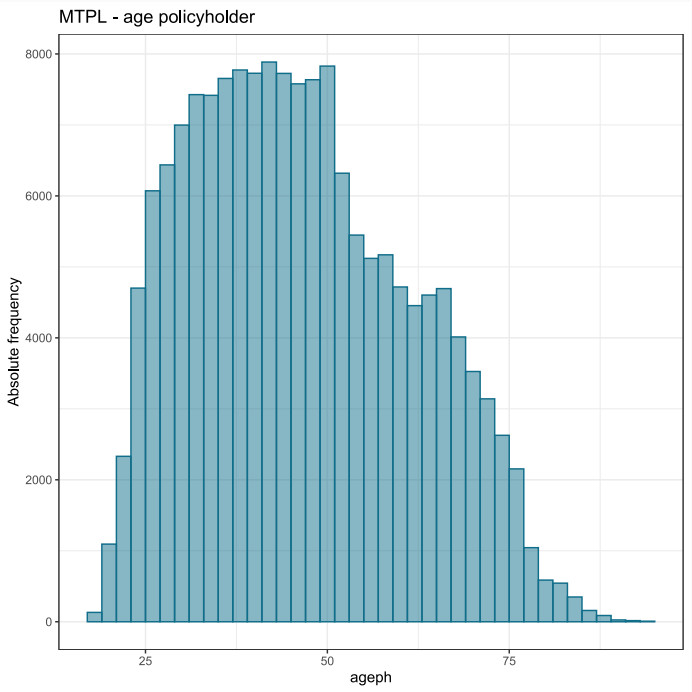
Q: you will work through the following exploratory steps.

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 = 0.5) +
      labs(y = "Absolute frequency") +
      ggtitle("MTPL - age policyholder")
```



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

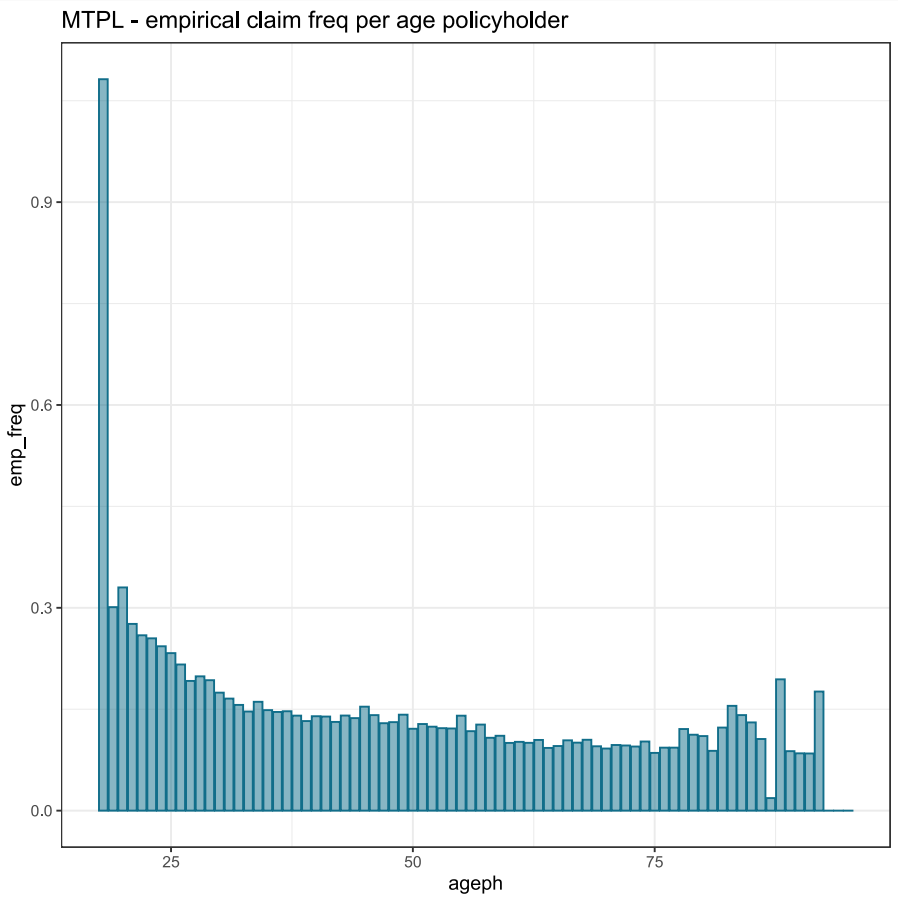
```
mtpl %>%
  group_by(ageph) %>%
  summarize(tot_claims = sum(nclaims),
            tot_expo = sum(expo), tot_obs = n())
```

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 = 0.5) +
  ggtitle("MTPL - empirical claim freq per
                                age policyholder")
```



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

Data sets used in this session - Housing data

We will use the Ames Iowa housing data. There are 2,930 properties in the data set.

The `Sale_Price` (target or response) was recorded along with 80 predictors, including:

- location (e.g. neighborhood) and lot information
- house components (garage, fireplace, pool, porch, etc.)
- general assessments such as overall quality and condition
- number of bedrooms, baths, and so on.

More details in De Cock (2011, Journal of Statistics Education).

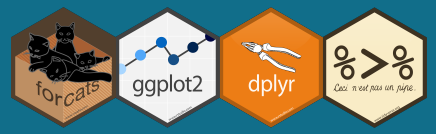
The raw data are at `http://bit.ly/2whgsQM` but we will use a processed version found in the `AmesHousing` package.

You will load the data with the `make_ames()` function from the `AmesHousing` library, and store the data in the object `ames`:

```
ames ← AmesHousing::make_ames()
```

You can now build your own, first exploration of the response variable and the covariates.

Data sets used in this session - AOV data



In your assignment you will be working with country-level data on becoming disabled (*invalidering*) and recovery (*revalidering*).

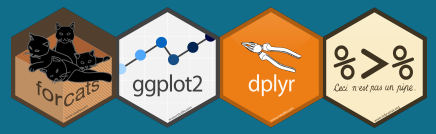
These data are available in .csv format, see the course material for a detailed description.

```
# install.packages("here")
library(here)
dir ← here::here()
setwd(dir)
invalidering ← read.csv('./data/invalidering landelijke data 2019.csv',
                  header = TRUE, sep = ";")
revalidering ← read.csv('./data/revalidering landelijke data 2019.csv',
                  header = TRUE, sep = ";")
```

To get a first feel for these data, think about the following questions:

- why does `invalidering` have 890 rows?
- for which years are data available?

Data sets used in this session - AOV data



Next to the country-level data, you will be working with the policy and claims data from insurance company ABC:

```
# install.packages("here")
library(here)
dir ← here::here()
setwd(dir)
polis ← read.csv('./data/polissen verzekeraar ABC 2019.csv',
                 header = TRUE, sep = ";", dec = ",")
schade ← read.csv('./data/schades verzekeraar ABC 2019.csv',
                 header = TRUE, sep = ";", dec = ",")
```

To get a first feel for these data, think about the following questions:

- what would be a meaningful ordering of the data, to better understand the structure?
- for which years are data available? What do you notice?

Your turn

Let's load the `invalidering` country-level data set and do some basic investigations into the variables. The idea is to get a feel for the data.

Q: you will work through the following preparatory steps.

1. In the `invalidering` data set change the variable names as follows: `Beschouwd.Jaar` to `jaar`, `Beroepsklasse` to `beroep`, `Leeftijdsklasse` to `leeft_kl`, `Aantal.validen` to `aant_val`, `Verzekerde.rente.in.euro.s` to `verz_rente` and `Som.uitkeringspercentage.met.ziektedatum.in.de.eerste.9.maanden.van.jaar.t` to `som_uitk_perc`. You can use `rename()` from {dplyr}.

