Loss Models : a collection of computer labs in R

Katrien Antonio, Jonas Crevecoeur and Bavo DC Campo

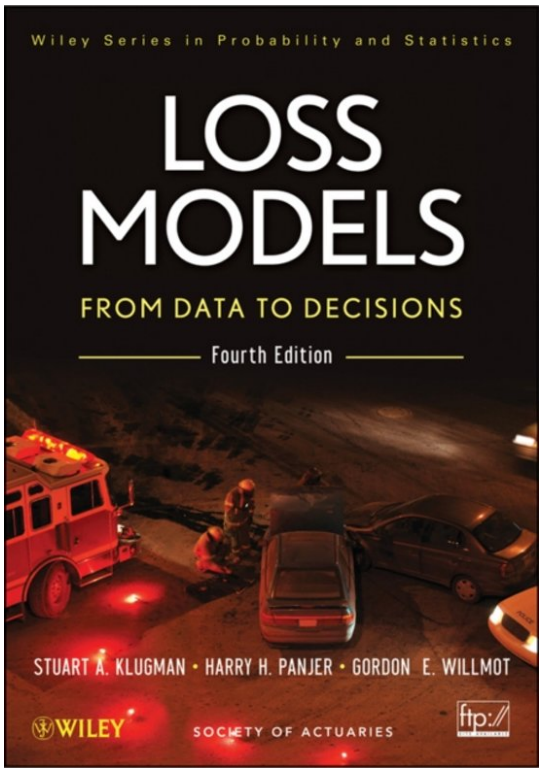
2020-10-09

Contents

Chapter 1

Intro

This book is a collection of computer labs that come with the course on Loss Models taught at KU Leuven by Katrien Antonio and Jonas Crevecoeur. The labs are inspired by the book Loss Models: from data to decsisions written by Stuart A. Klugman, Harry H. Panjer and Gordon E. Willmot and published by Wiley.



We assume basic knowledge of working with objects and data sets in R, writing functions and running optimizations.

Chapter 2

Data exploration

In this tutorial you import a dataset and perform an exploratory analysis of the variables. You will visualize the data with ggplot and calculate basic statistics such as the expected value and variance of each variable.

2.1 Importing data in R

The data for this tutorial is stored in the file CPS1985.txt, which is stored in a local subdirectory `data`. Check out chapter 4 of Data science in insurance: an R intro for a detailed overview of the R methods for importing data.

2.1.1 Determining the file path

Before we can import the data, we have to identify the file location. The file location can be specified by an absolute path or a relative path.

```
absolute_path <- "C:\\Users\\u0095171\\Dropbox\\Verzekeringen niet-leven\\Bookdown\\data\\CPS1985
relative_path <- "data\\CPS1985.txt"
```

Directories in the file path can be separated by a forward slash (`/`) or a double backward slash (`\\`).

Relative paths in R always start from your working directory. The command `getwd()` returns your current working directory.

```
getwd()
```

```
[1] "C:/Users/u0095171/Dropbox/Research AFI/R software/Bookdown/LossModels"
```

`setwd()` specifies the working directory for the current R session. Once set, all files in the working directory can be referenced by relative paths.

```
# This is the map containing all files for the tutorial
setwd("C:\\Users\\u0095171\\Dropbox\\Verzekeringen niet-leven\\Bookdown\\")

# This is a relative path from the working directory to the file we want to import
path <- "data\\CPS1985.txt"
```

To make sure that the file is located in this directory, we can check it using the command `file.exists()`.

```
file.exists(path)
```

```
[1] TRUE
```

It is often convenient to set the working directory to the location of the active R file. In Rstudio the following code retrieves the directory of the active document.

```
dir <- dirname(rstudioapi::getActiveDocumentContext()$path)
setwd(dir)
```

2.1.2 Organizing your analysis

```
CheckDir <- function(x) if(!dir.exists(x)) dir.create(x)
pathDir  = path.expand("~/AnalysisLossModels/")
pathData = paste0(pathDir, "Data/")
pathFigs = paste0(pathDir, "Figures/")
pathRes  = paste0(pathDir, "Results/")
sapply(list(pathDir, pathData, pathFigs, pathRes), CheckDir)
```

```
[[1]]
NULL

[[2]]
NULL

[[3]]
NULL

[[4]]
NULL
```

```
file.copy(path, pathData)
```

```
[1] FALSE
```

```
setwd(pathDir)
dir()
```

```
[1] "Data"    "Figures" "Results"
```

```
list.files(recursive = T)
```

```
[1] "Data/CPS1985.txt"
```

2.1.3 Import a .txt file

`read.table()` is the most basic import function in R. You can specify tons of different arguments in this function (See `?read.table`).

We import the dataset `CPS1985.txt` in a variable `CPS`.

```
CPS <- read.table(path)
```

After importing a dataset, you should always inspect the data to confirm that the data was imported correctly. `head` prints the first records of the dataset.

```
head(CPS)
```

```
   wage education experience age ethnicity region gender occupation
1  5.10         8         21  35  hispanic  other female     worker
2  6.67        12          1  19      cauc  other   male     worker
3  4.00        12          4  22      cauc  other   male     worker
4  7.50        12         17  35      cauc  other   male     worker
5 13.07        13          9  28      cauc  other   male     worker
6  4.45        10         27  43      cauc  south   male     worker
         sector union married
1 manufacturing    no     yes
2 manufacturing    no      no
3         other    no      no
4         other    no     yes
5         other   yes      no
6         other    no      no
```

Some more data characteristics:

```
# nrow returns the number of rows in the dataset
nrow(CPS)
```

```
[1] 533
```

```
# colnames returns the variable names in the dataset
colnames(CPS)
```

```
 [1] "wage"       "education"  "experience" "age"        "ethnicity"
 [6] "region"     "gender"     "occupation" "sector"     "union"
[11] "married"
```

2.2 Data exploration

We continue to explore the variables in the `CPS` dataset with a focus on `wage` and `sector`.

2.2.1 variable types

The same data can often be stored in different formats. The function `class` returns the type of an R object.

Use the function `class` to

1. Determine the class of the dataset CPS;
2. Determine the class of the variables `wage` and `sector` in the dataset CPS.

```
# @1.
class(CPS)
```

```
[1] "data.frame"
```

```
# @2.
class(CPS$wage)
```

```
[1] "numeric"
```

```
class(CPS$sector)
```

```
[1] "character"
```

Within the `tidyverse` the class of all variables in a dataset can be determined using `map_chr(<dataset>, class)`.

The function `map(<dataset>, <function>)` applies `<function>` to each of the columns of `<dataset>`. We can use `map_chr` when the output of `<function>` is of type `character`.

```
require(tidyverse)
map_chr(CPS, class)
```

```
      wage  education experience        age  ethnicity     region
 "numeric"  "integer"  "integer"  "integer" "character" "character"
    gender occupation     sector      union    married
"character" "character" "character" "character" "character"
```

2.2.2 Factor variables

`factor` is the R-datatype for unordened categorical variables.

Some usefull functions for `factor`variables are:

- `levels`: shows the possible outcomes.
- `table`: shows the frequency of the possible outcomes.

```
levels(CPS$sector)
```

```
NULL
```

```
table(CPS$sector)
```

```
 construction manufacturing         other
           24            98           411
```

Factors are type-safe, that is you will get a warning when you try to insert a value that is not in `levels(<factor variable>)`.

```
test <- CPS$sector
test[1] <- 'insurance'
```

```
head(test)
```

```
[1] "insurance"     "manufacturing" "other"         "other"
[5] "other"         "other"
```

```
table(test)
```

```
test
 construction     insurance manufacturing         other
           24             1            97           411
```

`NA` values are an indication that some calculation went wrong. Notice that the `NA` value is not shown in the output of `table`. To obtain the number of `NA` observation use `sum(is.na(<dataset>))`.

```
sum(is.na(test))
```

```
[1] 0
```

To insert a new value in a `factor` variable, we first adapt the set of allowed outcomes.

```
levels(test) <- c(levels(test), 'insurance')
table(test)
```

```
test
 construction     insurance manufacturing         other
           24             1            97           411
```

```
test[1] <- 'insurance'
table(test)
```

```
test
 construction     insurance manufacturing         other
           24             1            97           411
```

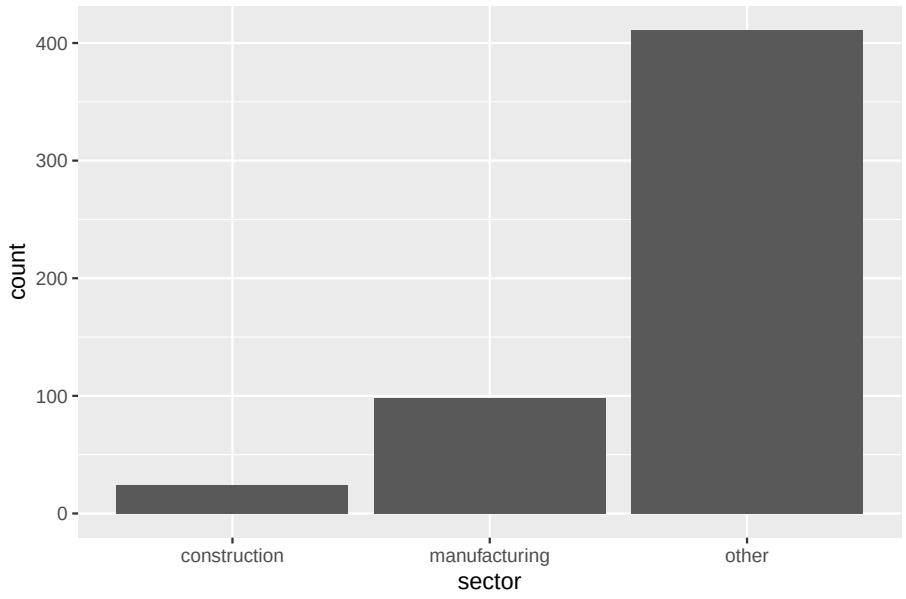
The following code applies `table` to all columns of the dataset `CPS` that are of class factor.

```
require(tidyverse)
CPS %>%
  keep(is.factor) %>% # Select all columns of class factor
  map(table) # apply the table function to these columns
```

```
named list()
```

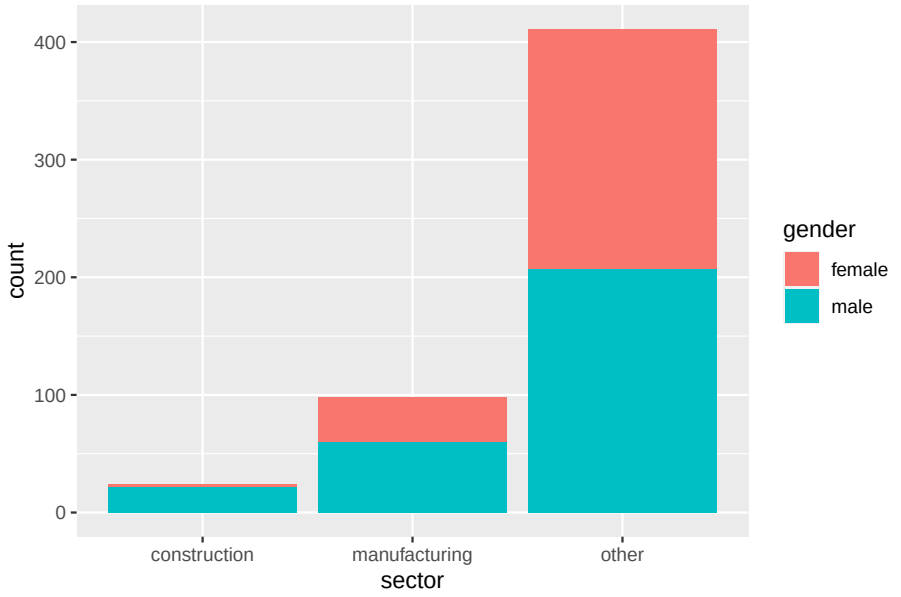
In `ggplot`, `geom_bar` is used to construct barplots. Check out the online documentation for examples.

```
require(ggplot2)
ggplot(CPS) +
  geom_bar(aes(sector))
```



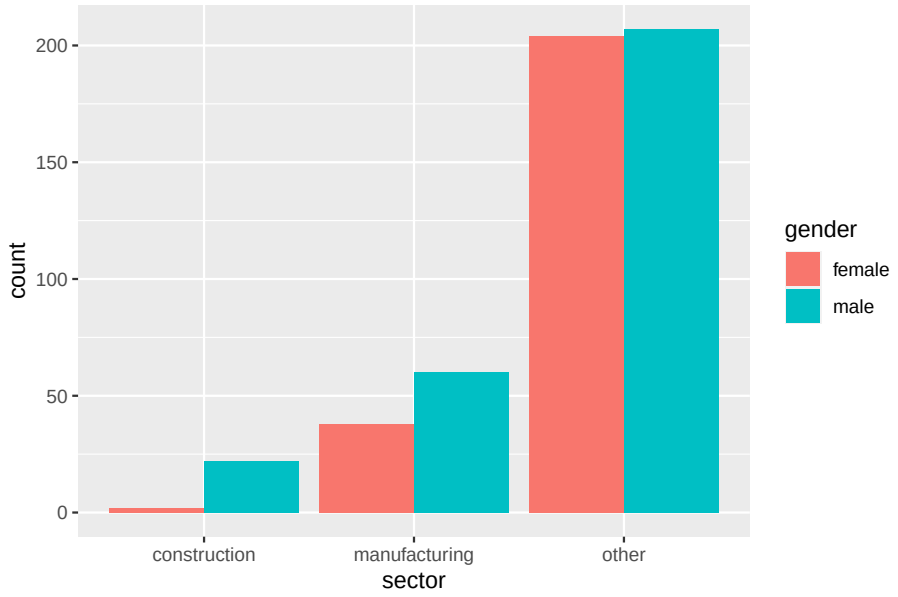
Adding `fill = <second factor variable>` visualizes the interaction between two factor variables.

```
ggplot(CPS) +
  geom_bar(aes(sector, fill = gender))
```



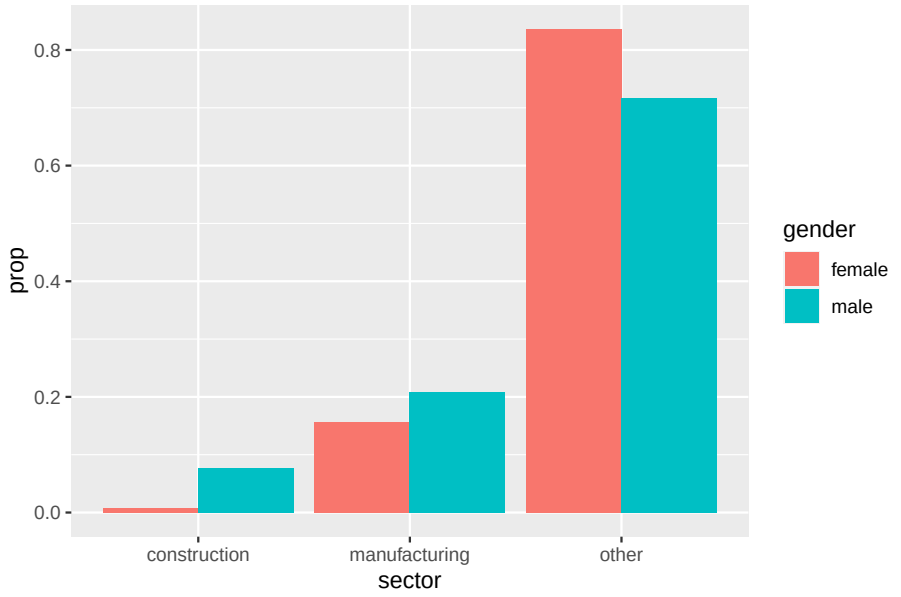
The default behavior is to stack bars. Adding `position = position_dodge()` shows bars of different color side-by-side.

```
ggplot(CPS) +
  geom_bar(aes(sector, fill = gender),
           position = position_dodge())
```



The following code visualizes the distribution of sector by gender.

```
ggplot(CPS) +
  geom_bar(aes(sector,
               fill = gender,
               y = ..prop..,
               group = gender),
           position = position_dodge())
```



2.2.3 Numeric variables

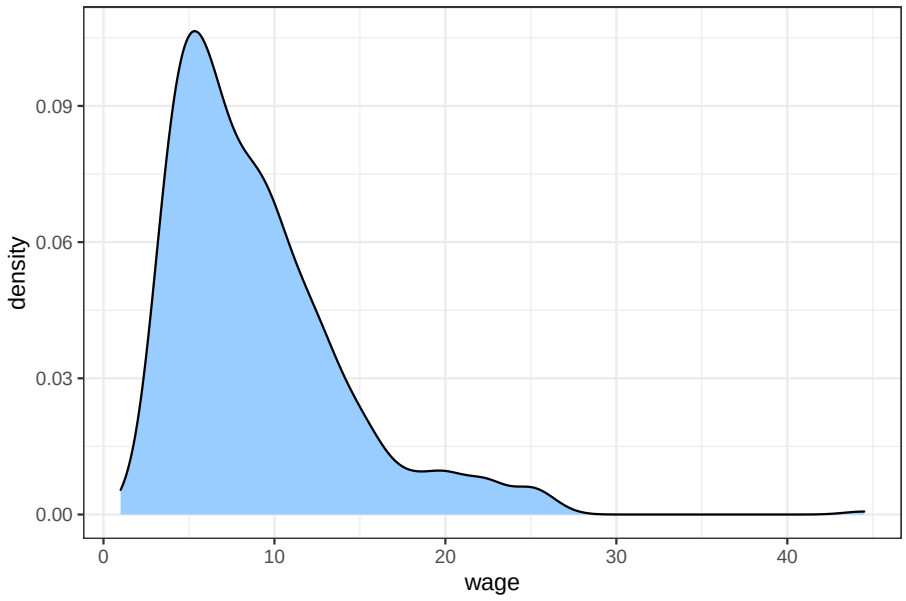
The variable `wage` in the dataset `CPS` is numeric.

```
class(CPS$wage)
```

```
[1] "numeric"
```

A good strategy is to start by visualizing numeric variables in a density plot.

```
ggplot(CPS) +
  theme_bw() +
  geom_density(aes(wage), fill = "#99CCFF")
```

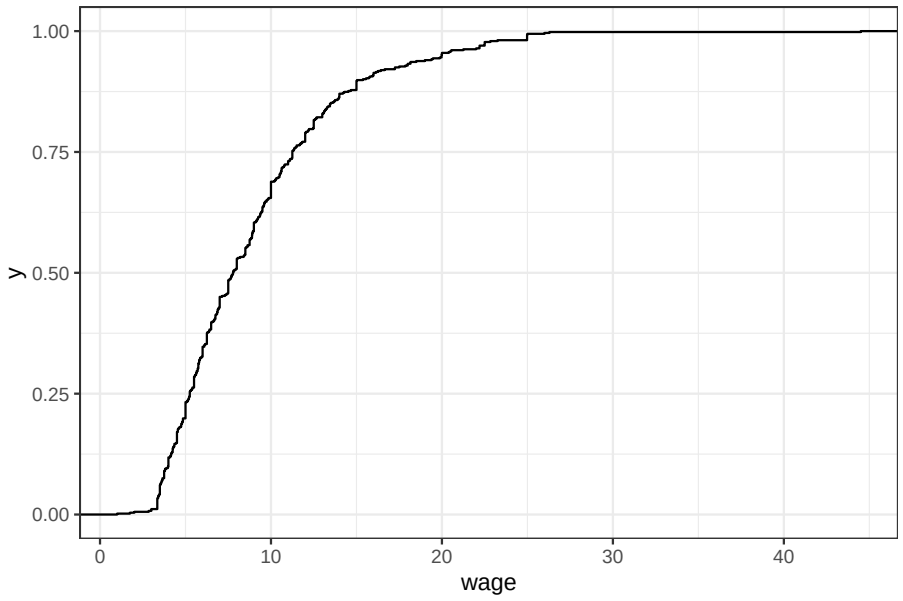


Adding `theme_bw()` selects the black and white theme for ggplot figures.

