Local GLM Net (LGN)

Worker's Compensation Claims

Daniel Meier and Jürg Schelldorfer, with support from Mario V. Wüthrich and Mirai Solutions GmbH.

2021-10-15

Introduction

This notebook was created for the course "Deep Learning with Actuarial Applications in R" of the Swiss Association of Actuaries (https://www.actuaries.ch/).

This notebook serves as accompanion to the tutorial "LocalGLMnet: a deep learning architecture for actuaries", available on SSRN.

The code is similar to the code used in above tutorial and combines the raw R code in the scripts, available on GitHub along with some more comments. Please refer to the tutorial for explanations.

Note that the results might vary depending on the R and Python package versions, see last section for the result of `sessionInfo()` and corresponding info on the Python setup.

Data Preparation

The tutorial uses the synthetically generated worker compensation insurance claims data set available on openML (ID 42876).

Load packages and data

```
library(keras)
library(locfit)
```

```
## locfit 1.5-9.4    2020-03-24
```

```
library(magrittr)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(tibble)
library(purrr)
```

```
##
## Attaching package: 'purrr'

## The following object is masked from 'package:magrittr':
##
##     set_names

## The following object is masked from 'package:locfit':
##
##     none
```

```
library(ggplot2)
library(gridExtra)
```

```
##
## Attaching package: 'gridExtra'

## The following object is masked from 'package:dplyr':
##
##     combine
```

```
library(tidyr)
```

```
##
## Attaching package: 'tidyr'

## The following object is masked from 'package:magrittr':
##
##     extract
```

```
library(corrplot)
```

```
## corrplot 0.90 loaded
```

```
RNGversion("3.5.0")
```

```
## Warning in RNGkind("Mersenne-Twister", "Inversion", "Rounding"): non-uniform
## 'Rounding' sampler used
```

Set global parameters

```
options(encoding = 'UTF-8')
```

```
# set seed to obtain best reproducibility. note that the underlying architecture may affect results non
seed <- 100
Sys.setenv(PYTHONHASHSEED = seed)
set.seed(seed)
reticulate::py_set_seed(seed)
tensorflow::tf$random$set_seed(seed)
```

```
# https://stackoverflow.com/questions/65366442/cannot-convert-a-symbolic-keras-input-output-to-a-numpy-
# https://tensorflow.rstudio.com/guide/tfhub/examples/feature_column/
tensorflow::tf$compat$v1$disable_eager_execution()
```

```
ax_limit <- c(0,50000)
line_size <- 1.1
```

The results below will not exactly match the results in the paper, since some packages and random seeds are different. However, the general conclusions remain the same.