2. Change all variable names to lower case, if necessary.

3. Explore the structure of the data set, e.g. number of observations, types of variables etc.

10:00

For **Q.1** combined with **Q.2** you can do

```
invalidering ← as_tibble(invalidering)
invalidering ← invalidering %>% rename(jaar = Beschouwd.Jaar, beroep = Beroepsklasse,
                                       leeft_kl = Leeftijdsklasse, aant_val = Aantal.validen,
                                       verz_rente = Verzekerde.rente.in.euro.s,
                som_uitk_perc = Som.uitkeringspercentage.met.ziektedatum.in.de.eerste.9.maanden.van.jaar.t)
```

```
str(invalidering) %>% kable(format = 'html')
## tibble [810 x 6] (S3: tbl_df/tbl/data.frame)
##  $ jaar         : int [1:810] 2004 2004 2004 2004 2004 2004 2004 2004 2004 2004 ...
##  $ beroep       : int [1:810] 1 1 1 1 1 1 1 1 1 2 ...
##  $ leeft_kl     : chr [1:810] "15 tm 25" "26 tm 30" "31 tm 35" "36 tm 40" ...
##  $ aant_val     : int [1:810] 291 1664 4130 4195 3077 3872 2333 1656 176 598 ...
##  $ verz_rente   : int [1:810] 2236436 19446219 58985108 84335669 58859278 79725314 42002031 24545951 2959686 49578
##  $ som_uitk_perc: int [1:810] 0 564 2495 2029 2936 2990 1572 815 0 314 ...
```

=

Before you get started, which years, professions and age classes are in this data set?

In base R you would do something like ...

```
unique(invalidering$jaar)
##  [1] 2004 2005 2006 2007 2008 2009 2010 2011 2012 20
unique(invalidering$beroep)
## [1] 1 2 3 4 5 6
unique(invalidering$leeft_kl)
## [1] "15 tm 25"  "26 tm 30"  "31 tm 35"  "36 tm 40"
## [7] "51 tm 55"  "56 tm 60"  "61 tot 65"
```

Thus, the data `invalidering` has 15 x 6 x 9 rows, one per year, profession and age group.

If you want to stick to {dplyr} instructions, you can go for:

```
invalidering %>% distinct(jaar)
```

jaar
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014

You probably want to treat the variables `beroep` and `leeft_kl` as factor variables.

```
invalidering ← invalidering %>% mutate(beroep = as.factor(beroep),
                                       leeft_kl = as.factor(leeft_kl))
```

Check the structure of the data set again.

The functions in the {forcats} package (part of the tidyverse) are handy when working with factor variables, e.g.

```
invalidering ← invalidering %>% mutate(leeft_kl = fct_recode(leeft_kl, "61 tm 65" = "61 tot 65"))
```

to recode the last level of `leeft_kl`.

In the course we introduced the following definition of the unweighted ('ongewogen') disability probabilities ('invalideringskansen')

$$i_{x,t} = \frac{\sum_{\text{groep } x,t} \text{uitk percentage}}{\sum_{\text{groep } x,t} \text{validen}},$$

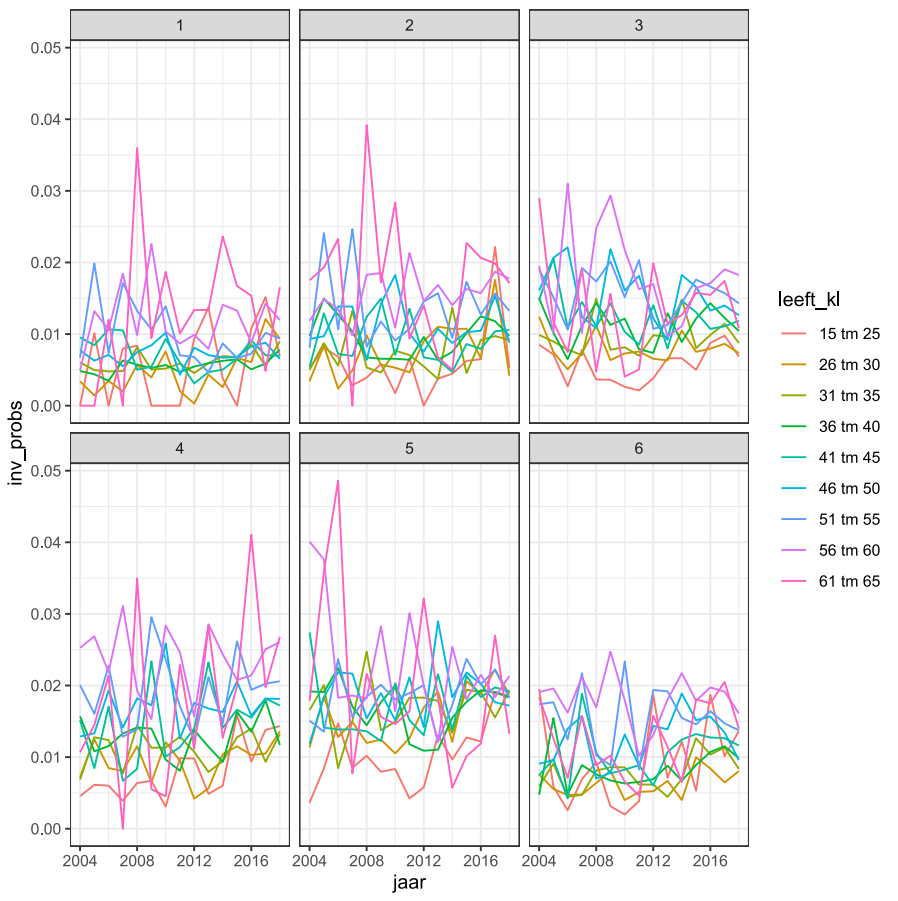
where 'uitkeringspercentage' refers to 'ultimo jaar t, invalidering in eerste 9 maanden'.

Let's put this into practice: (recall the definition of variable `som_uitk_perc` from the course documentation)

```
invalidering ← mutate(invalidering,
          inv_probs = som_uitk_perc/aant_val/100)
```