The density shows that most wages are between 0 and 30, but there is an outlier with a wage of more than 40.

`stat_cdf` visualizes the empirical cdf.

```
ggplot(CPS) +
  theme_bw() +
  stat_ecdf(aes(wage))
```



`mean` and `sd` calculate the empirical mean and standard deviation of a numeric variable.

```
mu <- mean(CPS$wage)
sigma <- sd(CPS$wage)

print(c(mu, sigma))
```

```
[1] 9.032 5.141
```

You can also calulate the mean and standard deviation using the formulas

$$\mu = E(X) = \frac{1}{n}\sum_{i=1}^{n} x$$

and

$$\sigma = \sqrt{E((X-\mu)^2} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x-\mu)^2}$$

```
n <- length(CPS$wage)

# mean
sum(CPS$wage)/n
```

```
[1] 9.032
```

```
# sd
sqrt(sum((CPS$wage - mu)^2)/n)
```

```
[1] 5.136
```

There is a slight difference between our outcome and the result of `sd(.)`, since the latter divides by `n-1` instead of `n`.

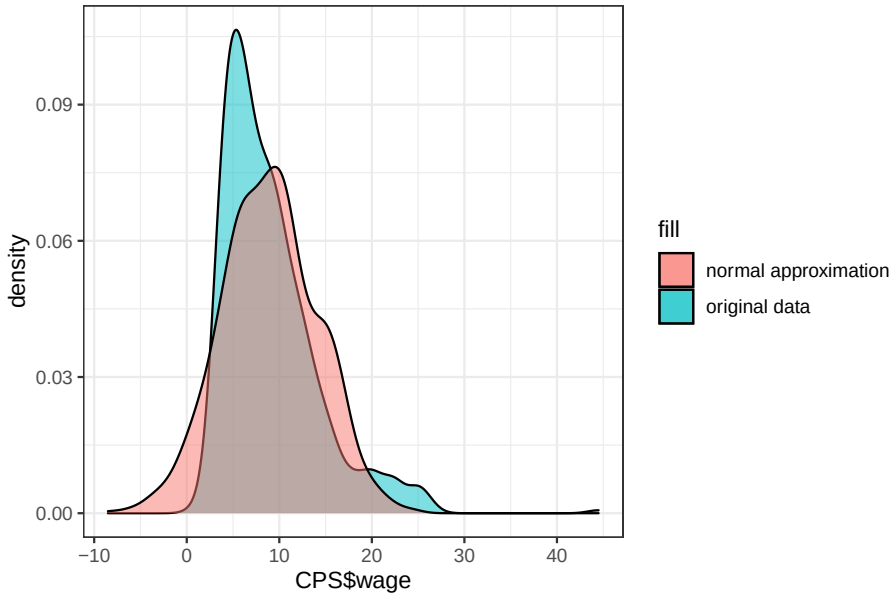
We test whether the normal distribution is a good approximation for `CPS$wage`. For this we visualize the density of `CPS$wage` and its normal approximation in a single graph.

In order to visualize the normal approximation we simulate 1000 observations from a normal distribution with the same mean and standard deviation as `CPS$wage`.

```
normal_approx <- rnorm(1000, mean = mu, sd = sigma)
```

`rnorm` generates observations from a normal distribution.

```
ggplot() +
  theme_bw() +
  geom_density(aes(CPS$wage, fill = 'original data'), alpha = .5) +
  geom_density(aes(normal_approx, fill = 'normal approximation'), alpha = .5)
```



The shape of these densities is quite different and we conclude that the normal distribution is not a good approximation for `CPS$wage`. One of the reasons for

this poor approximation is that the normal distribution is defined on $[-\infty, \infty]$, whereas wage has to be positive.

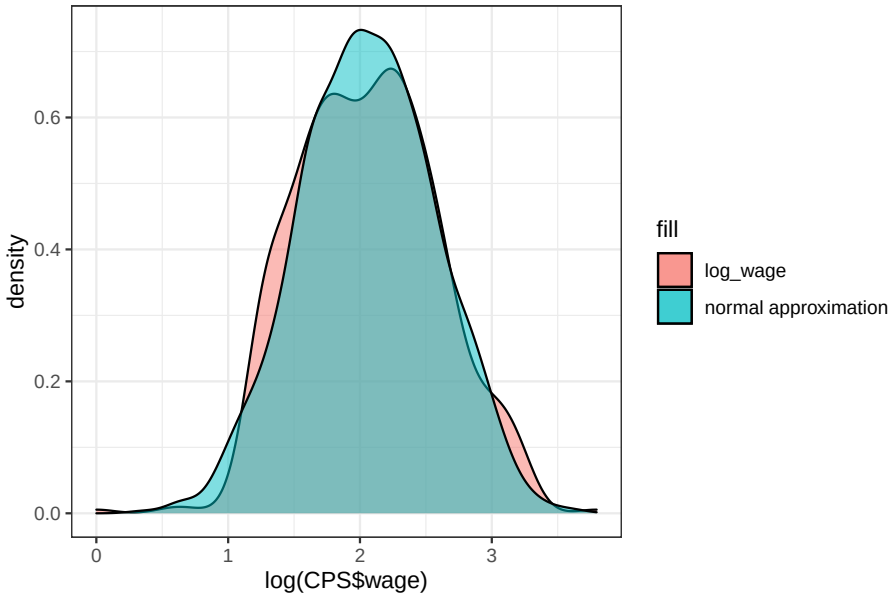
We transform the outcomes of `CPS$wage` by taking the logarithm.

```
log_wage = log(CPS$wage)
```

Exercise:

Check whether the normal distribution is a good approximation for `log(CPS$wage)`.

```
mu = mean(log_wage);
sigma = sd(log_wage);

normal_approx <- rnorm(1000, mean = mu, sd = sigma)

ggplot() +
  theme_bw() +
  geom_density(aes(log(CPS$wage), fill = 'log_wage'), alpha = .5) +
  geom_density(aes(normal_approx, fill = 'normal approximation'), alpha = .5)
```



The approximation is better, but the normal distribution has fatter tails than the log(wage) distribution.

Chapter 3

Simple, parametric distributions for frequency and severity data

3.1 The exponential distribution

In this tutorial you will simulate data from an exponential distribution with density

$$f(x) = \lambda \cdot e^{-\lambda \cdot x}.$$

You will then explore and visualize these data. Finally, you will fit an exponential distribution to the data using Maximum Likelihood Estimation (MLE) (as discussed in Chapter 13 of the Loss Models book).

3.1.1 Simulating data

Use the R function `rexp` to simulate `10 000` observations from an exponential distribution with mean 5.

1. Create a variable `nsim` for the number of simulations;
2. Create a variable `lambda` for the λ value of the exponential distribution. Hint: the mean of the exponential distribution is given by $\frac{1}{\lambda}$ when using the parametrization given above;
3. Check the documentation of `rexp` to see which parametrization `R` uses for the exponential distribution;
4. Simulate `nsim` observations from the exponential distribution. Store the result in the variable `sim`;

5. Calculate `mean(sim)` and verify that the simulated mean is close to 5.

```
# @1.
nsim <- 10000;

# @2.
lambda <- 1/5;

# @3.
?rexp

# @4.
sim <- rexp(nsim, rate = lambda);

# @5.
mean(sim)
```

```
[1] 5.025
```

3.1.2 Exploratory analysis

1. Calculate the (empirical) variance of the simulated sample;
2. Calculate the (emprical) skewness of the simulated sample. The skewness is defined as
$$\frac{E((X-\mu)^3)}{\sigma^3};$$
3. Calculate (empirically) $VaR_{0.95}$ and $TVaR_{0.95}$ for the simulated sample.

```
# @1.
variance <- var(sim)
variance
```

```
[1] 25.63
```

```
# @2.
mu <- mean(sim)
sigma <- sqrt(var(sim))
numerator <- mean((sim - mu)^3)

skewness <- numerator / sigma^3
skewness
```

```
[1] 2.189
```

```
# @3.
var95 <- quantile(sim, 0.95);
var95
```

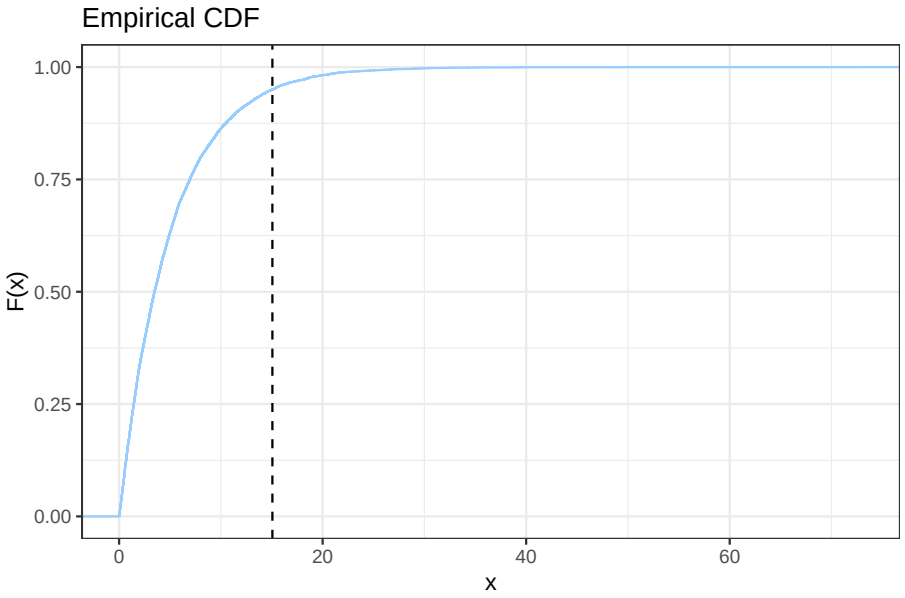
```
  95%
15.06
```

```
tvar95 <- mean(sim[sim > var95])
tvar95
```

```
[1] 20.12
```

3.1.3 Data visualization with ggplot

1. Load the package `ggplot2`;
2. You will construct step-by-step the following graph of the empirical CDF



Let $x_{(i)}$ be the i-th simulated value when sorted ascending. The empirical CDF is given by

$$\hat{F}(x_{(i)}) = \frac{\#\{\text{observations } \leq x_{(i)}\}}{n} = \frac{i}{n}.$$

2.1. Create a new ggplot. Add `stat_ecdf` using the simulated data;

```
ggplot() +
  stat_ecdf(aes(???))
```

2.2. Change the color of the line to blue by adding the option `col = #99CCFF` to `stat_ecdf`;

2.3. Add the black and white ggplot theme, `theme_bw()`;

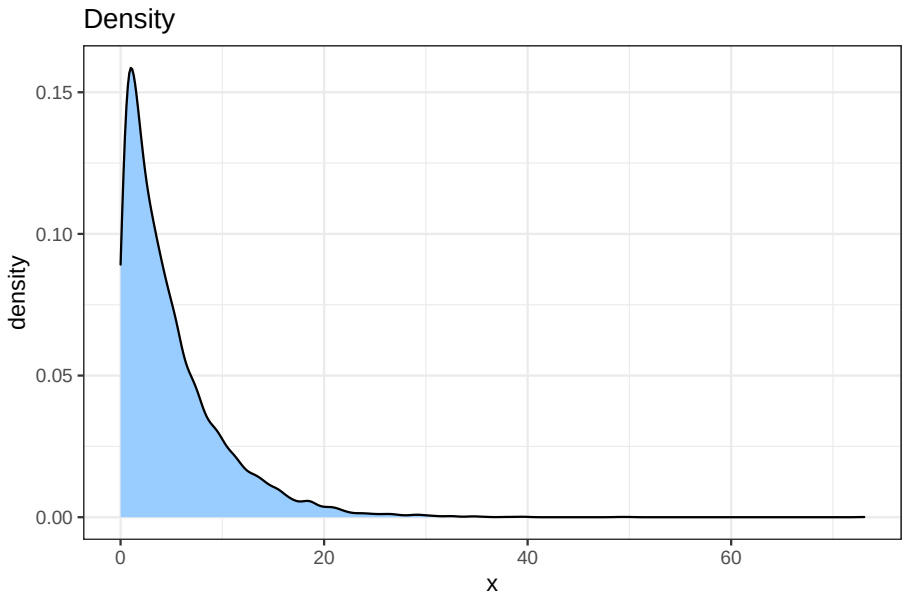
2.4. Add x and y labels to the graph. Hint: use `xlab`, `ylab`;

2.5. Add a vertical line to indicate the $VaR_{0.95}$. Check the documentation for `geom_vline`;

2.6. Add a title to the plot using `ggtitle`;

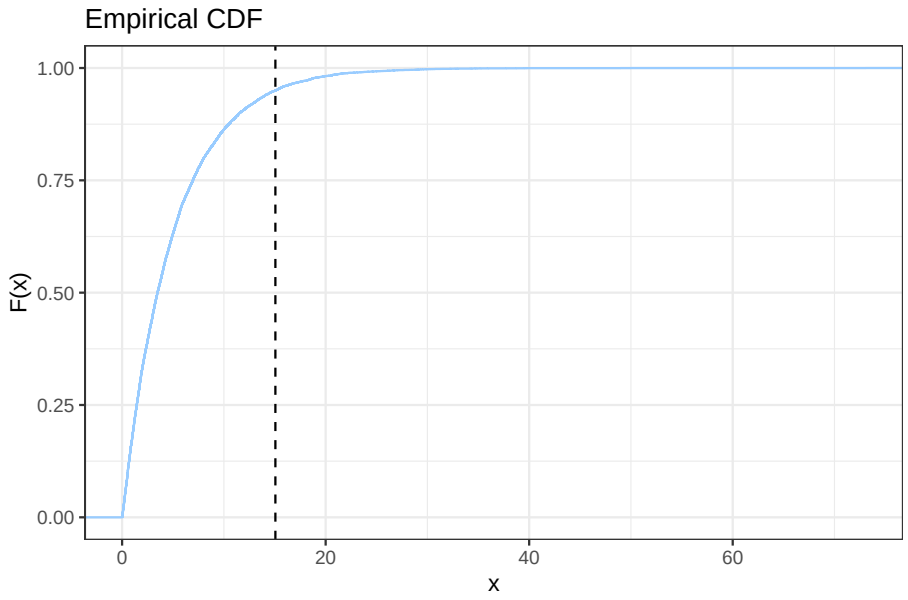
2.7. Change the number of simulations `nsim` to 50 and observe the effect on the empirical CDF.

3. Use `geom_density` to create a density plot of the data. Improve the look of the graph using what you learned when creating the plot of the empirical CDF.

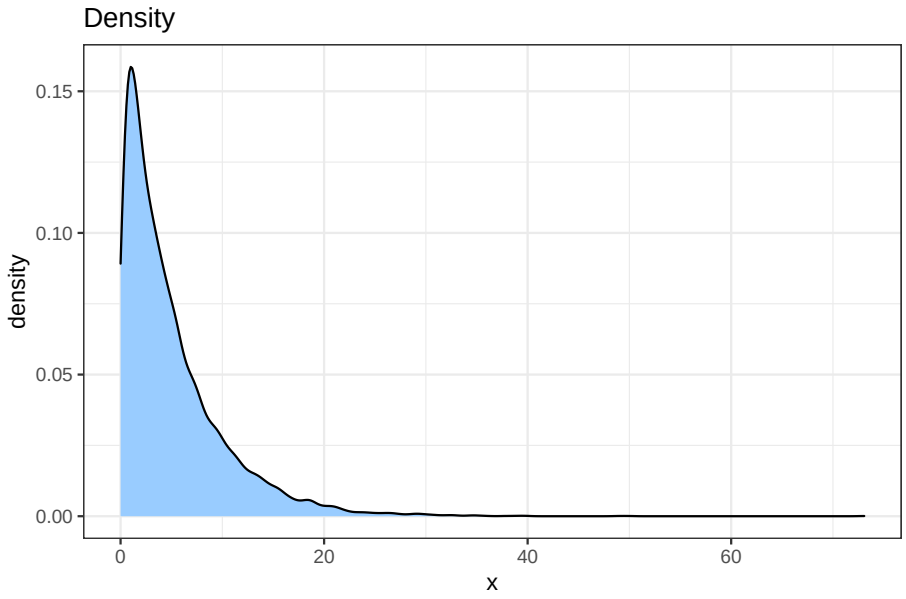



```r
# @1
library(ggplot2)

# @2.1

p <- ggplot() +
  stat_ecdf(aes(sim))

# @2.2
p <- ggplot() +
  stat_ecdf(aes(sim), col = "#99CCFF")

# @2.3
p <- p + theme_bw()
```

```
# @2.4
p <- p +
  xlab('x') +
  ylab('F(x)')

# @2.5
p <- p +
  geom_vline(xintercept = var95, linetype = 'dashed')

# @2.6
p <- p +
  ggtitle('Empirical CDF')

print(p)
```




```
# @3
ggplot() +
  geom_density(aes(sim), fill = "#99CCFF") +
  theme_bw() +
  ggtitle('Density') +
  xlab('x')
```



3.1.4 Maximum Likelihood Estimation (MLE)

The density of the exponential distribution is given by

$$f(x) = \lambda \cdot e^{-\lambda \cdot x}.$$

You have `nsim` simulated observations $x_i, \ldots, x_{nsim}$ from this distribution. In this exercise you look for the MLE of the parameter λ using the simulated data.