Helper functions

Subsequently, for ease of reading, we provide all the helper functions which are used in this tutorial in this section.

```
# MinMax scaler
preprocess_minmax <- function(varData) {
  X <- as.numeric(varData)
  2 * (X - min(X)) / (max(X) - min(X)) - 1
}
```

```
# One Hot encoding for categorical features
preprocess_cat_onehot <- function(data, varName, prefix) {
  varData <- data[[varName]]
  X <- as.integer(varData)
  n0 <- length(unique(X))
  n1 <- 1:n0
  addCols <- purrr::map(n1, function(x, y) {as.integer(y == x)}, y = X) %>%
    rlang::set_names(paste0(prefix, n1))
  cbind(data, addCols)
}
```

```
#https://stat.ethz.ch/pipermail/r-help/2013-July/356936.html
scale_no_attr <- function (x, center = TRUE, scale = TRUE)
{
    x <- as.matrix(x)
    nc <- ncol(x)
    if (is.logical(center)) {
        if (center) {
            center <- colMeans(x, na.rm = TRUE)
            x <- sweep(x, 2L, center, check.margin = FALSE)
        }
    }
    else if (is.numeric(center) && (length(center) == nc))
        x <- sweep(x, 2L, center, check.margin = FALSE)
    else stop("length of 'center' must equal the number of columns of 'x'")
    if (is.logical(scale)) {
        if (scale) {
            f <- function(v) {
                v <- v[!is.na(v)]
                sqrt(sum(v^2)/max(1, length(v) - 1L))
            }
            scale <- apply(x, 2L, f)
            x <- sweep(x, 2L, scale, "/", check.margin = FALSE)
        }
    }
    else if (is.numeric(scale) && length(scale) == nc)
        x <- sweep(x, 2L, scale, "/", check.margin = FALSE)
    else stop("length of 'scale' must equal the number of columns of 'x'")
    #if (is.numeric(center))
    #    attr(x, "scaled:center") <- center
    #if (is.numeric(scale))
    #    attr(x, "scaled:scale") <- scale
    x
}
```

```r
square_loss <- function(y_true, y_pred){mean((y_true-y_pred)^2)}
gamma_loss  <- function(y_true, y_pred){2*mean((y_true-y_pred)/y_pred + log(y_pred/y_true))}
ig_loss     <- function(y_true, y_pred){mean((y_true-y_pred)^2/(y_pred^2*y_true))}
p_loss      <- function(y_true, y_pred, p){2*mean(y_true^(2-p)/((1-p)*(2-p))-y_true*y_pred^(1-p)/(1-p)+y

k_gamma_loss  <- function(y_true, y_pred){2*k_mean(y_true/y_pred - 1 - log(y_true/y_pred))}
k_ig_loss     <- function(y_true, y_pred){k_mean((y_true-y_pred)^2/(y_pred^2*y_true))}
k_p_loss      <- function(y_true, y_pred){2*k_mean(y_true^(2-p)/((1-p)*(2-p))-y_true*y_pred^(1-p)/(1-p)
```

```r
keras_plot_loss_min <- function(x, seed) {
    x <- x[[2]]
    ylim <- range(x)
    vmin <- which.min(x$val_loss)
    df_val <- data.frame(epoch = 1:length(x$loss), train_loss = x$loss, val_loss = x$val_loss)
    df_val <- gather(df_val, variable, loss, -epoch)
    plt <- ggplot(df_val, aes(x = epoch, y = loss, group = variable, color = variable)) +
      geom_line(size = line_size) + geom_vline(xintercept = vmin, color = "green", size = line_size) +
      labs(title = paste("Train and validation loss for seed", seed),
           subtitle = paste("Green line: Smallest validation loss for epoch", vmin))
    suppressMessages(print(plt))
}
```

```r
plot_size <- function(test, xvar, title, model, mdlvariant) {
  out <- test %>% group_by(!!sym(xvar)) %>%
    summarize(obs = mean(Claim) , pred = mean(!!sym(mdlvariant)))

  ggplot(out, aes(x = !!sym(xvar), group = 1)) +
    geom_point(aes(y = pred, colour = model)) +
    geom_point(aes(y = obs, colour = "observed")) +
    geom_line(aes(y = pred, colour = model), linetype = "dashed") +
    geom_line(aes(y = obs, colour = "observed"), linetype = "dashed") +
    ylim(ax_limit) + labs(x = xvar, y = "claim size", title = title) +
    theme(legend.position = "bottom")
}
```

Load data

This dataset describes 100'000 realistic, synthetically generated worker compensation insurance claims. Along the ultimate financial losses, each claim is described by the initial case estimate, date of accident and reporting date, a text describing the accident and demographic info on the worker. The dataset was kindly created and provided by Colin Priest. While similar, it is not identical to the dataset used in www.kaggle.com/c/actuarial-loss-estimation.