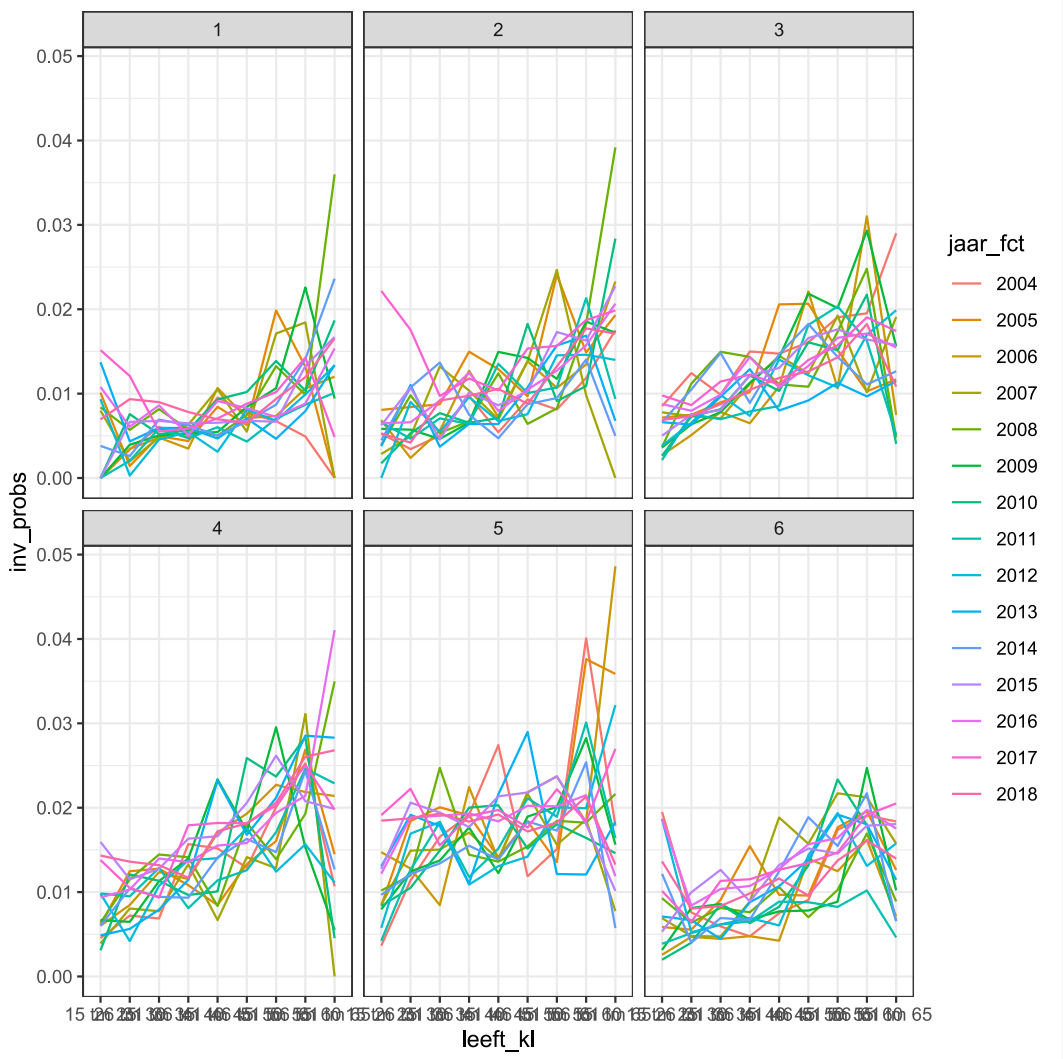
Now let's do a first plot with `jaar` on the x-axis and one line per `leeft_kl`, one panel per `beroep`

```
ggplot(invalidering) +
  geom_line(aes(x = jaar, y = inv_probs,
                color = leeft_kl)) +
  facet_wrap( ~ beroep, ncol = 3) + theme_bw()
```



Or a plot with `leeft_kl` on the x-axis, one line per `jaar` and one panel per `beroep`

```
invalidering ← invalidering %>%
            mutate(jaar_fct = as.factor(jaar))
ggplot(invalidering) +
  geom_line(aes(x = leeft_kl, y = inv_probs,
            color = jaar_fct, group = jaar_fct)) +
            facet_wrap( ~ beroep, ncol = 3) +
            theme_bw()
```

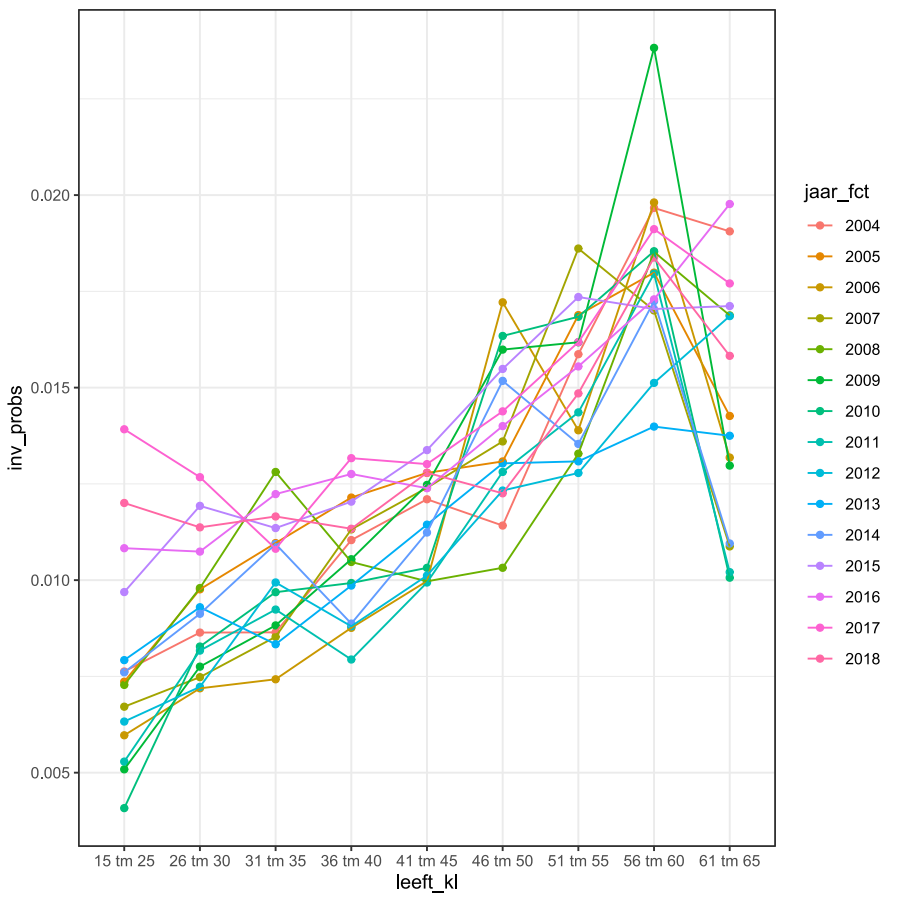


Instead of working per year, age group and profession, you can aggregate the data before calculating the disability probabilities, e.g.

```
inv_leeft_kl ← invalidering %>%
               group_by(leeft_kl, jaar_fct) %>%
               summarize(tot_val = sum(aant_val),
                   tot_perc = sum(som_uitk_perc))
inv_leeft_kl ← mutate(inv_leeft_kl,
                   inv_probs = tot_perc/tot_val/100)
```

Playing with the {ggplot2} instructions, you can visualize the data as follows

```
ggplot(inv_leeft_kl, aes(x = leeft_kl, y = inv_probs,
      color = jaar_fct, group = jaar_fct)) +
      geom_point() + geom_line() + theme_bw()
```



Your turn

Let's load the `polis` and `schade` data and do some basic investigations into the variables. The idea is to get a feel for the data.

Q: you will work through the following preparatory steps.

1. In the `polis` data set change the variable names `Verzekerde.rente` to `verz_rente`, `minstens.3.mnd.invalide` to `3_mnd_inv` and `wachttijd.in.dagen` into `wcht_dagen`. Or use your own preferred set of variable names. You can use `rename()` from {dplyr}.

2. In the `schade` data set change the variable names `verz.rente` to `verz_rente`, `uitkeringspercentage` to `uitk_perc` and `wachttijd.in.dagen` into `wcht_dagen` and `Ingangsdatum.uitkering.behorende.bij.schadejaar` to `ing_datum`. Or use your own preferred set of variable names.