1. Write down (on paper) the formula for the likelihood of the observed data as a function of λ;

The likelihood for the observed data is

$$L(\lambda) = \prod_{i=1}^{\text{nsim}} \lambda \cdot e^{-\lambda \cdot x_i}.$$

2. Derive (on paper) the loglikelihood;

The loglikelihood for the observed data is

$$l(\lambda) = \sum_{i=1}^{\text{nsim}} \log(\lambda) - \lambda \cdot x_i.$$

3. Compute (on paper) the MLE for λ. Hint: put $\frac{d\ell}{d\lambda}(\hat{\lambda}) = 0$ and solve for the unknown λ;

$$\frac{dl}{d\lambda} = \frac{n}{\lambda} - \sum_{i=1}^{\text{nsim}} x_i$$

This derivative is zero when

$$\hat{\lambda} = \frac{\text{nsim}}{\sum_{i=1}^{\text{nsim}} x_i}.$$

This is the MLE for λ.

4. You will now find the MLE numerically in R by optimizing the likelihood using the `nlm` procedure;

4.1. Define an R-function `loglikelihood` which takes as input λ and returns the loglikelihood for the simulated sample;

```
loglikelihood <- function(lambda)
{
  loglikelihood <- ???
  return(loglikelihood)
}
```

4.2. The `nlm` procedure minimizes a function. You will minimize the negative loglikelihood $-l(\lambda)$ to find the maximum likelihood estimator $\hat{\lambda}$. Start from the result of 4.1 and create a function `negLoglikelihood` which returns the negative loglikelihood;

4.3. The `nlm` procedure searches for the optimal parameter in the domain $(-\infty, \infty)$. You will use a transformation $\lambda = \exp(\beta)$ and optimize the likelihood for this parameter $\beta \in (-\infty, \infty)$. Update the function `negloglikelihood` to take β as its input;

4.4. Minimize the function `negLoglikelihood` you defined in 4.3. using the `nlm` procedure. Add the option `hessian = TRUE`;

4.5. Interpret the output of `nlm`. What is the maximum likelihood estimate for β and what about λ (see the discussion in Section 13.3 on Variable and interval estimation). Do you find the same result as in 3.?

4.6. You will now construct a 95% confidence interval for the unknown parameter β and afterwards for λ. Under MLE the actual parameter β is asymptotically distributed as (see Chapter 13 on Variance and interval estimation)

$$\beta \sim \mathcal{N}(\hat{\beta}, \mathcal{I}^{-1}(\hat{\beta})),$$

where $\mathcal{I}$ denotes the Fisher information matrix. You calculate this matrix as the negative of the Hessian, the matrix with the second order derivatives of the log-likelihood, evaluated in $\hat{\beta}$. Of course, since the Exponential distribution only has one paremeter, the matrix reduces to a scalar.

4.6.1. You added the option `hessian = TRUE` in `nlm` to obtain the Hessian (numerically) in the `nlm` procedure. Use the Hessian to calculate the standard error of the MLE $\hat{\beta}$. Because you calculated the Hessian of the negative log-likelihood, it suffices to take its inverse to obtain the (asymptotic) variance of the MLE.

4.6.2. A 95% confidence interval for the actual parameter β is then given by

$$[\hat{\beta} - \Phi^{-1}(0.975) \cdot \mathrm{se}_{\hat{\beta}}, \hat{\beta} + \Phi^{-1}(0.975) \cdot \mathrm{se}_{\hat{\beta}}],$$

where Φ is the CDF of the standard normal distributon. Calculate the 95% confidence interval for the intensity β based on the simulated sample. Is the orignal $\beta = \log \lambda$ (used for simulating the data) contained in this interval?

4.6.3 You will now use the delta method (see Section 13.3 in the book) to construct a confidence interval for the unknown λ. The MLE for λ is obtained from the transformation $\hat{\lambda} = \exp \hat{\beta}$. The corresponding se is calculated as $se_{\hat{\lambda}} = (\exp \hat{\beta})^2 \cdot \mathrm{se}_{\hat{\beta}}$. Using these ingredients you are now ready to construct the confidence interval for the unknown parameter λ.

```
# @1
loglikelihood <- function(lambda)
{
  loglikelihood <- nsim * log(lambda) - sum(lambda * sim)
  return(loglikelihood)
}

# @2
negLoglikelihood <- function(lambda)
{
  loglikelihood <- nsim * log(lambda) - sum(lambda * sim)
  return(-loglikelihood)
}

# @3
negLoglikelihood <- function(beta)
{
  lambda <- exp(beta)
  loglikelihood <- nsim * log(lambda) - sum(lambda * sim)

  return(-loglikelihood)
}

# @4
fit <- nlm(negLoglikelihood, p = 0, hessian = TRUE)
```

```
Warning in nlm(negLoglikelihood, p = 0, hessian = TRUE): NA/Inf replaced by
```

```
maximum positive value
```

```
fit
```

```
$minimum
[1] 26145

$estimate
[1] -1.614

$gradient
[1] -9.013e-05

$hessian
      [,1]
[1,] 10001

$code
[1] 1

$iterations
[1] 8
```

```
# @5
lambdaMLE <- exp(fit$estimate)
lambdaMLE
```

```
[1] 0.199
```

```
nsim / sum(sim)
```

```
[1] 0.199
```

```
# @6
sigma.beta <- sqrt(solve(fit$hessian))
sigma.lambda <- sigma.beta * lambdaMLE^2

c(lambda - qnorm(0.975) * sigma.lambda, lambda + qnorm(0.975) * sigma.lambda)
```

```
[1] 0.1992 0.2008
```

3.2 Discrete distributions

In this computer lab you will simulate discrete data (e.g. claim counts). You will then explore and fit a statistical model to the simulated data set.

3.2.1 Simulating the data

You simulate `10 000` observations from a lognormal distribution (a continuous distribution!). You will then discretize the simulated data by rounding down.

1. Simulate `10 000` observations from a lognormal distribution with density

$$f(x) = \frac{1}{x \cdot \sqrt{2\pi}} \cdot \exp\left(-(\ln(x) + 1.5)^2\right).$$

Hint: Check the specification of the lognormal distribution in R, `?rlnorm`.

2. Discretize the data by rounding down using the `floor` function.

```
# Example of the floor function
x <- runif(6)*3
rbind(x = x, floor = floor(x))
```

```
       [,1] [,2]  [,3]  [,4]   [,5]  [,6]
x     2.794 1.34 2.188 1.705 0.4086 1.212
floor 2.000 1.00 2.000 1.000 0.0000 1.000
```

```
nsim <- 10000;
sim <- exp(rnorm(nsim, mean = -1.5, sd = 1))

# we only observe the data discrete
sim <- floor(sim)
```

3.2.2 Exploratory analysis

You just obtained simulated discrete data. You now want to investigate which discrete distributions could be good candidates for modelling the simulated data.

1. Start by calculating the mean and variance of the simulated data. Is the data underdispersed or overdispersed?

The variance is larger than the mean of the data. The data is overdispersed.

2. Which of the following three distributions will most likely describe the data in a good way?

- Binomial
- Poisson
- Negative binomial

The Negative binomial distribution is the best candidate, since this distribution is overdispersed.

3. Test visually whether the data belongs to the $(a, b, 0)$ class, i.e. see whether the relation

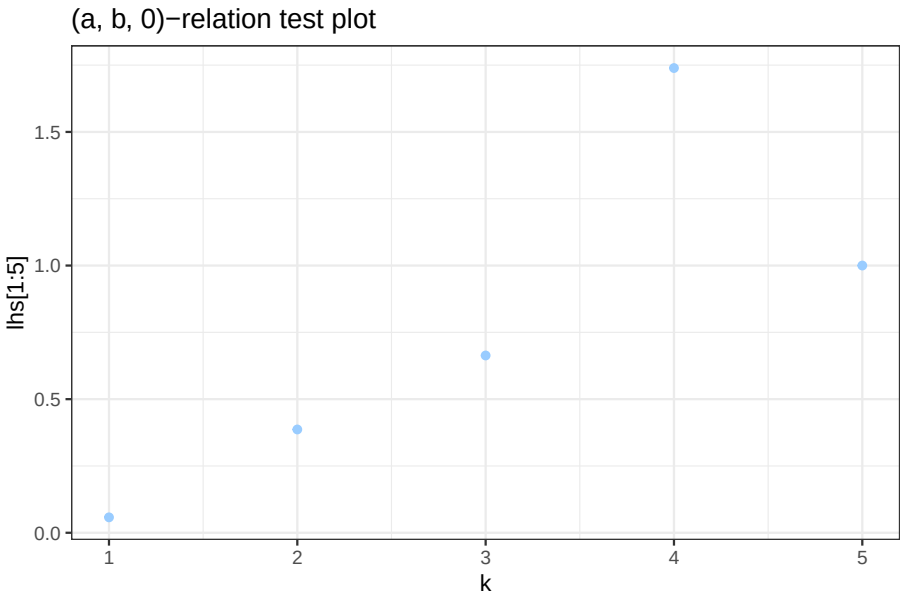
$$k \cdot \frac{p_k}{p_{k-1}} = a \cdot k + b, \quad k = 1, 2, \ldots$$

holds for the simulated data.

3.1 Compute the left hand side (lhs) of this relation.

- Use `prop.table(table(???))` to get the empirical probability distribution p_k;
- The data is heavy tailed and the lhs can become very large when p_{k-1} is small. You check the relation for $k = 1, \ldots, 5$;
- Create a vector k, p_k and p_{k-1} for $k = 1, \ldots, 5$;
- Combine these results to obtain the lhs of the equation.

3.2 Use ggplot to construct a graph containg the points $(k, k \cdot \frac{p_k}{p_{k-1}})$. Your graph should look similar to



- Load the package `ggplot2`;
- Create a new ggplot figure. Add a `geom_point` graphic using the data points (k, lhs);

```
ggplot() +
  geom_point(aes(???, ???))
```

- Change the color of the points to blue by adding the option `col = #99CCFF` to `geom_point`;
- You can further customize the graph with `theme_bw()`, `xlab`, `ylab`, `ggtitle`, ...

3.3. Discuss. Is a distribution from the $(a, b, 0)$ class a good candidate for this data?

```
mean(sim)
```

```
[1] 0.0897
```

```
var(sim)
```

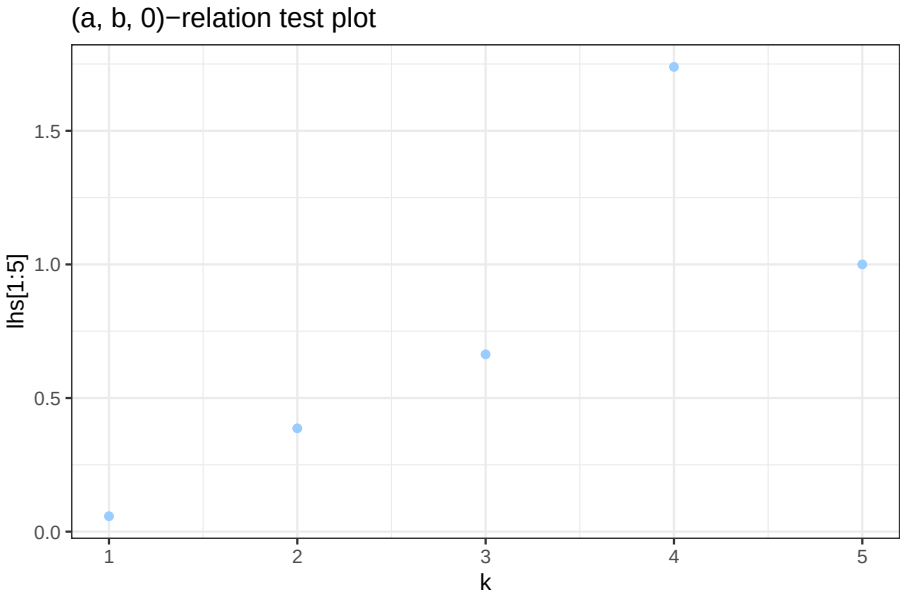
```
[1] 0.1555
```

```
prob <- prop.table(table(sim))
prob
```

```
sim
     0      1      2      3      4      5      6      8     10
0.9319 0.0538 0.0104 0.0023 0.0010 0.0002 0.0001 0.0002 0.0001
```

```
k <- 1:5
pk <- prob[k+1]
pkmin1 <- prob[k]

lhs <- k * pk / pkmin1

ggplot() +
  geom_point(aes(k[1:5], lhs[1:5]), color = "#99CCFF") +
  theme_bw() +
  xlab("k") +
  ggtitle('(a, b, 0)-relation test plot')
```

```
Don't know how to automatically pick scale for object of type table. Defaulting to con
```



3.2.3 Maximum Likelihood Estimation (MLE)

You will now fit two count distributions to the simulated data:

- Geometric
- Negative binomial (NB)

3.2.3.1 Geometric

For a Geometric distribution with parameter $p \in [0, 1]$ the probability of observing k events is given by

$$P(N = k) = (1 - p)^k \cdot p.$$

In the Appendix 'An inventory of discrete distributions' of the Loss Models book you will find a different parameterization. That is

$$P(N = k) = \left(\frac{\theta}{1 + \theta}\right)^k \cdot \frac{1}{(1 + \theta)}.$$

If you put $p = \frac{1}{1+\theta}$ you can work from the second to the first parametrization. Verify this.

1. Derive an expression for the loglikelihood;

The likelihood is given by

$$L(p) = \prod_{i=1}^{\text{nsim}} P(N = x_i) = \prod_{i=1}^{\text{nsim}} (1 - p)^{x_i} \cdot p.$$

The loglikelihood is

$$l(p) = \sum_{i=1}^{\text{nsim}} \left(\log(1-p) \cdot x_i + \log(p)\right).$$

2. Implement the negative loglikelihood as a function in R;

```
geom.negLoglikelihood <- function(p)
{
  loglikelihood <- ???

  return(-loglikelihood)
}
```

3. The probability p can only take values in $[0, 1]$. Change the function `geom.negLoglikelihood` to take a parameter $\beta \in (-\infty, \infty)$. Then transform the interval $(-\infty, \infty)$ to $[0, 1]$ using the logit transform

$$\text{logit}(p) = \log\left(\frac{p}{1-p}\right) = \beta.$$

Inverting this expression, you find (verify this!)

$$p = \frac{\exp(\beta)}{1+\exp(\beta)}.$$

4. Maximize the likelihood using the `nlm` procedure in R and interpret the results.

```
geom.negLoglikelihood <- function(beta)
{
  p <- exp(beta) / (1+exp(beta))

  loglikelihood <- sum(log(1-p) * sim) + nsim * log(p)

  return(-loglikelihood)
}

fit <- nlm(geom.negLoglikelihood, 1)
```

```
Warning in nlm(geom.negLoglikelihood, 1): NA/Inf replaced by maximum positive
value

Warning in nlm(geom.negLoglikelihood, 1): NA/Inf replaced by maximum positive
value
```

```
fit
```

```
$minimum
[1] 3099

$estimate
[1] 2.411

$gradient
[1] -0.000222

$code
[1] 1

$iterations
[1] 6
```

```
geom.p <- exp(fit$estimate) / (1+exp(fit$estimate))
geom.p
```

```
[1] 0.9177
```

```
geom.loglik <- -fit$minimum
```

3.2.3.2 Negative binomial

You will now go from the one parameter geometric distribution to a two parameter discrete distribution, the Negative Binomial. Its pf is specified as follows:

$$Pr(N = k) = \frac{\Gamma(a+k)}{\Gamma(a)k!}\left(\frac{\mu}{\mu+a}\right)^k\left(\frac{a}{\mu+a}\right)^a.$$

1. Follow the same steps as with the geometric distribution to fit the NB distribution to the simulated data.

- The parameters μ and a can take values on the positive real line $[0, \infty)$. Choose an appropriate transormation to convert this interval to the whole real line, $(-\infty, \infty)$.

```
NB.negativeLoglikelihood <- function(beta)
{
  mu <- exp(beta[1])
  a <- exp(beta[2])

  loglikelihood <- sum(lgamma(a + sim) - lgamma(a) - lfactorial(sim) + sim * log(mu/(mu + a)) + a

  return(-loglikelihood)
}
```

```
fit <- nlm(NB.negativeLoglikelihood, c(0, 0), hessian=TRUE)
```

```
Warning in nlm(NB.negativeLoglikelihood, c(0, 0), hessian = TRUE): NA/Inf
replaced by maximum positive value

Warning in nlm(NB.negativeLoglikelihood, c(0, 0), hessian = TRUE): NA/Inf
replaced by maximum positive value

Warning in nlm(NB.negativeLoglikelihood, c(0, 0), hessian = TRUE): NA/Inf
replaced by maximum positive value
```

```
fit
```

```
$minimum
[1] 2975

$estimate
[1] -2.411 -1.893

$gradient
[1] 8.958e-04 2.859e-05

$hessian
          [,1]      [,2]
[1,] 562.22093   0.01064
[2,]   0.01064 121.47852

$code
[1] 1

$iterations
[1] 19
```

```
# Store the fitted values
nb.mu <- exp(fit$estimate[1])
nb.a <- exp(fit$estimate[2])

c(mu = nb.mu, a = nb.a)
```

```
    mu      a
0.0897 0.1506
```

```
nb.loglik <- -fit$minimum
```

3.2.4 Comparing fitted models

You will now compare which model best fits the data using AIC as well as some visual inspection tools.