Before we can use this data set we need to do some data cleaning. This pre-processed data can be downloaded from the course GitHub page here. You also find at the same place the details for this pre-processing.

```r
load(file.path("../0_data/WorkersComp.RData"))  # relative path to .Rmd file
```

General data preprocessing

We drop 10 claims with claim occurrence year 1987. This provides us with 99'990 claims, all having occurred between 1988/01/01 and 2006/07/20. These claims have been reported in the calendar years from 1998 to 2008. The maximal reporting delay is 1'042 days, this corresponds to 2.85 years or to almost 149 weeks. We drop all claims with non-positive HoursWorkedPerWeek.

```
dat <- WorkersComp %>% filter(AccYear > 1987, HoursWorkedPerWeek > 0)
```

```
# Order claims in decreasing order for split train/test (see below), and add an ID
dat <- dat %>% arrange(desc(Claim))
dat <- dat %>% mutate(Id=1:nrow(dat))
```

For successful regression modeling we need to pre-process (raw) covariate information. Moreover, continuous and binary variables are normalized to having zero mean and unit variance, i.e., having raw covariates, the j-th raw covariate component is normalized by

$$x_{i,j}^{\text{raw}} \;\mapsto\; x_{i,j} = \frac{x_{i,j}^{\text{raw}} - \text{mean}(x_{i,j}^{\text{raw}})}{\text{stddev}(x_{i,j}^{\text{raw}})},$$

where the empirical mean and the empirical standard deviation (stddev) are calculated for fixed component j over all available claims i.

$\Rightarrow$ Standardization of continuous and binary variables to zero mean and unit variance is going to be important because comparison of variables requires that they live on the same scale.

```
# scaling and cut-off
dat <- dat %>% mutate(
        Age = pmax(16, pmin(70, Age)),
        AgeNN = scale_no_attr(Age),
        GenderNN = as.integer(Gender),
        GenderNN = scale_no_attr(GenderNN),
        DependentChildren = pmin(1, DependentChildren),
        DependentChildrenNN = scale_no_attr(DependentChildren),
        DependentsOther = pmin(1, DependentsOther),
        DependentsOtherNN = scale_no_attr(DependentsOther),
        WeeklyPay = pmin(1200, WeeklyPay),
        WeeklyPayNN = scale_no_attr(WeeklyPay),
        PartTimeFullTimeNN = scale_no_attr(as.integer(PartTimeFullTime)),
        HoursWorkedPerWeek = pmin(60, HoursWorkedPerWeek),
        HoursWorkedPerWeekNN = scale_no_attr(HoursWorkedPerWeek),
        DaysWorkedPerWeekNN = scale_no_attr(DaysWorkedPerWeek),
        AccYearNN = scale_no_attr(AccYear),
        AccMonthNN = scale_no_attr(AccMonth),
        AccWeekdayNN = scale_no_attr(AccWeekday),
        AccTimeNN = scale_no_attr(AccTime),
        RepDelay = pmin(100, RepDelay),
        RepDelayNN = scale_no_attr(RepDelay)
)
```

Exercise: We cut several variables at a maximal value. Exclude this in order to understand why we have applied an upper threshold for these variables using the subsequent statistics and charts.

Categorical (nominal) variables with more than two levels are pre-processed by one-hot encoding. Assume that the j-th raw covariate component is a categorical variable with L levels labeled by $c_1, c_2, ..., c_L$. One-hot encoding maps these L levels to the L unit vectors of the Euclidean space $\mathbb{R}^L$. Thus, having raw categorical variable, we receive its one-hot encoding by the embedding

$$x_j^{\text{raw}} \;\mapsto\; \left(\mathbb{1}_{\{x_j^{\text{raw}}=c_1\}}, \ldots, \mathbb{1}_{\{x_j^{\text{raw}}=c_L\}}\right)^\top \in \mathbb{R}^L.$$

Note that one-hot encoding is different from dummy coding. Dummy coding selects a reference level and maps the levels different from the reference level to the $L-1$ unit vectors of the Euclidean space $\mathbb{R}^{L-1}$. Dummy coding leads to full rank design matrices which is important in GLM. One-hot encoding does not give full rank design matrices because we have one redundancy in this encoding, however, this is exactly needed, here, and it will be explained below.

$\Rightarrow$ We require one-hot encoding for categorical variables with more than two levels. This does not lead to full rank design matrices, however, this is not a necessary property in the LocalGLMnet approach.

```
# one-hot encoding (not dummy encoding!)
dat <- dat %>% preprocess_cat_onehot("MaritalStatus", "Marital")
```

In view of variable selection (see below), we need to understand whether for some components j we have to analyze the null hypothesis H_0 for that j. In order to make this decision we extend the original covariates xi by additional information that does not belong to our data.