3. Change all variable names to lower case.

4. Explore the structure of both data sets, e.g. number of observations, types of variables etc.

10:00

For **Q.1** combined with **Q.3** you can do

```
polis ← as_tibble(polis)
polis ← polis %>% rename(verz_rente = Verzekerde.rente, wcht_dagen = wachttijd.in.dagen,
                 drie_mnd_inv = minstens.3.mnd.invalide)
```

Change the variable names to lower case using the `rename_all()` instruction introduced in the MTPL example.

```
str(polis) %>% kable(format = 'html')
## tibble [162,281 x 9] (S3: tbl_df/tbl/data.frame)
##  $ index       : int [1:162281] 1 2 3 4 5 6 7 8 9 10 ...
##  $ leeftijd    : int [1:162281] 52 52 46 52 40 40 47 60 54 41 ...
##  $ geslacht    : chr [1:162281] "M" "M" "M" "M" ...
##  $ beroep      : int [1:162281] 4 3 4 4 2 4 4 4 4 3 ...
##  $ verz_rente  : int [1:162281] 5502 4540 4845 9624 84911 25043 5308 3786 3220 1902 ...
##  $ jaar        : int [1:162281] 2004 2004 2004 2004 2004 2004 2004 2004 2004 2004 ...
##  $ duur        : num [1:162281] 1 1 1 1 1 1 1 1 1 1 ...
##  $ drie_mnd_inv: int [1:162281] 0 0 0 0 0 0 0 1 1 0 ...
##  $ wcht_dagen  : int [1:162281] 7 7 14 180 360 360 14 14 30 14 ...
```

=

For **Q.2** combined with **Q.3** you can do

```
schade ← as_tibble(schade)
schade ← schade %>% rename(verz_rente = verz.rente, uitk_perc = uitkeringspercentage,
                           wcht_dagen = wachttijd.in.dagen,
                           ing_datum = Ingangsdatum.uitkering.behorende.bij.schadejaar)
```

Change the variable names to lower case using the `rename_all()` instruction introduced in the MTPL example.

```
str(schade) %>% kable(format = 'html')
## tibble [8,291 x 11] (S3: tbl_df/tbl/data.frame)
##  $ index      : int [1:8291] 1 1 1 1 1 1 1 1 1 1 ...
##  $ geslacht   : chr [1:8291] "M" "M" "M" "M" ...
##  $ verz_rente : int [1:8291] 5502 5502 5502 5502 5502 5502 5502 5502 5502 5502 ...
##  $ uitk_perc  : num [1:8291] 0.72 0.72 0.72 0.72 0.72 0.72 0.72 0.72 0.72 0.72 ...
##  $ jaar       : int [1:8291] 2017 2016 2015 2014 2013 2012 2011 2010 2009 2008 ...
##  $ wcht_dagen : int [1:8291] 7 7 7 7 7 7 7 14 7 7 ...
##  $ schadejaar : int [1:8291] 2006 2006 2006 2006 2006 2006 2006 2006 2006 2006 ...
##  $ schademaand: int [1:8291] 3 3 3 3 3 3 3 3 3 3 ...
##  $ leeftijd   : int [1:8291] 52 52 52 52 52 52 52 52 52 52 ...
##  $ beroep     : int [1:8291] 4 4 4 4 4 4 4 4 4 4 ...
##  $ ing_datum  : int [1:8291] 2007 2007 2007 2007 2007 2007 2007 2007 2007 2007 ...
```

=

First, check the years during which observations are recorded in ABC's portfolio

```
unique(polis$jaar)
##  [1] 2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016 2017 2018
unique(schade$jaar)
##  [1] 2017 2016 2015 2014 2013 2012 2011 2010 2009 2008 2007 2006 2018 2005
```

Thus, you have data for 2004 in the policy but not in the claims data set.

It is insightful to order `polis` by `index` and `jaar`

```
polis ← polis %>% arrange(index, jaar)
```

and `schade` by `index`, `schadejaar` and `jaar`

```
schade ← schade %>% arrange(index, schadejaar, jaar)
```

Inspect the data using `View()`.

How about duplicate records, e.g. same `index` and `jaar` in `polis` data set?

```
polis %>%
  group_by(index, jaar) %>%
  filter(n() > 1)
```

... and same `index`, `schadejaar` and `jaar` in `schade` data set?

```
schade %>%
  group_by(index, schadejaar, jaar) %>%
  filter(n() > 1)
```

You will have to propose some data cleaning steps to remove these duplicate observations.

How about the unique claims in `schade`?

```
schade ← schade %>%
  mutate(dev_jaar = jaar - ing_datum + 1)
```

```
schade_unique ← schade %>% group_by(index, schadejaar) %>% filter(row_number() == 1)
schade_unique ← unique(schade_unique$index)
```

and the unique `polis` numbers

```
polis_unique ← unique(polis$index)
```

Now check (or the other way around, to collect the policies that do not report a claim)

```
length(which(! schade_unique %in% polis_unique))
## [1] 0
```

It might also be insightful to create new variables indicating the `start` and `end` year of a claim.

```
schade ← schade %>% group_by(index, schadejaar) %>%
                    mutate(first_occ = ifelse(row_number() == 1, 1, 0),
                           last_occ = ifelse(row_number() == n(), 1, 0))
```

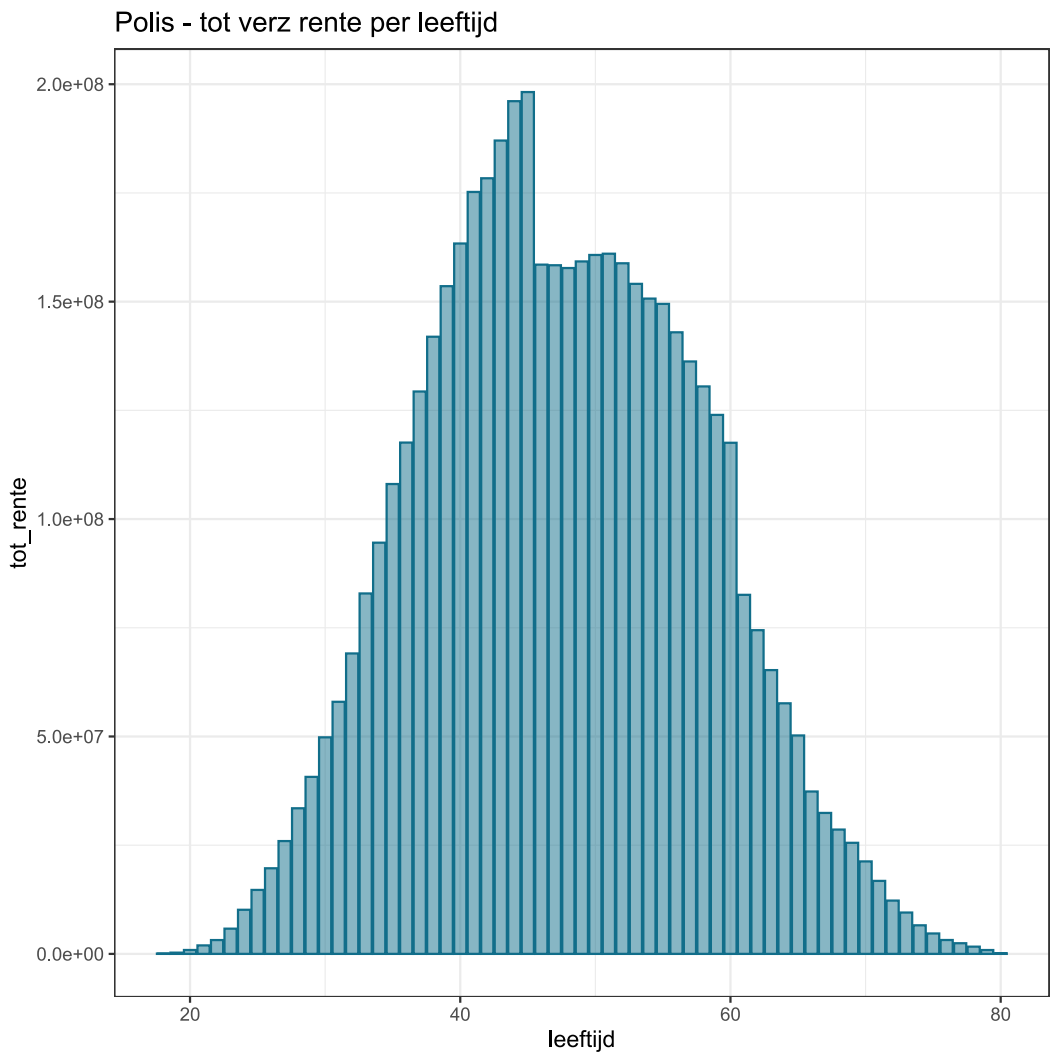
Verify what these instructions do!