3.2.4.1 AIC

Suppose that you have a statistical model calibrated on some data. Let k be the number of estimated parameters in the model. Let $\hat{L}$ be the maximum value of the likelihood function for the model. Then the AIC of the investigated model is the following

$$\mathrm{AIC} = 2k - 2\ln(\hat{L}).$$

Given a set of candidate models for the data, the preferred model is the one with the minimum AIC value. Thus, AIC rewards goodness of fit (as assessed by the likelihood function), but it also includes a penalty that is an increasing function of the number of estimated parameters. The penalty discourages overfitting, because increasing the number of parameters in the model almost always improves the goodness of the fit. For more information see wikipedia.

1. Calculate the AIC for both fitted models. Hint: $-\ln(\hat{L})$ is the minimum reached by the `nlm` procedure.

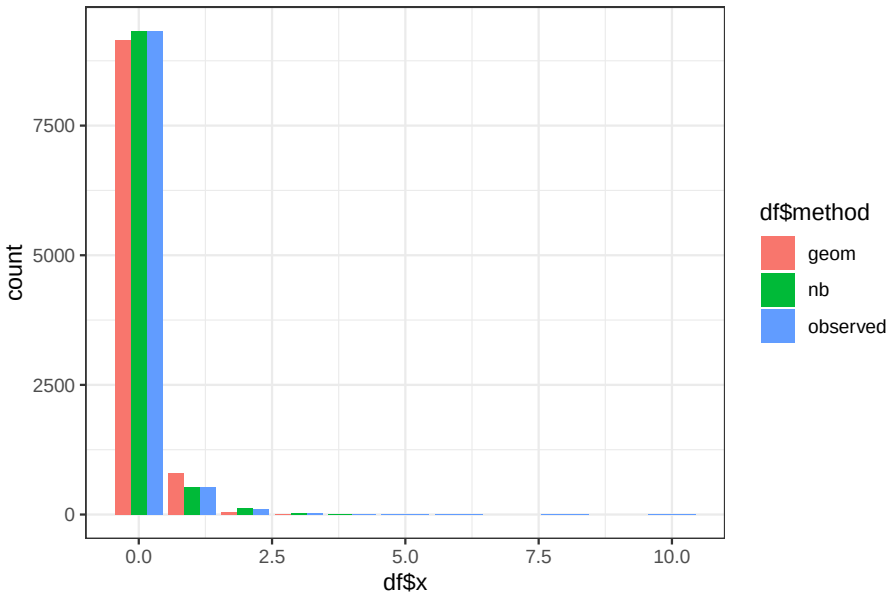
2. Which of the two models does AIC prefer?

```
aic <- c(geom = 2 * 1 - 2 * geom.loglik,
         nb = 2 * 2 - 2 * nb.loglik)

# the nb distribution has the lowest AIC
print(aic)
```

```
geom   nb
6200 5955
```

3.2.4.2 Visual inspection

Using the fitted parameters you will now simulate new datasets of `nsim` observations from the Geometric and Negative Binomial distribution. You compare the shapes of the fitted and the original data.



1. Simulate a dataset from the Geometric distribution using the fitted parameters;
2. Simulate a dataset from the Negative binomial distribution using the fitted parameters;
3. You will now create a barplot using `geom_barplot`. First the data has to be merged into a single data frame containing two columns:

- x: the simulated values;
- method: a string, referring to the method used to simulate the data (i.e. observed, geom or nb).

```
x   method
0 observed
0 observed
0     geom
1     geom
0       nb
```

3.1 Create datasets `df.observed`, `df.geom`, `df.nb` with the simulated data in one column and a string referring to the method used in the other column.

```
df.observed <- data.frame(x = ???, method = 'observed')
```

3.2 Combine these three datasets into a single dataset using `rbind`.

```
df <- rbind(df.observed, df.geom, df.nb);
```

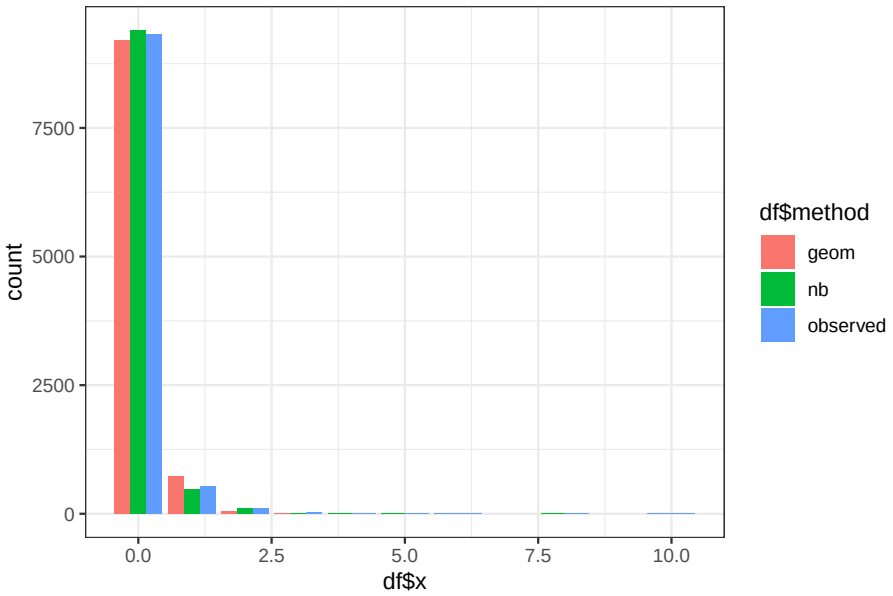
4. Create a barplot using `geom_bar`.

```
ggplot() +
  geom_bar(aes(???, fill = ???)) +
  theme_bw()
```

5. By default `geom_bar` stacks the bars for the different methods. To show the bars sideways add the option `position = position_dodge()`.

6. Discuss. Which distribution best mimicks the original data?

```
sim.geom <-  rgeom(nsim, geom.p)
sim.nb <- rnbinom(nsim, mu = nb.mu, size = nb.a)

df.observed <- data.frame(x = sim, method = 'observed')
df.geom <- data.frame(x = sim.geom, method = 'geom')
df.nb <- data.frame(x = sim.nb, method = 'nb')

df <- rbind(df.observed, df.geom, df.nb);

ggplot() +
  geom_bar(aes(df$x, fill=df$method), position = position_dodge()) +
  theme_bw()
```



Chapter 4

Putting it all together: case study on modelling claim counts

In this tutorial you will import a data set with the number of claims registered on a group of policyholders during one year. You will look for a suitable discrete distribution to model this data set. Hereto you will fit the Poisson, the Negative Binomial, the Zero-Inflated and the Hurdle Poisson to the data, while estimating the parameters used by these distributions with Maximum Likelihood Estimation. As a final step, you will compare the different model fits and select the best fitting parametric distribution.

4.1 Read in data

Importing data is often the first step in any analysis. In this tutorial the data is stored in the file `NonFleetCo507Final.txt`, which is located in a local subdirectory `data`. Check out chapter 4 of Data science in insurance: an R intro for a detailed overview of the R methods for importing data.

4.1.1 Determining the file path

Before we can import the data, we should first determine the exact file path where the data are located. R offers several methods for this task.

1. `file.choose()`

`file.choose()` opens an interactive prompt, which allows you to manually select the location of the file. Once selected R prints the aboslute path to the file in the console.

```
file.choose()
```

```
"C:\Users\u0095171\Dropbox\Verzekeringen niet-leven\Bookdown\data\NonFleetCo507Final.t
```

```
# Store the path in a variable for later use.
path <- "C:\\Users\\u0095171\\Dropbox\\Verzekeringen niet-leven\\Bookdown\\data\\NonFl
```

2. `setwd(<path>)`

`setwd(<path>)` specifies the working directory of the current R session to `<path>`. Once set all files in the working directory can be referenced by relative paths.

```
# This is the map containing all files for the tutorial
setwd("C:\\Users\\u0095171\\Dropbox\\Verzekeringen niet-leven\\Bookdown\\")

# This is a relative path from the working directory to the file we want to import
path <- "data\\NonFleetCo507Final.txt"
```

3. Additional methods for `RStudio`

`RStudio` offers two additional methods for setting the working directory to the location of the current R file.

Method 1: In the menu click Session -> Set Working Directory -> To Source File Location.

Method 2: Run the following code in the R console

```
dir <- dirname(rstudioapi::getActiveDocumentContext()$path)
setwd(dir)

path <- "data\\NonFleetCo507Final.txt"
```

The advantage of these methods is that the working directory is automatically updated when the file is moved.

4.1.2 Importing a .txt file

After obtaining the file path, we read in the `txt` file using `read.table`. We specify the following options:

1. `header = TRUE`: The first row of the file contains the variable names.
2. `sep = \t`: A tab splits the records in the text file.

```
NonFleet <- read.table(file = path, header = TRUE, sep = '\t')
```

The data is now imported in the data.frame `NonFleet`. `head(<data.frame>)` prints the first records of a data.frame. This is a good first check to see whether the data was imported correctly.

```
# Show the first records of the imported data set
head(NonFleet)
```

```
  AgeInsured SexInsured Experience TLength Clm_Count VAge PrivateCar NCD_0
1         32          M         11  0.4654         0   10          1     0
2         26          M          5  0.8077         0   13          1     1
3         32          M          5  0.3997         0    0          1     0
4         32          M          5  0.5832         0    1          1     0
5         41          M         14  0.7748         0    9          1     0
6         28          F          3  0.4928         0    0          1     1
  Cover_C VehCapCubic VehCapTonn
1       1        1797          0
2       0        1590          0
3       1        1997          0
4       1        1997          0
5       0        1597          0
6       1        1587          0
```

Everything looks good. In this tutorial we focus on the variables:

1. `Clm_Count`: Number of claims for the policyholder;
2. `TLength`: Fraction of the year that the policyholder was insured. In insurance this is often called the exposure.

We create separate variables in R to store these covariates

```
Clm_Count <- NonFleet$Clm_Count;
TLength <- NonFleet$TLength;
```

4.2 Exploratory analysis

We now explore the available data and analyze the number of claim counts per insured.

4.2.1 Summary statistics disregarding exposure

We start our analysis by computing the mean and variance of the number of observed claims. If we denote by n_i the number of claims observed for policyholder i, we can compute the mean and variance as

$$\mu = E(X) = \frac{1}{m} \cdot \sum_{i=1}^{m} n_i$$

and

$$\sigma^2 = E((X-\mu)^2) = \frac{1}{m} \cdot \sum_{i=1}^{m} (n_i - \mu)^2.$$

In these formulas m denotes the number of observations.

```
m <- length(Clm_Count)

mu <- sum(Clm_Count) / m
var <- sum((Clm_Count - mu)^2) / m

c(mean = mu,
  variance = var)
```

```
    mean variance
 0.09848  0.10925
```

4.2.2 Summary statistics taking into account exposure

The previous calculation of the mean and variance does not consider the difference in exposure between policyholders. However, it is important to take exposure into account. Let d_i be the exposure for policyholder i, then we calculate the mean as

$$\mu_{\text{exp}} = \sum_{i=1}^{m} \frac{d_i}{\sum_{i=1}^{m} d_i} \frac{n_i}{d_i} = \frac{\sum_{i=1}^{m} n_i}{\sum_{i=1}^{m} d_i}$$

and the variance as

$$\sigma^2_{\text{exp}} = \frac{\sum_{i=1}^{m} (n_i - \mu_{exp} \cdot d_i)^2}{\sum_{i=1}^{m} d_i}.$$

For more intuition behind these estimators, check out the blog of Arthur Charpentier and Section 15.6.6 from Klugman et al..

```
mu <- sum(Clm_Count) / sum(TLength);
var <- sum((Clm_Count-mu*TLength)^2)/sum(TLength)

c(mean = mu,
  variance = var)
```

```
    mean variance
  0.1546   0.1675
```

This is the expected number of accidents for a policyholder who is insurerd throughout the whole year, i.e. $d_i = 1$.

4.2.3 Empirical probability distribution

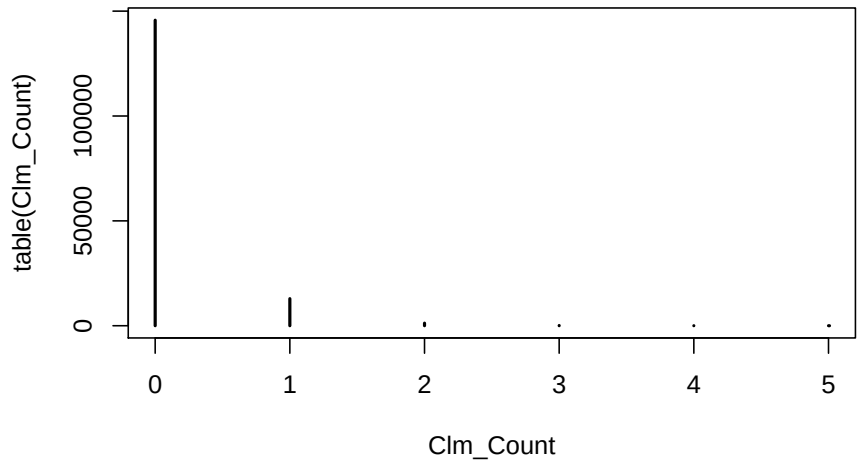
`table` allows us to easily construct a contingency table of the counts.

```
table(Clm_Count)
```

```
Clm_Count
     0      1      2      3      4      5
145683  12910   1234    107     12      1
```

R can plot this table

```
plot(table(Clm_Count))
```



`prop.table` can be used to obtain the empirical probability distribution

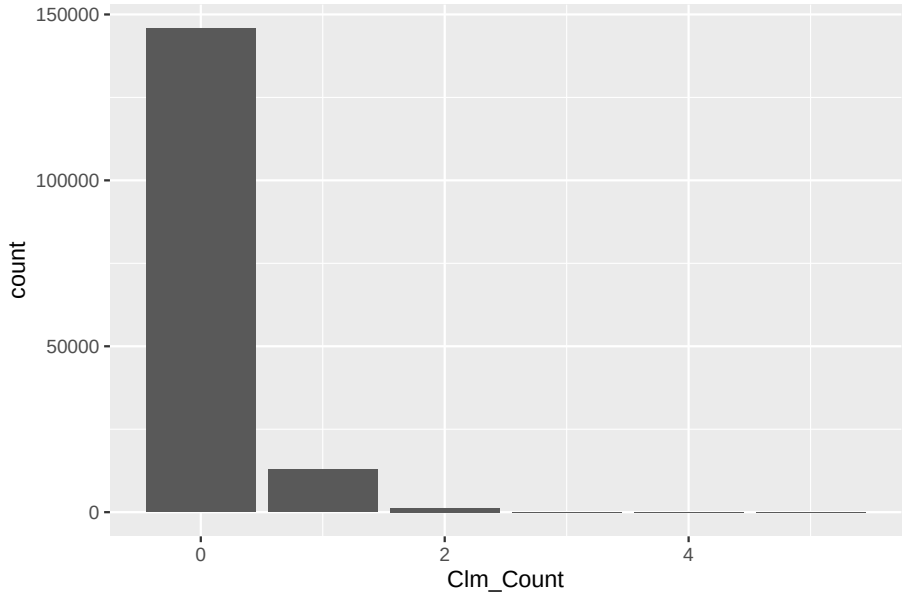
```
prop.table(table(Clm_Count))
```

```
Clm_Count
        0         1         2         3         4         5
9.108e-01 8.071e-02 7.715e-03 6.690e-04 7.502e-05 6.252e-06
```

We can create a better barplot using ggplot

- `ggplot()`: starts the construction of a ggplot figure;
- `geom_bar(...)`: creates a bar plot;
- `aes(<var>)`: specifies the variables used to create the plot.

```
# Run the following line of code when the package ggplot2 is not yet installed.
# install.package(ggplot2)

# Load the package ggplot2
library(ggplot2)

ggplot() +
  geom_bar(aes(Clm_Count))
```

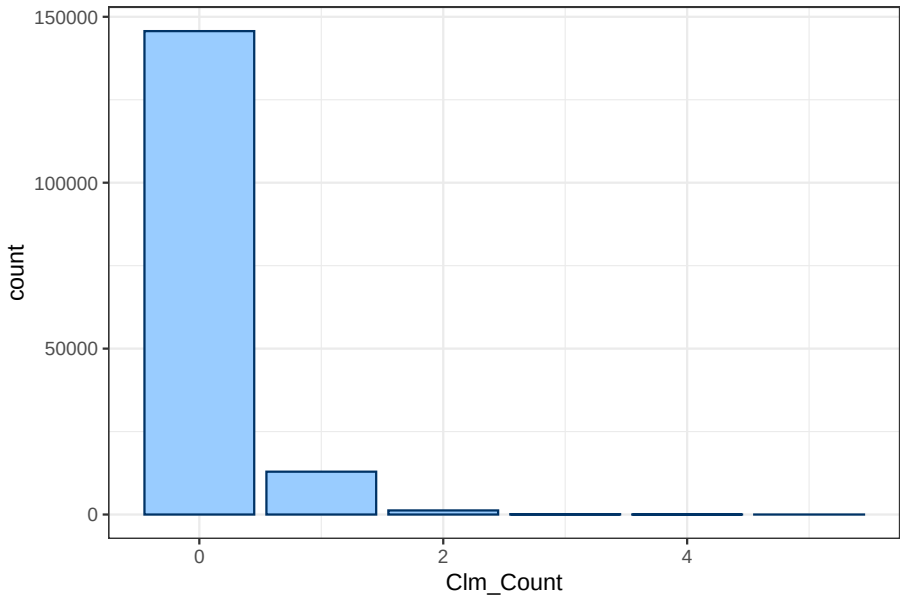


To specify your own theme, you define some visualisation parameters and colors that will be used in your ggplot calls.

```
col <- "#003366"
fill <- "#99CCFF"
```

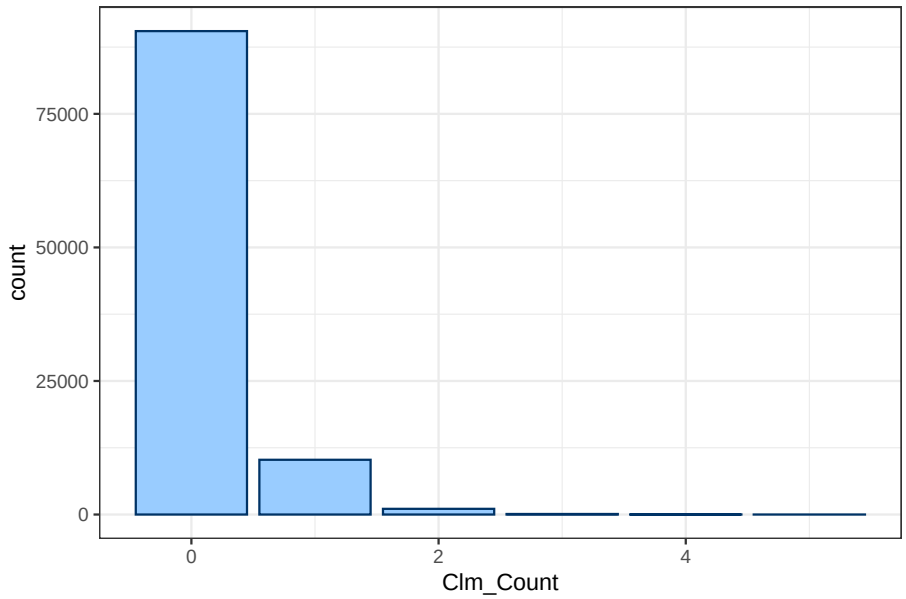
Instead of manually changing all details of the plot, ggplot also offers some general layout schemes. In this tutorial we use the black and white theme `theme_bw()`.

```
ggplot() +
  geom_bar(aes(Clm_Count), col = col, fill = fill) +
  theme_bw()
```



The weight argument in `aes` allows you to weight the number of policyholders who file 0 claims, 1 claim and so on by exposure instead of simply counting the number of policyholders.

```
ggplot() +
  geom_bar(aes(Clm_Count, weight = TLength), col = col, fill = fill) +
  theme_bw()
```



You should check ggplot2 barplot to learn more. https://ggplot2.tidyverse.org/reference/geom_bar.html

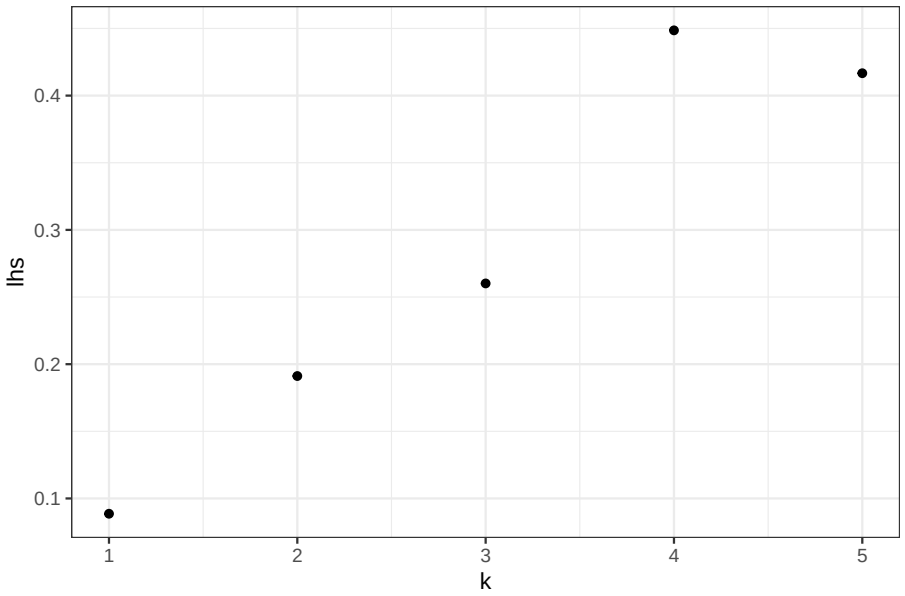
4.2.4 The (a, b, 0) class of distributions

We test whether the data could come from a distribution in the (a, b, 0) class of distributions. Distributions in this family satisfy

$$\frac{k \cdot p_k}{p_{k-1}} = a \cdot k + b, \quad k = 1, \ldots, \infty$$

- `geom_point`: adds a scatterplot to ggplot. Two variables have to be specified in `aes`.
- `xlab`: specifies the name of the label on the x-axis.

```
# We first determine the empirical probabilities p_k
p <- as.numeric(table(Clm_Count) / length(Clm_Count))

# We calculate the left hand side (lhs) of the relation above
lhs <- (1:(length(p)-1)) * p[2:length(p)] / p[1:(length(p)-1)]

ggplot() +
  geom_point(aes(x = 1:(length(p)-1), y = lhs)) +
  xlab('k') +
  ylab('lhs') +
  theme_bw()
```



You should check ggplot2 geom_point to learn more. https://ggplot2.tidyverse.org/reference/geom_point.html

The observations $(k, \frac{k \cdot p_k}{p_{k-1}})$ seem to be on a straight line with positive intercept. This indicates that the Negative Binomial distribution might be a good fit for the data.

4.3 Fitting count distributions

We fit several count distributions to the observed claim count data:

- Poisson
- Negative binomial (NB)
- Modified Poisson distributions

We do not consider the explanatory variables, but take the exposure into account. We fit these distributions using Maximum Likelihood Estimation (MLE).

4.3.1 Poisson

For a Poisson distribution with intensity λ the probability of observing k events is given by

$$P(N = k) = \exp(-\lambda)\frac{\lambda^k}{k!}.$$

The expected value of the poisson distribution is

$$E(N) = \lambda.$$

Not all policyholders are insured throughout the whole year ($d_i = 1$) and obviously policyholders who are only at risk for a small fraction of the year are less likely to experience a claim. We assume that the claim intensity is proportional to the exposure, i.e.

$$N_i \sim \text{POI}(d_i \cdot \lambda),$$

such that the expected value scales with exposure

$$E(N_i) = d_i \cdot \lambda.$$

We then interpret λ as the expected number of claims for a policyholder who was insured throughout the whole year.

Let m be the number of observations and n_i be the observed number of claims for policyholder i, then the likelihood is given by

$$\mathcal{L}(\lambda) = \prod_{i=1}^{m} P(N_i = n_i) = \prod_{i=1}^{m} \exp(-\lambda \cdot d_i) \cdot \frac{(\lambda \cdot d_i)^{n_i}}{n_i!}$$

and the loglikelihood is

$$\ell(\lambda) = \sum_{i=1}^{m} -\lambda \cdot d_i + n_i \cdot \log(\lambda \cdot d_i) - \log(n_i!).$$

We want to maximize this loglikelihood with respect to λ. We define a function `poisson.loglikelihood` which returns the loglikelihood as a function of λ.

```
poisson.loglikelihood <- function(lambda)
{
  loglikelihood <- sum(-lambda * TLength + Clm_Count * log(lambda * TLength) - lfactori

  return(loglikelihood)
}
```

Unfortunately it is not possible to maximize this function directly in R. We make two small adjustments

1. We will use the `nlm` (non-linear minimizer) function in `R` for finding the maximum likelihood parameter $\hat{\lambda}$. Because, `nlm` is used to minimize a function, we change the routine to return the negative loglikelihood ($-\ell(\lambda)$). Minmizing the negative loglikelihood is equivalent to maximizing the loglikelihood.

2. The parameter λ is restricted to positive values. The built-in algorithms in R look for parameters in the unrestricted domain $(-\infty, \infty)$. We reparametrize the likelihood and optimize for $\beta = \log(\lambda)$, which can take values in $(-\infty, \infty)$.

```
poisson.negLoglikelihood <- function(beta)
{
  lambda = exp(beta)

  return(-poisson.loglikelihood(lambda))
}
```

We use the non-linear minimization function `nlm` to carry out the minimization. This routine requires a starting value for β, which is here simply set to 0. The function returns a list containing the following output (check the help page ?nlm for more details):

- minimum: the value of the estimated minimum of f.
- estimate: the point at which the minimum value of f is obtained.
- gradient: the gradient (first derivative) at the estimated minimum of f. The gradient should be close to zero.
- hessian: the hessian (second derivative) at the estimated minimum of f. The hessian is used to determine confidence bounds for the parameters fitted through maximum likelihood.
- code: an integer indicating why the optimization process terminated. When `code` equals 1 or 2, the algorithm converged and the current estimate is probably the solution. When `code` equals 3, 4 or 5 the algorithm has not converged. For more details see the help page of nlm (?nlm).

```
fit <- nlm(poisson.negLoglikelihood, 0, hessian = TRUE)
```

```
Warning in nlm(poisson.negLoglikelihood, 0, hessian = TRUE): NA/Inf replaced by
maximum positive value
```

```
fit
```

```
$minimum
[1] 50834

$estimate
[1] -1.867

$gradient
```

```
[1] 0.0003585

$hessian
      [,1]
[1,] 15754

$code
[1] 1

$iterations
[1] 7
```

```
# Store the fitted lambda value
poisson.lambda <- exp(fit$estimate)

# Store the minimal value found for the loglikelihood
poisson.loglik <- poisson.loglikelihood(poisson.lambda)
```

The estmiate for λ is identical to the expected value μ_{exp} we calculated earlier by taking exposure into account.

When we fit the data with maximum likelihood we obtain a point estimate $\hat{\lambda}$ for the intensity of the Poisson process However, sometimes (e.g. when doing hypothesis testing) we also need a confidence interval for the actual parameter λ. Under MLE we know that the actual parameters of the distribution are assymptoticaly distributed as (see Chapter 13 on Variance and interval estimation)

$$\lambda \sim \mathcal{N}(\hat{\lambda}, \mathcal{I}^{-1}),$$

where $\mathcal{I}$ denotes the Fisher information matrix. You calculate this matrix as the negative of the Hessian, the matrix with the second order derivatives of the log-likelihood, evaluated in $\hat{\beta}$.

```
# The standard error of the parameter estimate for beta is
beta.se = sqrt(solve(fit$hessian))
```

You can now use the delta method (see Section 13.3 in the book) to construct a confidence interval for the unknown λ. The MLE for λ is obtained from the transformation $\hat{\lambda} = \exp \hat{\beta}$. The corresponding se is calculated as $se_{\hat{\lambda}} = (\exp \hat{\beta})^2 \cdot \text{se}_{\hat{\beta}}$.

We can use this normal approximation to calculate a 95% confidence interval for the actual parameter λ as

$$[\lambda - \Phi^{-1}(0.975) \cdot \sigma_\lambda, \lambda + \Phi^{-1}(0.975) \cdot \sigma_\lambda]$$

,

where Φ is the CDF of the standard normal distributon and σ_λ is the standard error of the λ parameter.

```
# The standard error of the parameter estimate for lambda is
poisson.se = poisson.lambda^2 * beta.se;

# The 95% confidence interval for lambda is
c(poisson.lambda - qnorm(0.975) * poisson.se, poisson.lambda + qnorm(0.975) * poisson.se)
```

```
[1] 0.1542 0.1549
```

4.3.2 Negative binomial

The probability function for the negative binomial distribution is given by

$$Pr(N = k) = \frac{\Gamma(a+k)}{\Gamma(a)k!}\left(\frac{\mu}{\mu+a}\right)^k\left(\frac{a}{\mu+a}\right)^a.$$

We take exposure into account and model $\mu_i = d_i \cdot \mu$, where μ is the expected number of claims for a policyholder who is insured for a full year. The likelihood now contains two parameters a and μ which we have to optimize simultaneously. We define the negative loglikelihood

```
NB.negativeLoglikelihood <- function(beta)
{
  mu <- exp(beta[1])
  a <- exp(beta[2])

  loglikelihood <- sum(lgamma(a + Clm_Count) - lgamma(a) - lfactorial(Clm_Count) + Clm_Count * lo

  return(-loglikelihood)
}
```

In this case it's more important to supply good starting values for the convergence speed of the algorithm. For the Negative Binomial distribution

$$E(X) = \mu \quad \text{and} \quad Var(X) = \mu + \frac{1}{a}\cdot\mu^2.$$

We match the first two moments and set

$$\mu = E(X) \quad \text{and} \quad a = \frac{\mu^2}{Var(X) - \mu}.$$

```
mu.initial <- mu
a.initial <- mu^2 / (var - mu)

fit <- nlm(NB.negativeLoglikelihood, log(c(mu.initial, a.initial)),hessian=TRUE)
```

```r
# Store the fitted values
nb.mu <- exp(fit$estimate[1])
nb.a <- exp(fit$estimate[2])

# Store the minimal value found for the loglikelihood
nb.loglik <- -fit$minimum
```

4.3.3 Modified Poisson distributions

We consider two popular modifications of the Poisosn distribution, namely the Zero Inflated Poisson (ZIP) distribution and the Hurdle Poisson distribution.

4.3.3.1 Zero Inflated Poisson (ZIP)

The ZIP distribution is a Poisson distribution where the probability of having zero claims is increased by p.

$$P(N = k) = \begin{cases} p + (1-p) \cdot P(\tilde{N} = 0) & k = 0 \\ (1-p) \cdot P(\tilde{N} = k) & k > 0 \end{cases},$$

where $\tilde{N}$ follows a Poisson distribution. The ZIP distribution is the mixture of a degnerate distribution in zero with weight p and a Poisson distribution with weight $1 - p$.

The parameter p can take values in $[0, 1]$, we transform the interval $[0, 1]$ to the real line $(-\infty, \infty)$ using the logit transform

$$\text{logit}(p) = \log\left(\frac{p}{1-p}\right) = \beta.$$

Inverting this expression, we find

$$p = \frac{\exp(\beta)}{1 + \exp(\beta)}.$$

```r
ZIP.negativeLoglikelihood <- function(beta)
{
  lambda <- exp(beta[1])
  p <- exp(beta[2])/(1+exp(beta[2]))

  density <- (p + (1-p) * exp(-TLength * lambda))^(Clm_Count == 0) * ((1-p) * exp(-TLe

  loglikelihood <- sum(log(density))
```

```
  return(-loglikelihood)
}
```

```
fit <- nlm(ZIP.negativeLoglikelihood, c(0, 0),hessian=TRUE)
```

```
Warning in nlm(ZIP.negativeLoglikelihood, c(0, 0), hessian = TRUE): NA/Inf
replaced by maximum positive value

Warning in nlm(ZIP.negativeLoglikelihood, c(0, 0), hessian = TRUE): NA/Inf
replaced by maximum positive value
```

```
fit
```

```
$minimum
[1] 50663

$estimate
[1] -1.3754 -0.4551

$gradient
[1] -0.02996  0.02012

$hessian
      [,1]  [,2]
[1,] 14607 -5623
[2,] -5623  2423

$code
[1] 1

$iterations
[1] 13
```

```
ZIP.lambda <- exp(fit$estimate[1])
ZIP.p <- exp(fit$estimate[2])/(1+exp(fit$estimate[2]))
c(lambda = ZIP.lambda, p = ZIP.p)
```

```
lambda      p
0.2527 0.3881
```

```
ZIP.loglik <- -fit$minimum
```

Many policyholders file zero claims, which is captured by increasing the probability of observing zero claims by 38.8%.

4.3.3.2 Hurdle Poisson

In the Hurdle Poisson we set the probability of observing zero claims to p. Conditional on there being a claim the distribution follows a zero-truncated Poisson distribution. The probability of observing k claims becomes

$$P(N = k) = \begin{cases} p & k = 0 \\ (1-p) \cdot P(\tilde{N} = k \mid \tilde{N} > 0) & k > 0 \end{cases},$$

where $\tilde{N}$ follows a Poisson distribution. The probability distribution of the zero-truncated Poisson distribution is given by

$$P(\tilde{N} = k \mid \tilde{N} > 0) = \frac{P(\tilde{N} = k)}{P(\tilde{N} > 0)} = \frac{P(\tilde{N} = k)}{1 - \exp(-\lambda)}.$$

We assume that the intensity of the zero-truncated Poisson distribution is proportional to the exposure, i.e. $\lambda_i = d_i \cdot \lambda$. The probability of observing zero claims is p and does not depend on the exposure d_i.

```
Hurdle.negativeLoglikelihood <- function(beta)
{
  lambda <- exp(beta[1])
  p <- exp(beta[2])/(1+exp(beta[2]))

  density <- (p)^(Clm_Count == 0) * ((1-p) * exp(-TLength * lambda) / (1-exp(-lambda *

  loglikelihood <- sum(log(density))

  return(-loglikelihood)
}
```

```
fit <- nlm(Hurdle.negativeLoglikelihood, c(0, 0),hessian=TRUE)
```

```
Warning in nlm(Hurdle.negativeLoglikelihood, c(0, 0), hessian = TRUE): NA/Inf
replaced by maximum positive value

Warning in nlm(Hurdle.negativeLoglikelihood, c(0, 0), hessian = TRUE): NA/Inf
replaced by maximum positive value
```

```
fit
```

```
$minimum
[1] 52955

$estimate
[1] -1.381  2.324
```

```
$gradient
[1] 4.741e-05 1.088e-02

$hessian
           [,1]       [,2]
[1,]  1.541e+03 -3.131e-04
[2,] -3.131e-04  1.299e+04

$code
[1] 1

$iterations
[1] 12
```

```
Hurdle.lambda <- exp(fit$estimate[1])
Hurdle.p <- exp(fit$estimate[2])/(1+exp(fit$estimate[2]))
c(lambda = Hurdle.lambda, p = Hurdle.p)
```

```
lambda      p
0.2513 0.9108
```

```
Hurdle.loglik <- -fit$minimum
```

4.4 AIC

Suppose that we have a statistical model of some data. Let k be the number of estimated parameters in the model. Let $\hat{L}$ be the maximum value of the likelihood function for the model. Then the AIC value of the model is the following

$$\text{AIC} = 2k - 2\ln(\hat{\mathcal{L}})$$

Given a set of candidate models for the data, the preferred model is the one with the minimum AIC value. Thus, AIC rewards goodness of fit (as assessed by the likelihood function), but it also includes a penalty that is an increasing function of the number of estimated parameters. The penalty discourages overfitting, because increasing the number of parameters in the model almost always improves the goodness of the fit. For more information see wikipedia.

```
AIC <- round(c("AIC Poi" = -2 * poisson.loglik + 2 * 1,
               "AIC NB" = - 2 * nb.loglik + 2 * 2,
               "AIC ZIP" = -2 * ZIP.loglik + 2 * 2,
               "AIC Hurdle" = -2 * Hurdle.loglik + 2 * 2))
AIC
```

```
   AIC Poi     AIC NB    AIC ZIP AIC Hurdle
    101670     101318     101330     105914
```

```
AIC[which.min(AIC)]
```

```
AIC NB
101318
```

The lowest AIC value is achieved by the NB distribution, closely followed by the ZIP. The Hurdle distribution attains a remarkably higher AIC value which reflects the poor way in which the exposure is incorporated.