You can do a first exploration of the `polis` data with instructions similar to those used on the MTPL data.

```
verz_by_leeft ← polis %>%
  group_by(leeftijd) %>%
  summarize(tot_rente = sum(verz_rente),
            tot_obs = n(),
            tot_duur= sum(duur),
            tot_inv = sum(drie_mnd_inv),
            rel_dis = sum(drie_mnd_inv)/tot_obs,
            avg_rente = tot_rente/tot_obs)
```

And then you can plot for instance

```
ggplot(verz_by_leeft,
       aes(x = leeftijd, y = tot_rente)) +
  theme_bw() +
  geom_bar(stat = "identity", color = KULbg,
           fill = KULbg, alpha = 0.5) +
  ggtitle("Polis - tot verz rente per leeftijd")
```

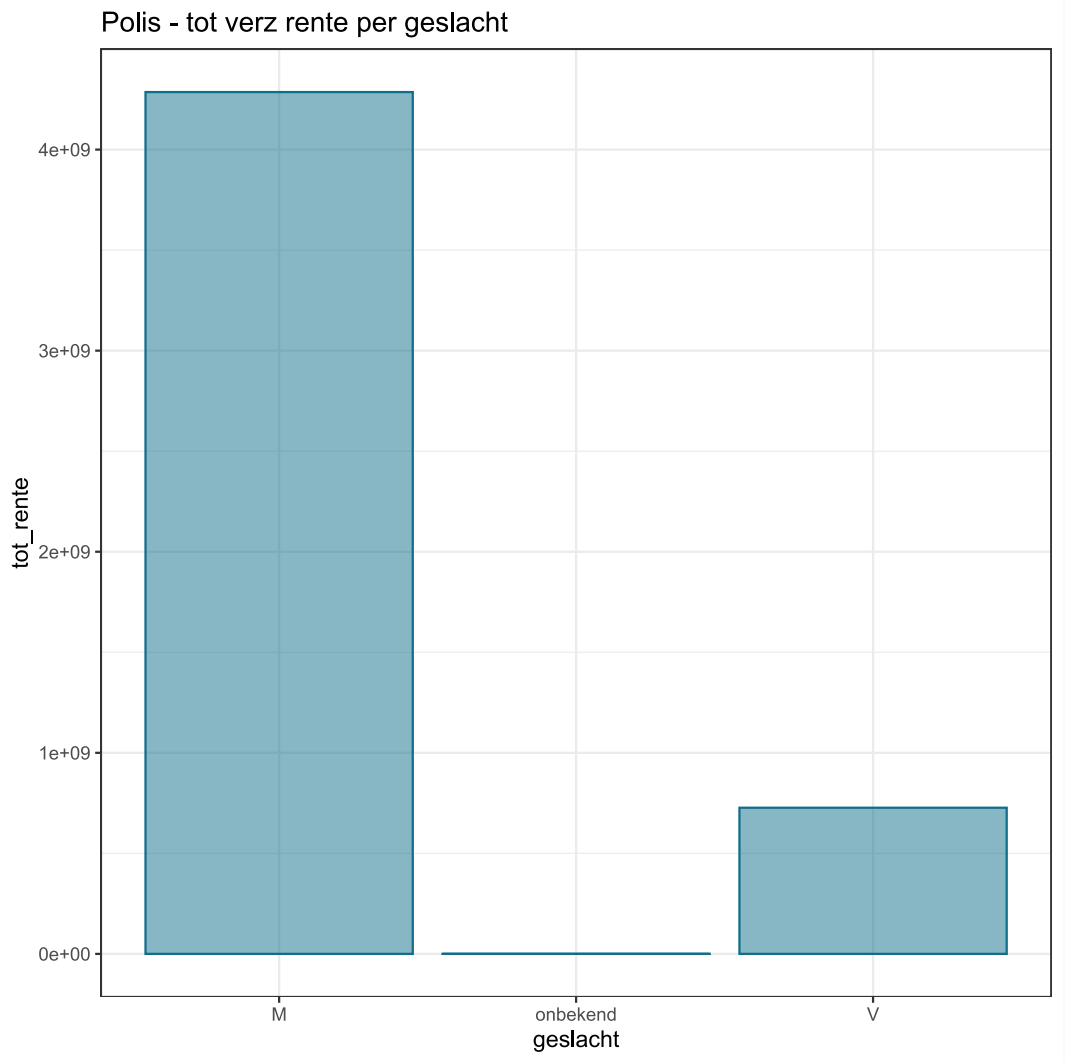


You can do a first exploration of the `polis` data with instructions similar to those used on the MTPL data.

```
verz_by_geslacht ← polis %>%
  group_by(geslacht) %>%
  summarize(tot_rente = sum(verz_rente),
            tot_obs = n(),
            tot_duur= sum(duur),
            tot_inv = sum(drie_mnd_inv),
            rel_dis = sum(drie_mnd_inv)/tot_obs,
            avg_rente = tot_rente/tot_obs) %>%
  mutate(freq = prop.table(tot_obs))
```

And then you can plot for instance

```
ggplot(verz_by_geslacht,
       aes(x = geslacht, y = tot_rente)) +
  theme_bw() +
  geom_bar(stat = "identity", color = KULbg,
           fill = KULbg, alpha = 0.5) +
  ggtitle("Polis - tot verz rente per geslacht")
```