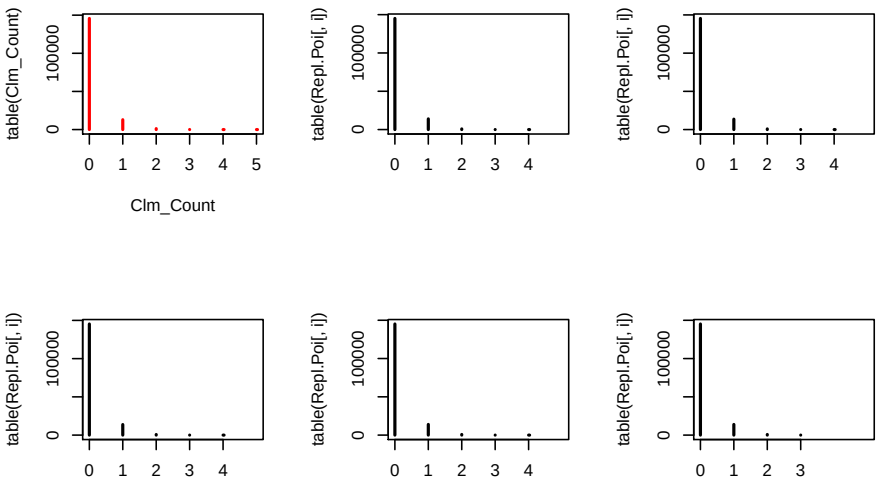
4.5 Replicating data sets

We now show how to generate replicating data sets based on each of these models. We generate 5 random samples of the same size as the original data set and using the estimated parameters. We then plot the contingency tables to compare.

4.5.1 Poisson

`rpois` allows us to generate from a Poisson distribution for any given positive lambda. We use the fitted lambdas, taking exposure into account. Notice that the length of argument lambda is n whereas we sample $5n$ observations. `rpois` deals with this mismatch by recycling the lambdas 5 times. The resulting output vector is then used to form a matrix with 5 columns, filled by its columns (by default), such that each column contains a replicating data set.

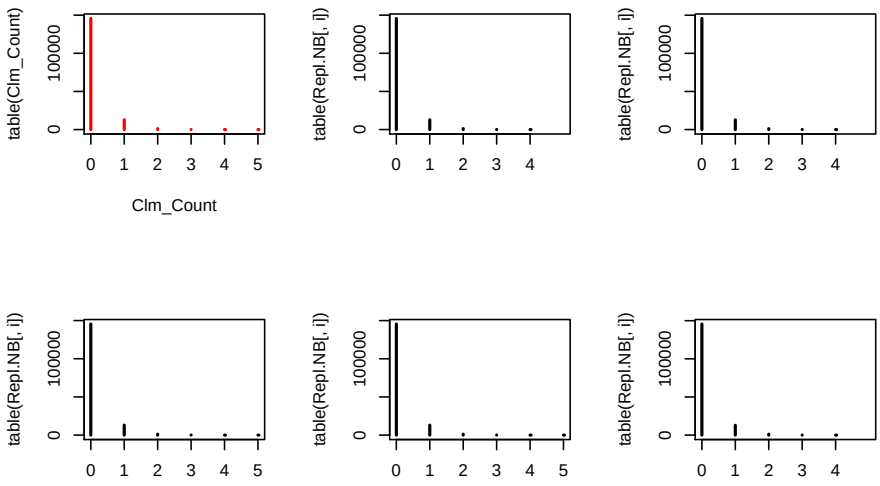
```
n <- length(Clm_Count)
Repl.Poi <- matrix(rpois(n*5, lambda = poisson.lambda*TLength), c(n,5)) # or fitted(fm

par(mfrow=c(2,3))
plot(table(Clm_Count), col="red")
for(i in 1:5){
  plot(table(Repl.Poi[,i]), xlim=c(0,5))
}
```



4.5.2 NB

`rnbinom` allows us to generate from a Negative Binomial distribution for a given mu and size. The rest is similar to the Poisson case.

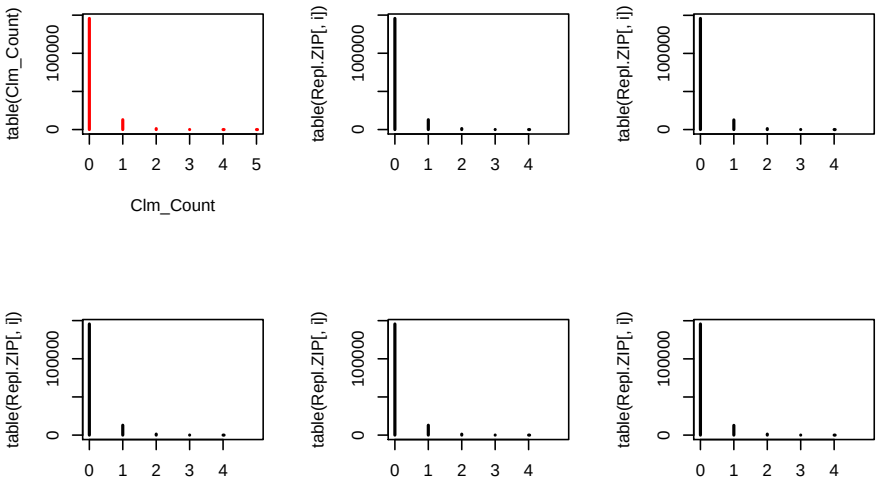
```
Repl.NB <- matrix(rnbinom(n*5, mu = nb.mu * TLength, size = nb.a), c(n,5))

par(mfrow=c(2,3))
plot(table(Clm_Count),col="red")
for(i in 1:5){
  plot(table(Repl.NB[,i]), xlim=c(0,5))
}
```



4.5.3 ZIP

To generate data from the ZIP distribution we first simulate Poisson distributed data. Afterwards, each observation is set to zero with probability p.

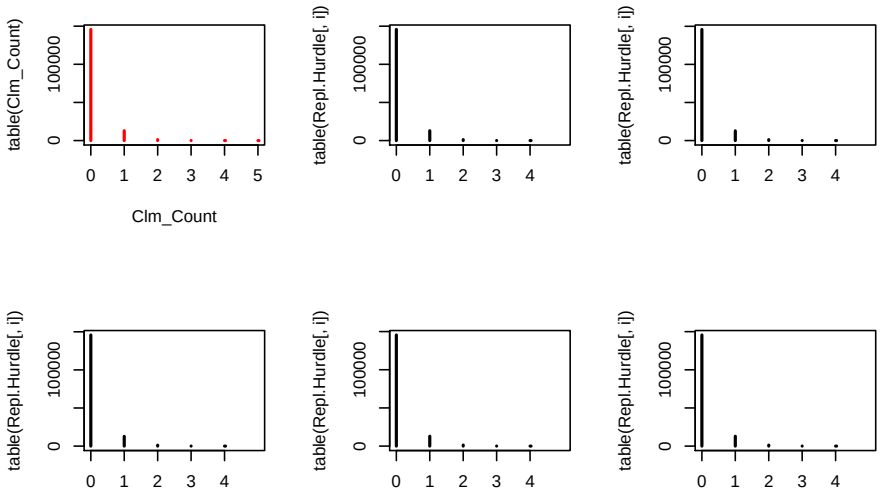
```
# install.packages('VGAM')
# install.packages('gamlss.dist')

Repl.ZIP <- matrix(rpois(n * 5, lambda = ZIP.lambda * TLength) * (runif(n * 5) > ZIP.p

par(mfrow=c(2,3))
plot(table(Clm_Count),col="red")
for(i in 1:5){
  plot(table(Repl.ZIP[,i]), xlim=c(0,5))
}
```



4.5.4 Hurdle Poisson

We first simulate data from a Poisson distribution, where we discard all observations with zero claims. As such, this is a simulation from a zero-truncated distribution. Afterwards, each observation is set to zero with probability p.

```
initial.sim <- rpois(n * 50, lambda = Hurdle.lambda * TLength)
initial.sim <- initial.sim[initial.sim > 0]

Repl.Hurdle <- matrix(initial.sim[1:(n*5)] * (runif(n * 5) > Hurdle.p), c(n, 5))

par(mfrow=c(2,3))
plot(table(Clm_Count),col="red")
for(i in 1:5){
  plot(table(Repl.Hurdle[,i]), xlim=c(0,5))
}
```



4.6 Mean and variance of the estimated ZIP, NB, Hurdle Poisson

We calculate the mean and variance of the estimated Poisson, NB, ZIP, and Hurdle Poisson (with exposure equal to one) and compare these with the empirical mean and variance. We expect that a model fitting the data well will have a similar mean and variance. Try to derive the stated expressions for the mean and variance yourself as an exercise.

4.6.1 Poisson

For the Poisson distribution $N \sim Poi(\lambda)$, the mean and the variance are both equal to λ. This is called equidispersion.

```
poisson.lambda
```

```
[1] 0.1546
```

4.6.2 NB

For the negative binomial distribution $N \sim NB(a, \lambda)$, the mean equals

$$E(N) = \lambda$$

and the variance

$$\text{var}(N) = \lambda + \lambda^2/a$$

exceeds the mean (overdispersion).

```
NB.mean <- nb.mu
NB.mean
```

```
[1] 0.1546
```

```
NB.var <- nb.mu + nb.mu^2 / nb.a
NB.var
```

```
[1] 0.171
```

4.6.3 ZIP

For the zero inflated Poisson distribution $N \sim ZIP(p, \lambda)$, the mean equals

$$E(N) = (1-p)\lambda$$

and the variance

$$\text{var}(N) = (1-p)(\lambda^2+\lambda) - (1-p)^2\lambda^2 = E(N) + \frac{p}{1-p}E(N)^2.$$

From the last expression we notice that the ZIP is overdispersed.

```
ZIP.mean <- (1-ZIP.p)*ZIP.lambda
ZIP.mean
```

```
[1] 0.1546
```

```
ZIP.var <- (1-ZIP.p)*(ZIP.lambda^2+ZIP.lambda) - (1-ZIP.p)^2*ZIP.lambda^2
ZIP.var <- ZIP.mean + ZIP.p/(1-ZIP.p)*ZIP.mean^2
ZIP.var
```

```
[1] 0.1698
```

4.6.4 Hurdle Poisson

For the hurdle Poisson distribution $N \sim \text{Hurdle}(p, \lambda)$, the mean equals

$$E(N) = \frac{1-p}{1-e^{-\lambda}}\lambda$$

and the variance

$$\text{var}(N) = \frac{1-p}{1-e^{-\lambda}}(\lambda^2+\lambda) - E(N)^2 = E(N) + \frac{p-e^{-\lambda}}{1-p}E(N)^2.$$

From the last expression we notice that the Hurdle Poisson is overdispersed if $p > e^{-\lambda}$, or, if the probability mass at zero is larger than it would be under a regular Poisson setting.

```
# note that Hurdle.p = 1-fhurdle.p = q = P(N=0)

Hurdle.mean <- (1-Hurdle.p)/(1-exp(-Hurdle.lambda))*Hurdle.lambda
Hurdle.mean
```

```
[1] 0.1009
```

```
Hurdle.var <- (1-Hurdle.p)/(1-exp(-Hurdle.lambda))*(Hurdle.lambda^2+Hurdle.lambda) - Hu
Hurdle.var <- Hurdle.mean + (Hurdle.p-exp(-Hurdle.lambda))/(1-Hurdle.p)*Hurdle.mean^2
Hurdle.var
```

```
[1] 0.116
```

4.6.5 Comparison with empirical mean and variance

```
means <- c("mean Obs" = mu, "mean Poisson" = poisson.lambda, "mean NB" = NB.mean, "mean
means
```

```
    mean Obs mean Poisson      mean NB     mean ZIP  mean Hurdle
      0.1546       0.1546       0.1546       0.1546       0.1009
```

All distributions expect the hurdle distribution closely approximate the mean of the data.

```
variances <- c("variance Obs" = var, "variance Poisson" = poisson.lambda, "variance NB"
variances
```

```
    variance Obs variance Poisson      variance NB     variance ZIP
          0.1675           0.1546           0.1710           0.1698
 variance Hurdle
          0.1160
```

The hurlde distribution also severly underestimates the variance in the data. Since the NB distribution and the ZIP have two parameters they are more flexibility and can capture both the mean and variance well.

4.7 Conclusion

We have investigated the number of claims per policyholder when taking exposure into account. All of our analyses show that the NB distribution is a good fit. The NB has the lowest AIC, the mean and variance of the data are well captured and relation of the (a, b, 0) class

$$\frac{k \cdot p_k}{p_{k-1}} = a \cdot k + b,$$

seems to be satisfied for a positive value of a.

Chapter 5

Simulation

You will now focus on simulation methods. You will first learn how to draw simulations from a specified, parametric distribution using the Probability Integral Transform (PIT) technique. Then, you will apply your insights to a real-life example where the future lifetimes of newborns are simulated using a recent life table published by Statistics Belgium.

5.1 Severity

In this exercise you will simulate 100 losses X from a Pareto distribution with distribution function

$$F_X(x) = 1 - \left(\frac{1000}{1000 + x}\right)^3.$$

You will use these simulated observations to calculate the expected loss with a deductible of 300, i.e. $E[(X - 300)_+]$.

5.1.1 Probability integral transform

The probability integral transform (PIT) refers to the following property (see Section 20.1 in the Loss Models book)

> If a random variable X has a continuous distribution with CDF F_X. Then the random variable Y defined as $Y = F_X(X)$ has a uniform distribution.

A modified version of this property is often used when simulating data

Given a CDF F_X of a continuous distribution and a uniformly distributed random variable U on $[0,1]$. Then F_X is the CDF of the random variable $X = F_X^{-1}(U)$.

We illustrate this with the exponential distribution,

$$F_Y(y) = \begin{cases} 1 - \exp(-\lambda \cdot y) & y \geq 0 \\ 0 & y < 0 \end{cases}$$

and

$$F_Y^{-1}(z) = -\frac{\log(1-z)}{\lambda}, \quad z \in (0,1).$$

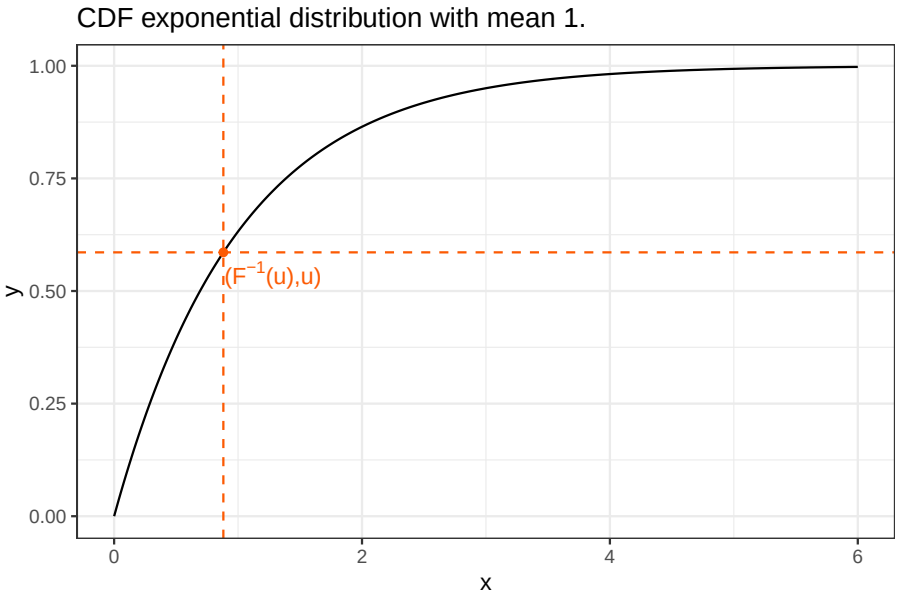
We simulate u from a uniform distribution on $[0,1]$.

```
set.seed(4)
u <- runif(1)
```

When we now apply the inverse cdf on this simulation u, we obtain a simulation from the exponential distribution.

```
Finverse <- function(z, lambda) {
  -log(1-z) / lambda
}

lambda <- 1
sim <- Finverse(u, lambda)

print(c(u = u, sim = sim))
```

```
     u    sim
0.5858 0.8814
```



For more information see probability integral transform and inverse transform sampling. Because of this property, you can follow the steps below to simulate n independent observations from a distribution with CDF F_X:

1. Simulate n indepdendent observations $u_1, \ldots, u_n$ from a uniform distribution (`runif` in R) on $[0, 1]$;
2. Calculate F_X^{-1};
3. $F_X^{-1}(u_1), \ldots, F_X^{-1}(u_n)$ are n independent observations with CDF F_X.

Now follow these steps in R and generate 100 independent observations from a Pareto distribution.

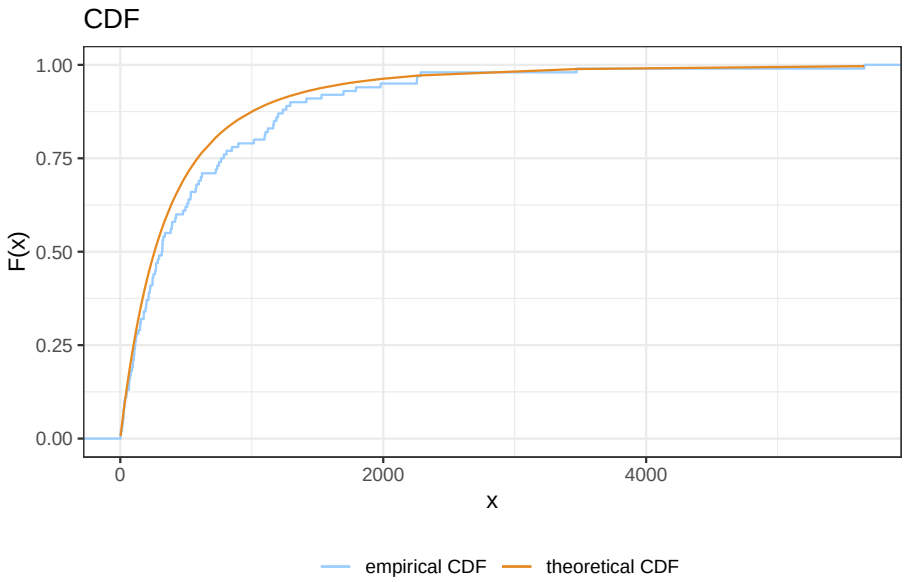
1. Create a vector `u` containing 100 independent observations from a uniform distribution;
2. Compute F_X^{-1} on paper and implement F_X^{-1} as a function in R;
3. Calculate $F_X^{-1}(u)$.

```
# @1.
n <- 100;
u <- runif(n, 0, 1)


# @2.
cdf.inverse <- function(y)
{
  return(1000 / (1-y)^{1/3} - 1000)
}
```

```
# @3.
sim <- cdf.inverse(u)
```

5.1.2 Visualization

You will create the following visualization to compare the actual and empirical CDF.



1. Create a function `cdf` in R with input a vector x and output a vector $F_X(x)$;
2. Compute the theoretical cdf in the simulated points and store the result in a vector y;
3. Load the package `ggplot`;
4. Complete the following `ggplot` command to plot the empiciral and theoretical CDF.

- `col = "empirical CDF"` selects a different color for each line and adds a legend to the figure;
- `scale_colour_manual` specifies the colors to be used in in the figure. The first argument specifies the title for the legend;

```
ggplot() +
  stat_ecdf(aes(???, col = "empirical CDF")) +
  geom_line(aes(???, ???, col = "theoretical CDF")) +
  scale_colour_manual("legend title", values=c("#99CCFF","#e58920"))
```

5. `theme(...)` changes many visual aspects of a ggplot figure. Search in the documentation for an option to move the legend to the bottom of the figure.

6. Improve your graph by adding options such as `theme_bw()`, `xlab`, `ylab`, `ggtitle`, ...

5.1.3 Expected loss with a deductible

Statistical quantities of the distribution can be approximated by means of the simulated data. You will now calculate the expected loss when there is a deductible of 300 using your simulated data.

1. Write a function `deductible` taking as input `x` (a vector containing the loss amounts) and `d` (the deductible) and returning a vector y with $y_i = (x_i - d)_+ = \max(x_i - d, 0)$;

2. Test the function `deductible` that you defined in the previous step. What is the output for `dedutible(c(500, 200), 300)`. Is your function working as expected? A common mistake in implementing the `deductible` function is a misunderstanding of the `max` function in R;

```
  # returns the maximum of 1, 3 and 2
  max(c(1, 3), 2)
```

```
[1] 3
```

```
  # returns a vector containing max(1, 2) and max(3, 2)
  pmax(c(1, 3), 2)
```

```
[1] 2 3
```

3. Calculate $E(X - 300)_+$ for your simulated vector.

```
deductible <- function(x, d)
{
  return(pmax(x-d, 0))
}

mean(deductible(sim, 300))
```

```
[1] 382.3
```

5.2 Aggregate loss

The number of claims N on an insurance contract is Poisson distributed with mean 2. The claim sizes X are independently distributed with CDF

$$F_X(x) = \begin{cases} 0 & x \leq 0 \\ x^2 & 0 < x \leq 1 \\ 1 & x > 1 \end{cases}.$$

The insurer wants to investigate the difference between imposing an aggregate or an ordinary deductible for this contract. You will calculate (using 10000 simulations) the expected loss by contract for both an aggregate deductible of 0.2 and an ordinary deductible of 0.2.

1. Simulate 10000 observations for the number of claims N. (Hint `rpois`);
2. Use what you learnt in the previous exercise and create a function `simulate`, which returns `n` simulations of the claim size distribution F_X.

```
simulate <- function(n)
{
  ??
}
```

3. Complete the following for-loop to simulate 10000 aggregate losses when there is no deductible

```
n <- 10000
nclaim <- # Result from 1.

# create an empty vector to store the aggregated loss
aggregated.loss <- rep(0, n)

for(i in 1:n)
{
  # the claims for the i-th contract
  claims <- simulate(nclaim[i])

  # without deductible the aggregated loss is the sum of the individual claims
  aggregated.loss[i] <- sum(claims)
}

# Calculate the expected loss per policy, when there is no deductible
mean(aggregated.loss)
```

4. Adapt the code in 3. to calculate the expected loss per policy when there is an aggregate deductible of 0.2;
5. Adapt the code in 3. to calculate the expected loss per policy when there is an ordinary deductible of 0.2.

```
# @1
n <- 10000
nclaim <- rpois(n, lambda = 2)
```

```r
# @2
cdf.inverse <- function(x)
{
  return(sqrt(x))
}

simulate <- function(n)
{
  u <- runif(n)
  return(cdf.inverse(u));
}

# @3
n <- 10000
nclaim <- rpois(n, lambda = 2)

# create an empty vector to store the aggregated loss
aggregated.loss <- rep(0, n)

for(i in 1:n)
{
  # the claims for the i-th contract
  claims <- simulate(nclaim[i])

  # without deductible the aggregated loss is the sum of the individual claims
  aggregated.loss[i] <- sum(claims)
}

# Calculate the expected loss per policy, when there is no deductible
mean(aggregated.loss)
```

```
[1] 1.328
```

```r
# @4
n <- 10000
nclaim <- rpois(n, lambda = 2)

# create an empty vector to store the aggregated loss
aggregated.loss <- rep(0, n)

for(i in 1:n)
{
  # the claims for the i-th contract
  claims <- simulate(nclaim[i])
```

```
  # without deductible the aggregated loss is the sum of the individual claims
  aggregated.loss[i] <- sum(claims)
}

# Calculate the expected loss per policy, when there is no deductible
mean(pmax(aggregated.loss - 0.2, 0))
```

```
[1] 1.16
```

```
# @5
n <- 10000
nclaim <- rpois(n, lambda = 2)

# create an empty vector to store the aggregated loss
aggregated.loss <- rep(0, n)

for(i in 1:n)
{
  # the claims for the i-th contract
  claims <- pmax(simulate(nclaim[i]) - 0.2, 0)

  # without deductible the aggregated loss is the sum of the individual claims
  aggregated.loss[i] <- sum(claims)
}

# Calculate the expected loss per policy, when there is no deductible
mean(aggregated.loss)
```

```
[1] 0.9413
```

5.3 Simulating future life times of newborns

In this exercise you simulate the age at death for 10000 newborns based on the lifetable for Belgian males in 2017 published by stabel. You will use these simulations to construct a histogram as well as a density plot of the lifetime distribution.

5.3.1 Importing the data

Download and save the file lifetableMaleBE2017.csv on your local drive. This file contains the columns `age` and `q`, where

$$q_{age} = P(T_0 \in (age, age + 1] \mid T_0 > age),$$

with T_0 the random variable describing the lifetime of a newborn. Hence, q_{age} is the one year probability of dying at age age.

1. Save the file lifetableMaleBE2017.csv on your local drive;
2. Use `read.table` to load the data into R. Semicolons `;` separate the data and the first line of the data contains the variable names;
3. Create variables `age` and `q` which store the corresponding variables from the dataset.

```
setwd('C:/Users/u0095171/Dropbox/Verzekeringen niet-leven/Bookdown/')
data <- read.table('data/lifetableMaleBE2017.csv', sep = ';', header = TRUE)

age <- data$age;
q <- data$q;
```

5.3.2 Simulate the whole life time

First, you simulate the whole remaining lifetime $K_0 = \lfloor T_0 \rfloor$ for a newborn. The whole lifetime is the integer age at which the newborn dies. You calculate the cdf of K_0 at integer ages x as follows

$$\begin{aligned} P(K_0 \leq x) &= P(T_0 < x+1) \\ &= 1 - P(T_0 \geq x+1) \\ &= 1 - P(T_0 \geq x+1 \mid T_0 > x) \cdot P(T_0 > x) \\ &= 1 - (1-q_x) \cdot P(T_0 > x) \\ &= 1 - \prod_{i=0}^{x} (1-q_{x-i}). \end{aligned}$$

or you reason as follows

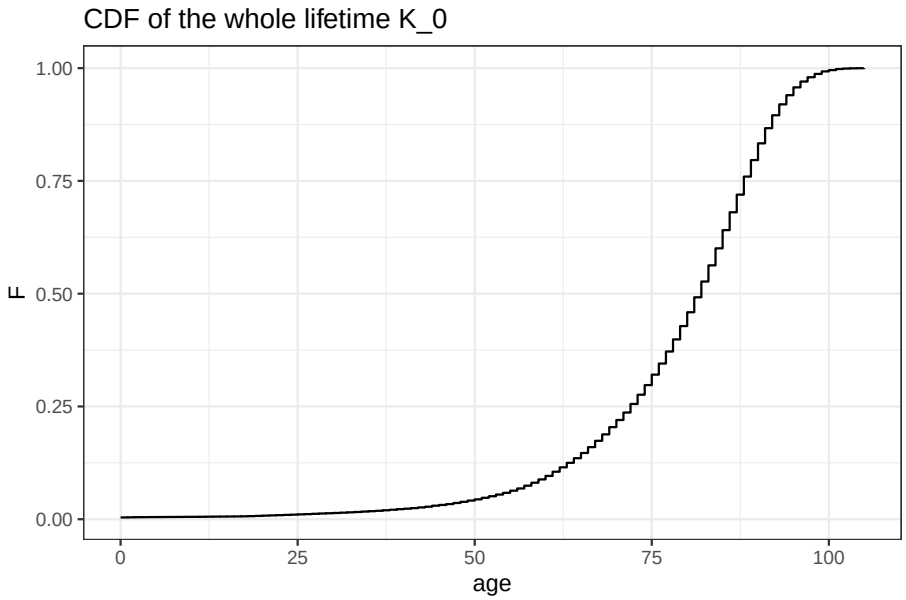
$$\begin{aligned} P(K_0 \leq x) &= P(K_0 = 0) + P(K_0 = 1) + \ldots + P(K_0 = x) \\ &= q_0 + (1-q_0) \cdot q_1 + (1-q_0) \cdot (1-q_1) \cdot q_2 + \ldots + (1-q_0) \cdot (1-q_1) \ldots (1-q_{x-1}) \cdot q_x \\ &= 1 - \prod_{i=0}^{x} (1-q_{x-i}). \end{aligned}$$

1. Create a vector `cdf` in R with $cdf[x+1] = P(K_0 \leq x)$ for $x \in 0, \ldots, 105$. Hint: `cumprod` in R;

```
cumprod(c(2, 3, 5, 7))
```

```
[1]   2   6  30 210
```

The Figure below shows the CDF of the whole lifetime K_0 for a newborn. Since K_0 is a discrete random variable this CDF is a step function.



The following Theorem presents a strategy for simulating from a discrete distribution given its CDF.

> Given a CDF F_X of a discrete distribution with possible outcomes $x_1, \ldots, x_n$ and a uniformly distributed random variable U on $[0, 1]$. Then F_X is the CDF of the random variable $X = \max_{x_i}\{F_X(x_i) \leq U\}$.

2. Complete the code below to simulate a single observation from K_0. This function performs the following steps:

- Simulate an observation u from a random variable $U \in [0, 1]$;
- Loop over all posible ages x in increasing order;
- Find the smallest age x for which $F_{K_0}(x) > u$;
- The previous age $x - 1$ is the largest age for which $F_{K_0}(x) \leq u$. By the theorem above, $x - 1$ is a simulation for the whole age.

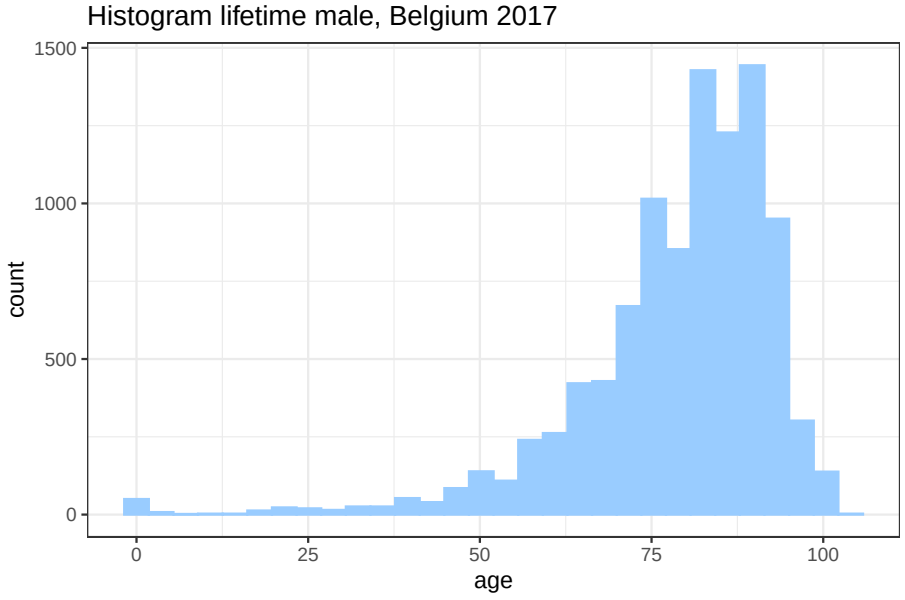
```
simulWholeAge <- function()
{
  # simulate one observation u from a uniform distribution on [0, 1]
  u <- runif(1);

  # loop over all posible values F_X(x_i) in increasing order
  for(j in 1:length(cdf))
  {
    # Check F_X(x_i) > U
    if( ??? )
```

```
    {
      # If F_X(x_i) > u, then the previous value x_{i-1} is the largest value for which F_X(x_{i-
      return( ??? );
    }
  }
}
```

3. Use `simulWholeAge` to generate 10000 observations from the distribution of K_0;

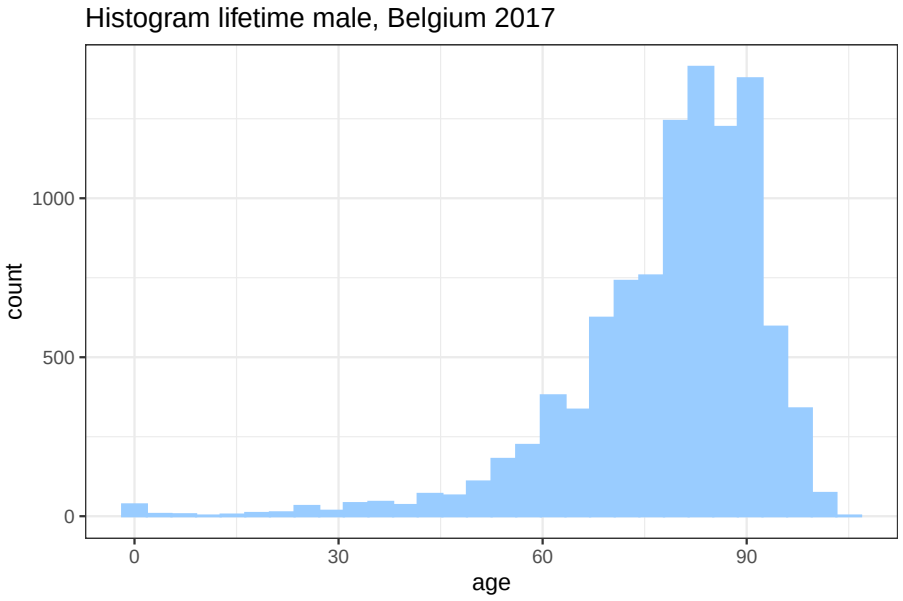
```
# the number of simulations
nsim <- 10000

# initialise a vector with nsim zeroes.
K <- rep(0, nsim)

for(i in 1:nsim)
{
  # Simulate the age of death for a single individual;
  K[i] <- ???
}
```