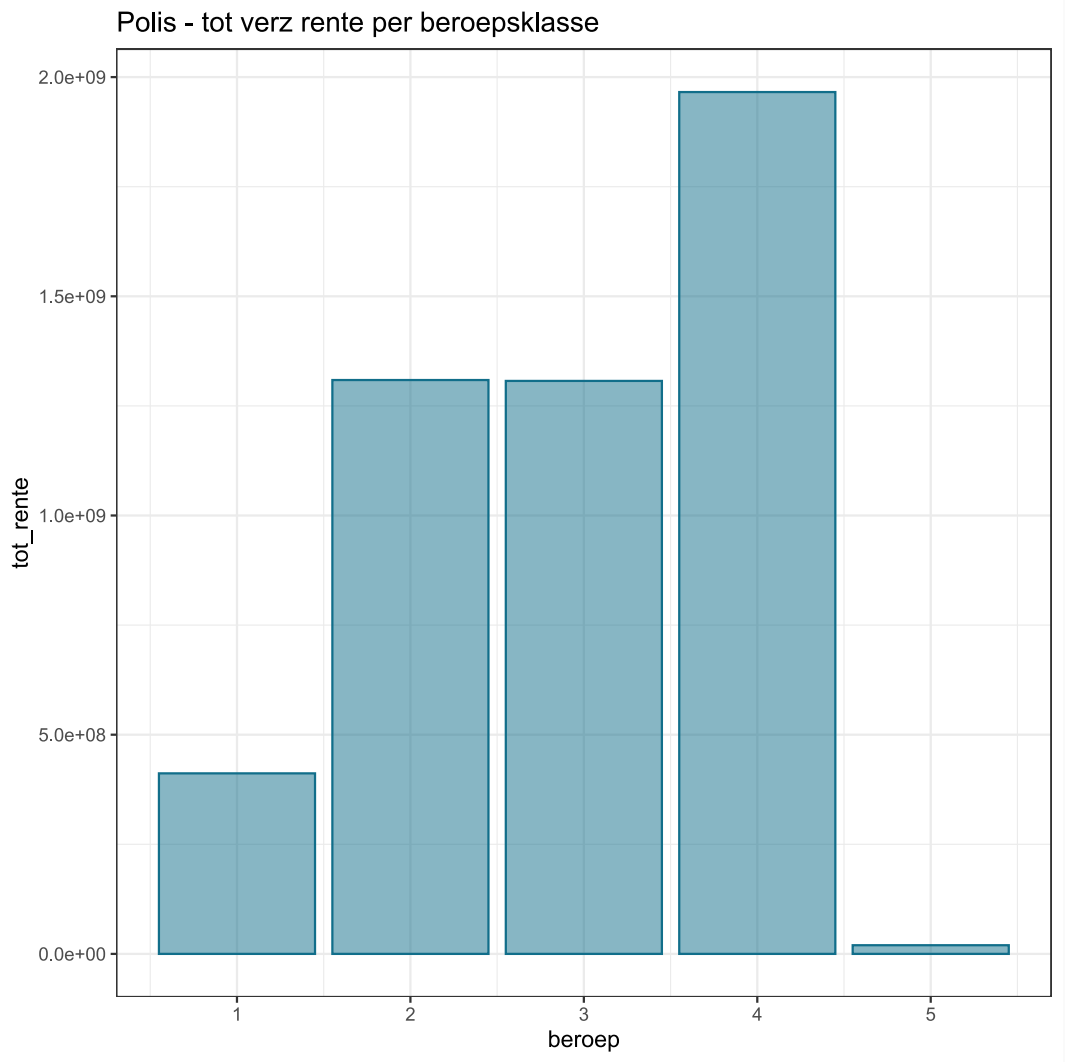


You can do a first exploration of the `polis` data with instructions similar to those used on the MTPL data.

```
verz_by_beroep ← polis %>%
  group_by(beroep) %>%
  summarize(tot_rente = sum(verz_rente),
            tot_obs = n(),
            tot_duur= sum(duur),
            tot_inv = sum(drie_mnd_inv),
            rel_dis = sum(drie_mnd_inv)/tot_obs,
            avg_rente = tot_rente/tot_obs) %>%
  mutate(freq = prop.table(tot_obs))
```

And then you can plot for instance

```
ggplot(verz_by_beroep,
       aes(x = beroep, y = tot_rente)) +
  theme_bw() +
  geom_bar(stat = "identity", color = KULbg,
           fill = KULbg, alpha = 0.5) +
  ggtitle("Polis - tot verz rente per beroepsklasse")
```



Note that `leeftijd` is registered in the `polis` data as a numeric variable.

How can you cluster or bin `leeftijd` into age classes (cfr. the country-level `invalidering`)?

```
leeft_breaks ← c(15, 25, 30, 35, 40, 45, 50, 55, 60, 65)
leeft_labels ← unique(invalidering$leeft_kl)
polis ← polis %>% mutate(leeftijd_fct = cut(leeftijd, breaks = leeft_breaks, labels = leeft_labels))
```

Now you can repeat the previous visualizations using the factor variable `leeftijd_fct`.

Let's demonstrate how to subset your data

```
polis_select ← polis %>% filter(jaar > 2004)
schade_select ← schade %>%
                    select(index, uitk_perc, jaar, schadejaar,
                           ing_datum, dev_jaar, first_occ, last_occ)
```

Try to join both data sets with instructions like these

```
test ← left_join(polis_select, schade_select, by = c("index" = "index", "jaar" = "jaar"))
```

Many other `join` type of instructions are available in {dplyr}.

Building linear and generalized linear models in R

Creating models in R

We introduce and illustrate model building syntax on the famous `ames` housing data.

The **formula** interface using R's formula rules to specify a *symbolic* representation of the terms:

- response ~ variable, with `model_fn` referring to the specific model function you want to use, e.g. `lm` for linear regression

```
model_fn(Sale_Price ~ Gr_Liv_Area, data = ames)
```

- response ~ variable_1 + variable_2

```
model_fn(Sale_Price ~ Gr_Liv_Area + Neighborhood, data = ames)
```

- response ~ variable_1 + variable_2 + their interaction

```
model_fn(Sale_Price ~ Gr_Liv_Area + Neighborhood + Neighborhood:Gr_Liv_Area, data = ames)
```

- shorthand for all predictors

```
model_fn(Sale_Price ~ ., data = ames)
```

Your turn

You will now fit some linear regression models on the `ames` housing data.

You will explore the model fits with `base` R instructions as well as the functionalities offered by the `broom` package.

Q: load the `ames` housing data set via `ames ← AmesHousing::make_ames()`

1. Fit a linear regression model with `Sale_Price` as response and `Gr_Liv_Area` as covariate. Store the resulting object as `model_1`.
2. Repeat your instruction, but now put it between brackets. What happens?
3. Inspect `model_1` with the following set of instructions

- `summary(___)`
- extract the fitted coefficients, using `___$coefficients`
- what happens with `summary(___)$coefficients`?
- extract fitted values, using `___$fitted.values`
- now try to extract the R^2 of this model.

Q.1 Linear model with `Sale_Price` as a function of `Gr_Live_Area`

```
model_1 ← lm(Sale_Price ~ Gr_Liv_Area, data = ames)
```

Q.3 Check `model_1` - What happens - do you *like* this display?

```
summary(model_1)
```

Now let's extract some meaningful information from `model_1` (using `base` R instructions)

```
model_1$coefficients
```

```
## (Intercept) Gr_Liv_Area
##   13289.634     111.694
```

```
summary(model_1)$coefficients
```

```
##              Estimate  Std. Error   t value     Pr(>
## (Intercept) 13289.634 3269.702768  4.064478 4.940672
## Gr_Liv_Area   111.694    2.066073 54.061006 0.000000
```

```
head(model_1$fitted.values)
```

```
##        1        2        3        4        5
## 198254.9 113367.5 161731.0 248964.0 195239.2 192446.
```

```
summary(model_1)$r.squared
```

```
## [1] 0.4995379
```

Tidy model output

The package {broom} allows to summarize key information about statistical objects (e.g. a linear regression model) in so-called tidy tibbles.

This makes it easy to report results, create plots and consistently work with large numbers of models at once.

We briefly illustrate the three essential verbs of `broom`: `tidy()`, `glance()` and `augment()`.

```
model_1 %>% broom::tidy()
```

term	estimate	std.error	statistic	p.value
(Intercept)	13289.634	3269.702768	4.064478	4.94e-05
Gr_Liv_Area	111.694	2.066073	54.061006	0.00e+00

```
model_1 %>% broom::glance()
```

r.squared	adj.r.squared	sigma	statistic	p.value	df	logLik	AIC	BIC	deviance	df.residual	nobs
0.4995379	0.4993669	56524.17	2922.592	0	1	-36217.79	72441.58	72459.53	9.354907e+12	2928	2930

Tidy model output

The package {broom} allows to summarize key information about statistical objects (e.g. a linear regression model) in so-called tidy tibbles.

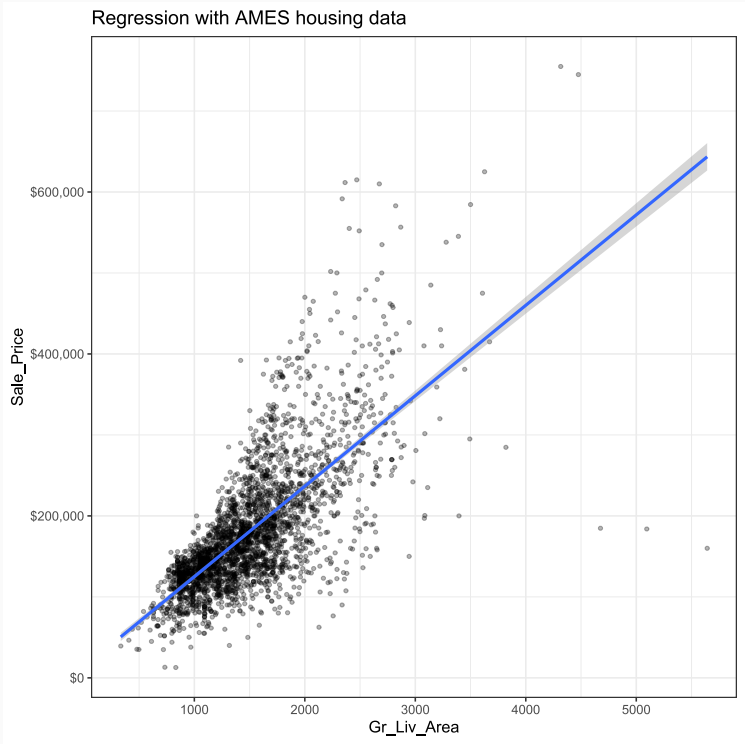
This makes it easy to report results, create plots and consistently work with large numbers of models at once.

We briefly illustrate the three essential verbs of `broom`: `tidy()`, `glance()` and `augment()`.

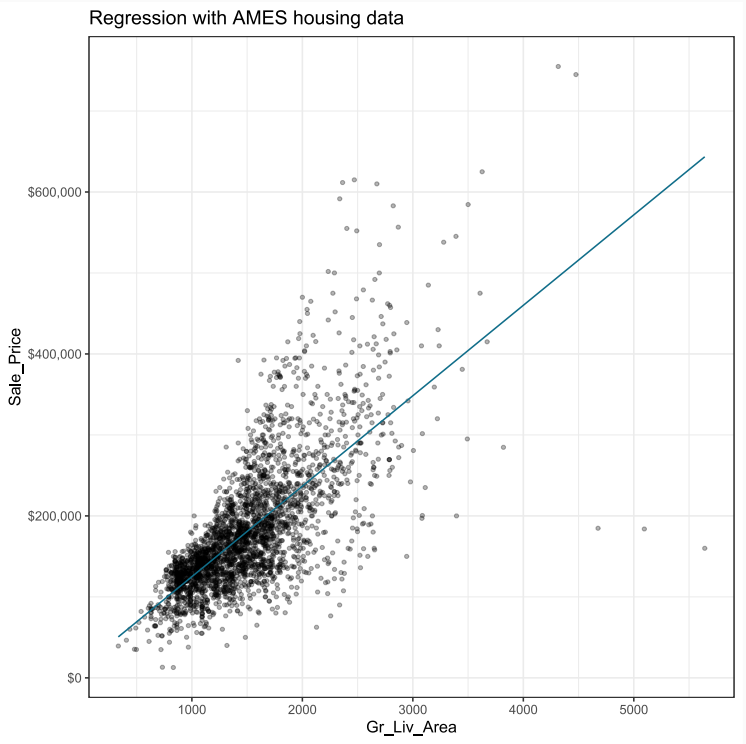
```
model_1 %>% broom::augment() %>% slice(1:5)
```

Sale_Price	Gr_Liv_Area	.fitted	.resid	.std.resid	.hat	.sigma	.cooksd
215000	1656	198254.9	16745.100	0.2963021	0.0003739	56532.98	1.64e-05
105000	896	113367.5	-8367.459	-0.1480946	0.0008282	56533.61	9.10e-06
172000	1329	161731.0	10269.038	0.1817097	0.0003802	56533.51	6.30e-06
244000	2110	248964.0	-4963.976	-0.0878573	0.0008389	56533.75	3.20e-06
189900	1629	195239.2	-5339.162	-0.0944752	0.0003636	56533.74	1.60e-06

```r
g_lm_1 ← ggplot(data = ames,
                aes(Gr_Liv_Area, Sale_Price)) +
  theme_bw() +
  geom_point(size = 1, alpha = 0.3) +
  geom_smooth(se = TRUE, method = "lm") +
  scale_y_continuous(labels = scales::dollar) +
  ggtitle("Regression with AMES housing data")
g_lm_1
```

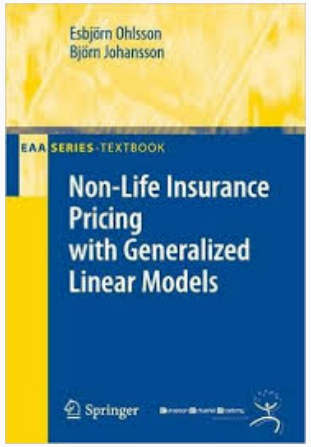


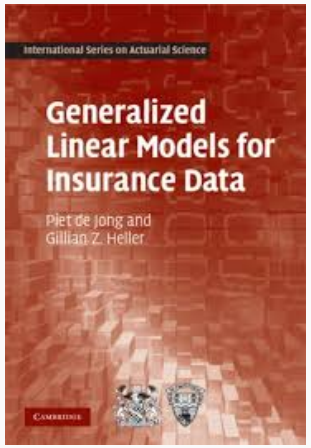

```r
g_lm_2 ← model_1 %>% broom::augment() %>%
ggplot(aes(Gr_Liv_Area, Sale_Price)) +
    theme_bw() +
    geom_point(size = 1, alpha = 0.3) +
    geom_line(aes(y = .fitted), col = KULbg) +
    scale_y_continuous(labels = scales::dollar) +
    ggtitle("Regression with AMES housing data")
g_lm_2
```



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

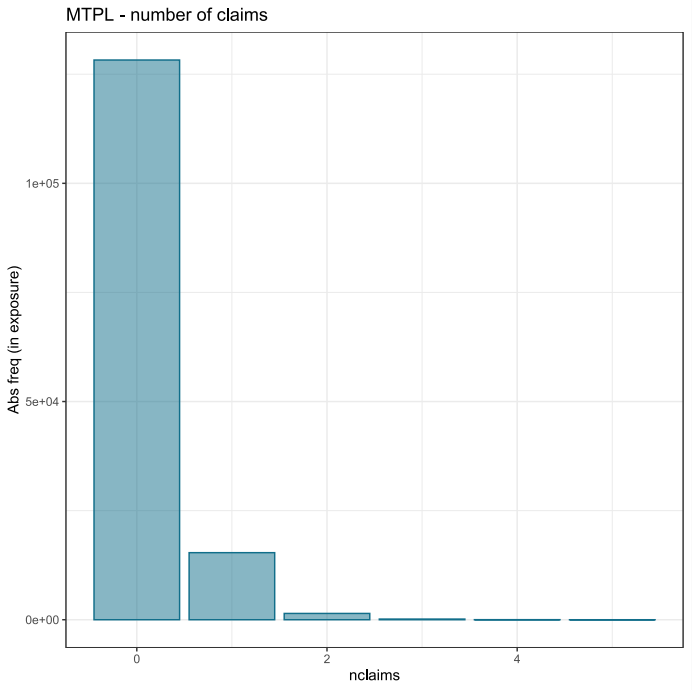
$$g(E[Y]) = x^{'}\beta.$$

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

Generalized Linear Models (GLMs)

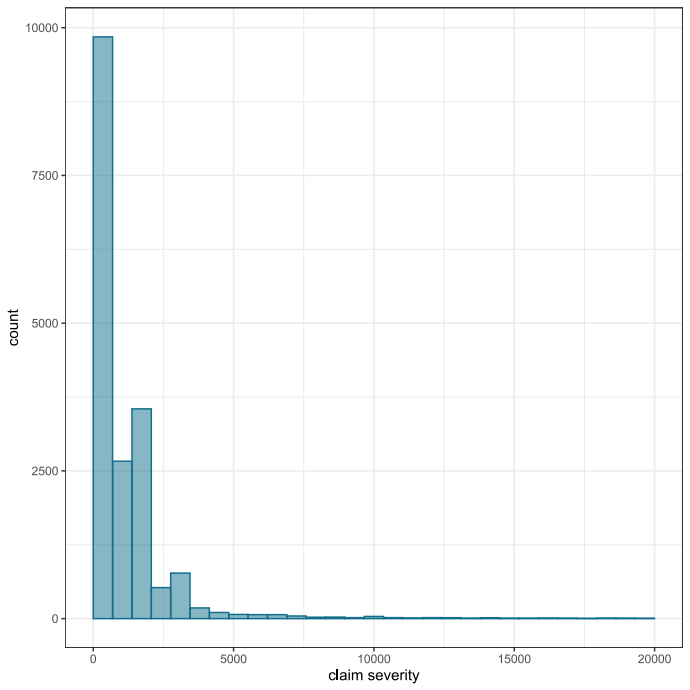
We return to the `mtpl` data set.