4. Use `ggplot` to create a histogram of your simulation for the whole life time. See `geom_histogram`.



```
cdf <- 1 - cumprod(1-q)

nsim <- 10000
K <- rep(0, nsim)

simulWholeAge <- function()
{
  u <- runif(1);

  for(j in 1:length(cdf))
  {
    if(u < cdf[j])
    {
      return(j-1);
    }
  }
}

for(i in 1:nsim)
{
  K[i] <- simulWholeAge();
}


ggplot() +
  geom_histogram(aes(K), col = "#99CCFF", fill = "#99CCFF") +
  theme_bw() +
  ggtitle("Histogram lifetime male, Belgium 2017") +
  xlab('age')
```

```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



5.3.3 Simulate the future lifetime

You will now simulate the exact (instead of integer) remaining lifetime T_0 of a newborn. Note that T_0 can be written as (explain!) $T_0 = \lfloor T_0 \rfloor + (T_0 - \lfloor T_0 \rfloor)$ where $\lfloor T_0 \rfloor \in \mathbb{N}$ and $(T_0 - \lfloor T_0 \rfloor) \in [0, 1)$. You will need to make an assumption regarding the probability of dying at non-integer ages. You assume a constant force of mortality between integer ages, i.e.

$$P(T_0 \leq t + x \mid T_0 > x) = 1 - \exp(-\mu_x \cdot t), \quad \text{for} \quad x \in \mathbb{N} \quad \text{and} \quad t \in [0, 1)$$

for some intensity μ_x of dying at age x. More insight behind this assumption will be given in the courses on Life Insurance Mathematics and Advanced Life Insurance Mathematics. The intensity μ_x can be found by taking $t = 1$ in the above expression,

$$1 - \exp(-\mu_x) = P(T_0 \leq 1 + x \mid T_0 > x) = q_x,$$

from which

$$\mu_x = -\log(1 - q_x).$$

From this you compute the distribution of $T_0 - K_0 \mid K_0 = x$,

$$P(T_0 - K_0 \leq t \mid K_0 = x) = P(T_0 \leq x + t \mid T_0 \geq x,\ T_0 \leq x + 1) \tag{5.1}$$

$$= \frac{P(T_0 \leq x + t \mid T_0 \geq x)}{P(T_0 \leq x + 1 \mid T_0 \geq x)} \tag{5.2}$$

$$= \frac{1 - (1 - q_x)^t}{q_x}. \tag{5.3}$$

In the previous section, you simulated an observation for the whole lifetime K_0 of a newborn. You will now simulate a corresponding observation from the distribution of $T_0 - K_0 \mid K_0$. The distribution for $T_0 - K_0 \mid K_0$ is continuous and the classical Probability Integral Transform can be used.

1. Show that the inverted CDF of $T_0 - K_0 \mid K_0$ is given by

$$F^{-1}_{T_0 - K_0 | K_0}(y \mid x) = \frac{\log(1 - q_x \cdot y)}{\log(1 - q_x)};$$

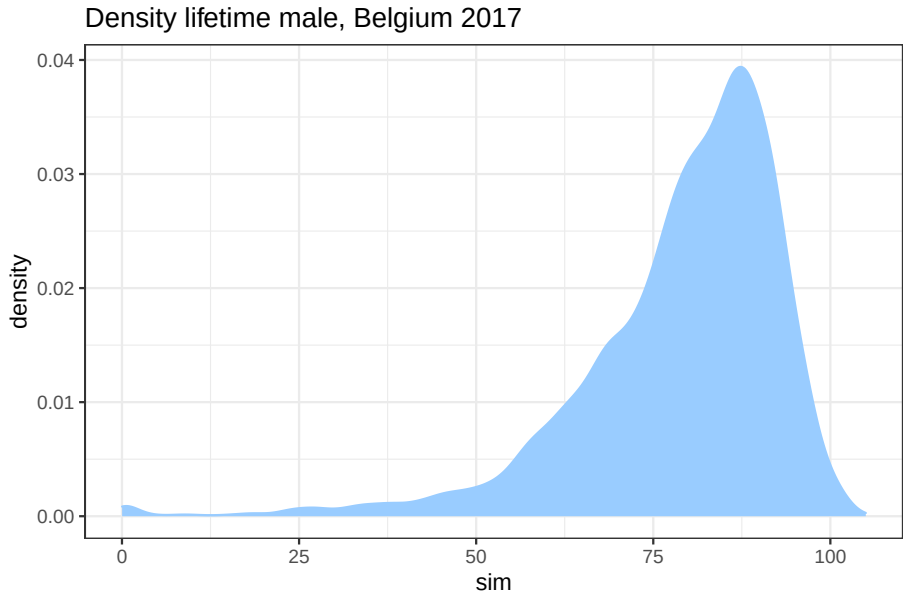
2. Simulate for each observation of K_0 a sample from the distribution $T_0 - K_0 \mid K_0$. Verify that the correct value of q_x is used for each sample;

3. Add the simulations from K_0 and $T_0 - K_0 \mid K_0$ to obtain simulations for the remaining lifetime T_0 of a newborn.

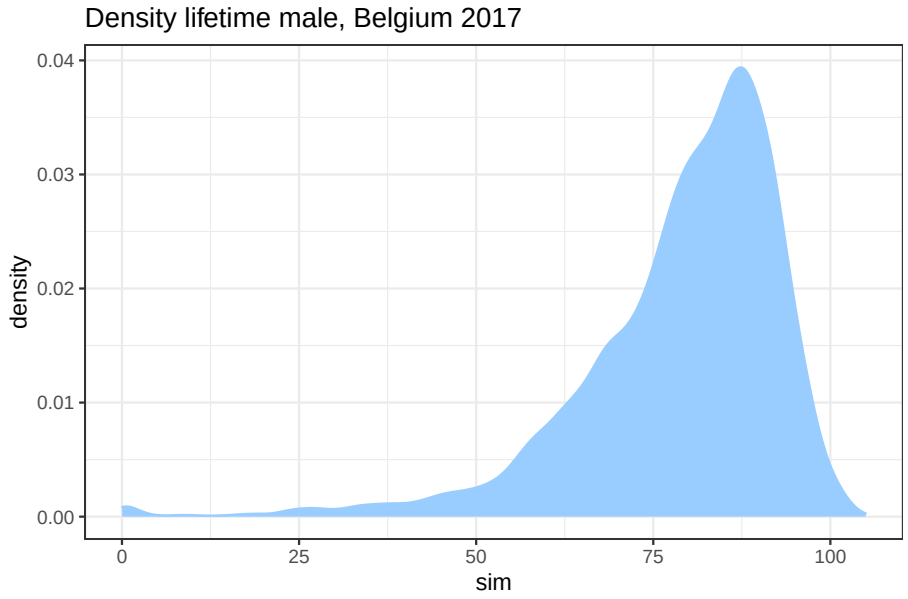
```
# @2
u <- runif(nsim)
remainder <- log(1 - q[K+1] * u) / log(1-q[K+1])

sim <- K + remainder
```

5.3.4 Visualize the data

Visualize the density of the remaining life time T_0 of a newborn with `ggplot`.




```
ggplot() +
  geom_density(aes(sim), col = "#99CCFF", fill = "#99CCFF") +
  theme_bw() +
  ggtitle("Density lifetime male, Belgium 2017")
```



Chapter 6

R implementation of the assignments 2019-2020

In this R tutorial you will implement the exercises from the assignment in R.

6.1 Assignment 1

The solution of the first assignment can be downloaded here.

6.1.1 Exercise 1

In the first exercise you computed a number of statistics for a distribution with cdf

$$F_X(x) = \begin{cases} 0 & x < 0 \\ 1 - 0.5 \cdot e^{-\beta \cdot x} & x \geq 0 \end{cases},$$

for some $\beta > 0$.

You can use the following function to simulate data from this distribution. Simulating from a given cdf will be covered in a later R tutorial.

```
simulate_assignment1 <- function(num_sim, beta) {
  rexp(num_sim, rate = beta) * (runif(num_sim) > 0.5)
}

# this code simulates 10 observations from this distribution with beta = 0.5
simulate_assignment1(10, beta = 0.5)
```

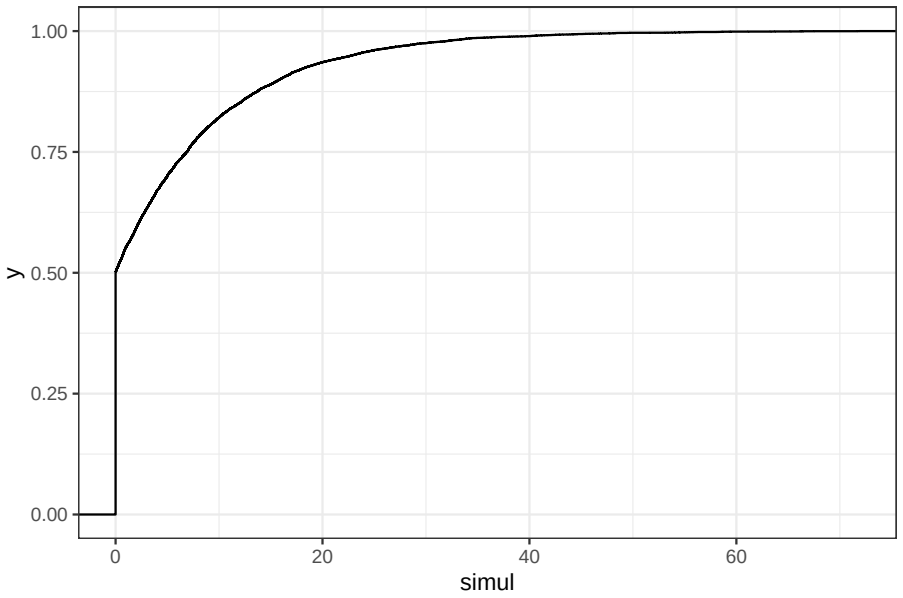
```
 [1] 4.31338 2.72700 0.00000 0.00000 0.00000 4.74313 0.00000 0.09661 0.23204
[10] 0.00000
```

Choose a value for β and simulate `10.000` observations from this distribution. Using this simulation:

1. Plot the empirical cdf of your simulation;
2. Compute $P(X = 0)$;
3. Compute $E(X)$;
4. Compute $\sigma(X) = \sqrt{E((X - E(X))^2)}$.

Compare your answers with the solution of the assignment.

```
# set the seed for replicability when working with random numbers
set.seed(1)
n <- 10000
beta = 0.1
simul <- simulate_assignment1(n, beta)

# @1
ggplot() +
  theme_bw() +
  stat_ecdf(aes(simul))
```




```
# @2
sum(simul == 0) / n
```

```
[1] 0.5018
```

```
#@3
mean(simul)
```

```
[1] 4.922
```

```
0.5/beta
```

```
[1] 5
```

```
#@4
sd(simul)
```

```
[1] 8.595
```

```
sqrt(0.75)/beta
```

```
[1] 8.66
```

Next, we calculate $E((X-30)_+)$. To calculate an expected value of the form $E(g(X))$ from a simulation, we compute

$$\widehat{E(g(X))} = \frac{1}{n}\sum_{i=1}^{n} g(x_i),$$

where x_i are our simulated outcomes from the random variable X.

In the case of the mean $E(X)$, $g(x) = x$, i.e. the identity function. We implement this function in R as

```
identity <- function(x) {
  return(x)
}

mu = 1/n * sum(identity(simul))
print(mu)
```

```
[1] 4.922
```

You will now calculate the expected loss when there is a deductible of 30 using your simulated data.

1. Write a function `deductible` taking as input `x` (a vector containing the loss amounts) and `d` (the deductible) and returning a vector y with $y_i = (x_i - d)_+ = \max(x_i - d, 0)$;
2. Test the function `deductible` that you defined in the previous step. What is the output for `dedutible(c(500, 200), 300)`. Is your function working as expected? A common mistake in implementing the `deductible` function is a misunderstanding of the `max` function in R;

```
  # returns the maximum of 1, 3 and 2
  max(c(1, 3), 2)
```

```
[1] 3
```

```
  # returns a vector containing max(1, 2) and max(3, 2)
  pmax(c(1, 3), 2)
```

```
[1] 2 3
```

3. Calculate $E(X-30)_+$ for your simulated vector.

```
deductible <- function(x, d)
{
  return(pmax(x-d, 0))
}

1/n * sum(deductible(simul, 30))
```

```
[1] 0.2519
```

```
0.5 * exp(-30*beta) / beta
```

```
[1] 0.2489
```

You will now calculate the remaining statistics from exercise 1.

1. $E(X \wedge 30)$
2. $e_X(30)$
3. $VaR_{0.95}(X)$

```
# @1
limit <- function(x, limit) {
  return(pmin(x, limit))
}

mean(limit(simul, 30))
```

```
[1] 4.67
```

```
0.5/beta * (1-exp(-30*beta))
```

```
[1] 4.751
```

```
# @2
simul_excess_30 <- simul[simul > 30] - 30
mean(simul_excess_30)
```

```
[1] 10.12
```

```
1/beta
```

```
[1] 10
```

```
# @3
quantile(simul, 0.95)
```

```
  95%
22.91
```

```
-log(0.1)/beta
```

```
[1] 23.03
```

6.2 Assignment 2

6.2.1 Exercise 2

You will solve a slightly adapted version of this exercise. Assume that the number of claims, N, for an insurance portfolio follows a Poisson distribution with mean $\lambda = 10$. The sizes of these claims are independent and follow a Gamma distribution with mean 1000 and variance 90000.

In this exercise you will compute various statistics related to the total loss.

1. Create a single simulation for the aggregate loss $S = X_1 + ... + X_N$.

```
set.seed(1)
lambda <- 10
mu <- 1000
sigmasq <- 90000;

# Simulate the number of claims from a Poisson distribution
N <- rpois(1, lambda)
```

We have to simulate `N` losses from a gamma distribution with mean 1000 and variance 90000. When we check the documentation of `rgamma` we see that the distribution is parametrized by a `shape` and `scale` parameter.

In the details section of the documentation, you find:

$$shape = a, scale = s, E(X) = a * s \text{ and } Var(X) = a * s^2.$$

From this you can compute:

$$scale = \frac{Var(X)}{E(X)} \text{ and } shape = \frac{E(X)}{scale}$$

```
scale = sigmasq / mu
shape = mu / scale

# Simulate N losses from a gamma distribution
X <- rgamma(N, shape = shape, scale = scale)

S = sum(X)
```

You have now created a single simulation of S. Using a for-loop, you can generate multiple simulations.

```
num_sim <- 1000

# create a vector for storing the result of the simulation
aggregated_loss <- rep(NA, num_sim)

for(i in 1:num_sim) {
  # Add your code for a single simulation here
  N <-
  X <-
  S <-

  aggregated_loss[i] <- S
}
```

You will now update the for-loop above to analyze the aggregated loss.

1. Complete the for-loop and generate 1000 simulations of the aggregated loss;
2. Use these simulations to compute the mean and standard deviation of the aggregated loss;
3. Assume that the insurer imposes an ordinary deductible of 750. Add the variables `N_after_deductible` and `aggregated_loss_deductible` in your simulation, which register the number of claims exceeding the deductible and the aggregate loss after imposing the deductible respectively;
4. Visualize the distribution of the number of claims before and after imposing the deductible;
5. Compare the density of the aggregate loss before and after imposing the deductible.

```
# @1
num_sim <- 1000

# create a vector for storing the result of the simulation
aggregated_loss <- rep(NA, num_sim)

for(i in 1:num_sim) {
  # Add your code for a single simulation here
  N <- rpois(1, lambda)
  X <- rgamma(N, shape = shape, scale = scale)
  S <- sum(X)

  aggregated_loss[i] <- S
}
```

```
# @2
mean(aggregated_loss)
```

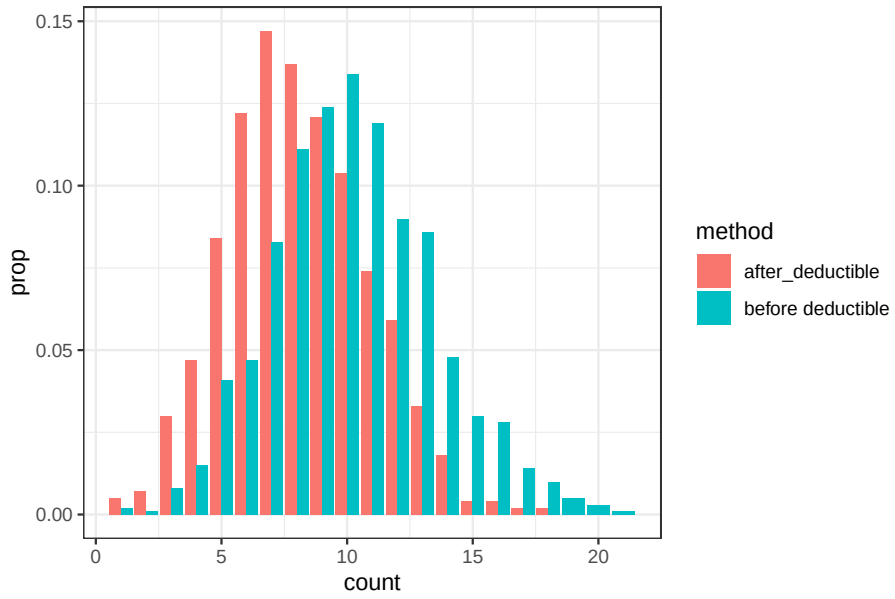
```
[1] 9999
```

```
sd(aggregated_loss)
```

```
[1] 3316
```

```
# @3
num_sim <- 1000

# create a vector for storing the result of the simulation
deductible <- 750
aggregated_loss <- rep(NA, num_sim)
aggregated_loss_deductible <- rep(NA, num_sim)
N <- rep(NA, num_sim)
N_after_deductible <- rep(NA, num_sim)

for(i in 1:num_sim) {
  # Add your code for a single simulation here
  N[i] <- rpois(1, lambda)
  X <- rgamma(N[i], shape = shape, scale = scale)
  S <- sum(X)

  aggregated_loss[i] <- S
  N_after_deductible[i] <- sum(X > deductible)
  aggregated_loss_deductible[i] <- sum(X[X > deductible])
}
```

```
df <- rbind(data.frame(count = N, method = 'before deductible'),
            data.frame(count = N_after_deductible, method = 'after_deductible'))

ggplot(df) +
  theme_bw() +
  geom_bar(aes(count, fill = method, y = ..prop.., group = method), position = position_dodge())
```




```
ggplot() +
  theme_bw() +
  geom_density(aes(aggregated_loss, fill = "before_dectitble"), alpha = .5) +
  geom_density(aes(aggregated_loss_deductible, fill = "after_dectitble"), alpha = .5)
```