Target variable `nclaims` (frequency)



Suitable distributions: Poisson, Negative Binomial.

... and `avg` (severity).



Suitable distributions: log-normal, gamma.

A Poisson GLM

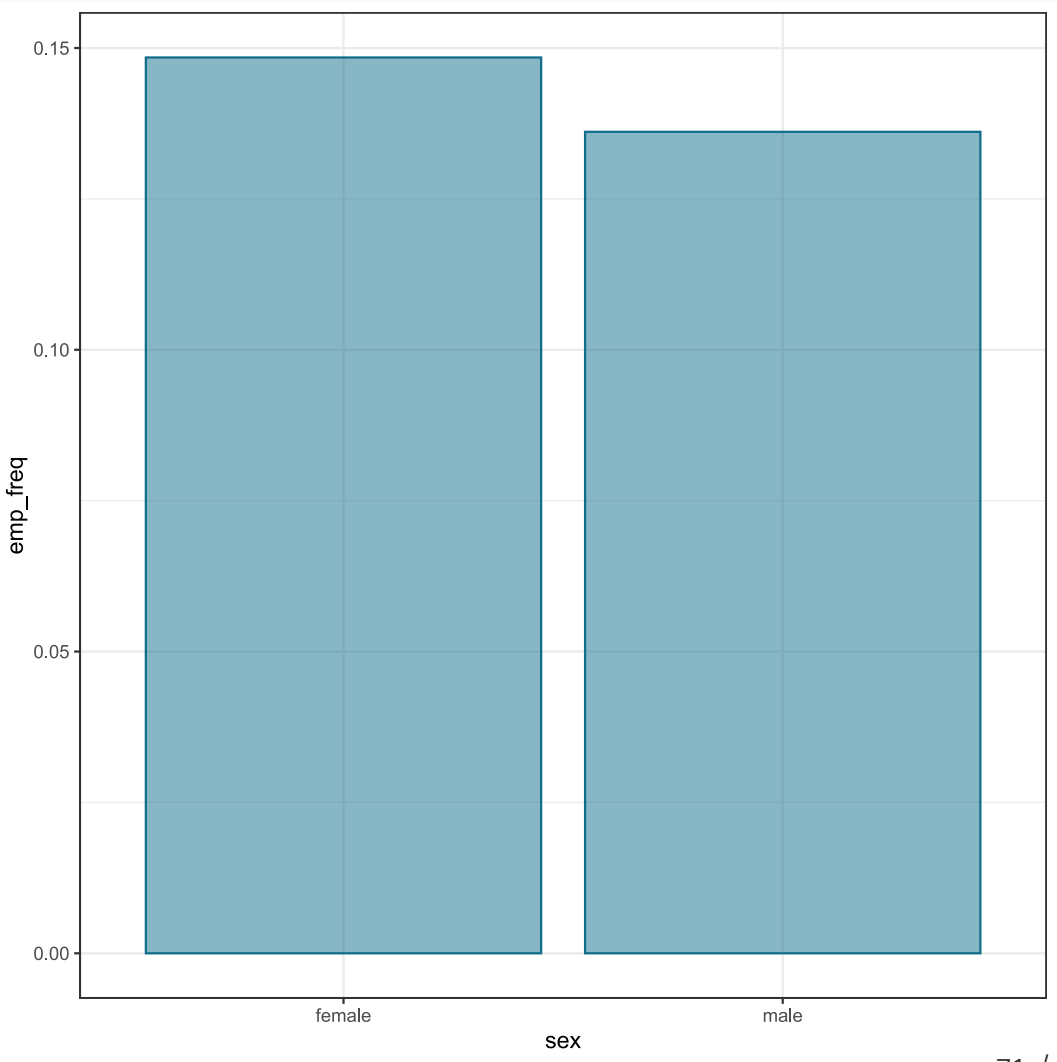
A brief recap..

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

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

Then,

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

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.

First, we inspect the output stored in `freq_glm_1` in the usual way: (e.g. AIC, Residual Deviance)

```
summary(freq_glm_1)
```

```
##
## Call:
## glm(formula = nclaims ~ sex, family = poisson(link = "log"),
##     data = mtpl, offset = log(expo))
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -0.5449  -0.5218  -0.5218  -0.4335   5.7100
##
## Coefficients:
##             Estimate Std. Error  z value Pr(>|z|)
## (Intercept) -1.90763    0.01332 -143.186  < 2e-16 ***
## sexmale     -0.08662    0.01568   -5.523 3.33e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 89944  on 163230  degrees of freedom
## Residual deviance: 89914  on 163229  degrees of freedom
## AIC: 127650
```

Then, we revisit the model stored in `freq_model_1` using the `broom::tidy()` and the `broom::glance()` instruction

```
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

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

null.deviance	df.null	logLik	AIC	BIC	deviance	df.residual	nobs
89944.32	163230	-63822.76	127649.5	127669.5	89914.22	163229	163231

Recall:

- different models (nested and non-nested) can be compared with AIC or BIC ('smaller is better')
- nested models can be compared with a drop-in-deviance analysis, use `anova()` with `glm()`
- need a recap? See my YouTube videos.

To illustrate variable selection steps with AIC and/or a drop-in-deviance analysis, we build a second GLM.

```
freq_glm_2 ← glm(nclaims ~ sex + coverage + use +
                           fuel + fleet,
                 offset = log(expo),
                 family = poisson(link = "log"),
                 data = mtpl)
anova(freq_glm_2)
```

What do you notice regarding the variables included in `freq_glm_2`?

```
## Analysis of Deviance Table
##
## Model: poisson, link: log
##
## Response: nclaims
##
## Terms added sequentially (first to last)
##
##
##          Df Deviance Resid. Df Resid. Dev
## NULL                    163230      89944
## sex       1   30.109    163229      89914
## coverage  2   68.584    163227      89846
## use       1    0.028    163226      89846
## fuel      1  181.007    163225      89665
## fleet     1   24.087    163224      89641
```

Now we switch to predicted (or fitted) values obtained with `freq_glm_1` (by means of example). In the traditional way:

```
predict(freq_glm_1, type = "response")[1:4]
##         1         2         3         4
## 0.1361164 0.1484325 0.1361164 0.1361164
```

and for residuals

```
resid(freq_glm_1, type = "deviance")[1:4]
##          1          2          3          4
##  1.5035700 -0.5448532 -0.5217593 -0.5217593
```

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

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

nclaims	sex	.fitted	.resid
1	male	0.1361164	1.5035700
0	female	0.1484325	-0.5448532

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?

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(expo = 1, sex = "male")
female_driver ← data.frame(expo = 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
```

Thanks!

Slides created with the R package xaringan.

Course material available via

https://github.com/katrienantonio/APC-workshop-Productontwikkeling