These two additional components will be taken purely at random and, thus, they cannot contain any systematic effects explaining the response Y.

```
# add two additional randomly generated features (later used)
set.seed(seed)

dat <- dat %>% mutate(
    RandNN = rnorm(nrow(dat)),
    RandNN = scale_no_attr(RandNN),
    RandUN = runif(nrow(dat), min = -sqrt(3), max = sqrt(3)),
    RandUN = scale_no_attr(RandUN)
)
```

Inspect the prepared dataset

```
head(dat)
```

```
##       ClaimNumber Age Gender MaritalStatus DependentChildren DependentsOther
## 96757   WC9730090  53      F             M                 0               0
## 43824   WC5305468  37      F             U                 0               0
## 54440   WC6197732  18      M             S                 0               0
## 79781   WC8318363  34      M             S                 0               0
## 87997   WC9001287  26      M             U                 0               0
## 67886   WC7321088  26      M             U                 0               0
##       WeeklyPay PartTimeFullTime HoursWorkedPerWeek DaysWorkedPerWeek
## 96757       683               F                 40                5
## 43824       535               F                 38                5
## 54440       250               F                 38                5
## 79781       544               F                 38                5
## 87997       435               F                 37                5
## 67886       530               F                 20                4
##                                           ClaimDescription
## 96757         BUFFING OUT DISC CUTTING RIGHT UPPER SWOLLEN HEAD
## 43824 FELL HEAD ON STEP PULLED TWISTED SOFT TISSUE RIGHT ANKLE
## 54440                   KNIFE SLIPPED AND METAL LACERATION HEAD
## 79781    SLIPPED DOWN ROCK SOFT TISSUE INJURY RIGHT INDEX KNEE
## 87997                       UNLOADING GLASS STRAINED LOWER BACK
## 67886                           PUSHING TOP LEFT HIP AND HERNIA
##       InitialCaseEstimate   Claim    AccDate AccDay AccYear AccMonth AccWeekday
## 96757               20000 3139046 2001-09-10   5005    2001        9          1
```

```
## 43824               7500 2895985 1994-07-18   2394    1994        7          1
## 54440                425 2049604 1990-10-16   1023    1990       10          2
## 79781              70000 2023994 2003-10-07   5762    2003       10          2
## 87997              25500 1669706 2001-08-27   4991    2001        8          1
## 67886             158500 1610744 2004-12-07   6189    2004       12          2
##       AccTime    RepDate RepDay RepYear RepMonth RepWeekday RepDelay Id
## 96757      12 2001-10-08   5033    2001       10          1       28  1
## 43824       7 1994-08-19   2426    1994        8          5       32  2
## 54440      13 1991-03-07   1165    1991        3          4      100  3
## 79781      12 2003-10-23   5778    2003       10          4       16  4
## 87997      13 2001-09-17   5012    2001        9          1       21  5
## 67886      10 2005-01-06   6219    2005        1          4       30  6
##             AgeNN   GenderNN DependentChildrenNN DependentsOtherNN WeeklyPayNN
## 96757  1.51961384 -1.8133890          -0.2525208       -0.08278064  1.09728748
## 43824  0.21894590 -1.8133890          -0.2525208       -0.08278064  0.47167188
## 54440 -1.32559727  0.5514475          -0.2525208       -0.08278064 -0.73306086
## 79781 -0.02492934  0.5514475          -0.2525208       -0.08278064  0.50971607
## 87997 -0.67526330  0.5514475          -0.2525208       -0.08278064  0.04895864
## 67886 -0.67526330  0.5514475          -0.2525208       -0.08278064  0.45053622
##       PartTimeFullTimeNN HoursWorkedPerWeekNN DaysWorkedPerWeekNN  AccYearNN
## 96757         -0.3136904           0.39373762           0.1612509  0.5880789
## 43824         -0.3136904           0.08730534           0.1612509 -0.7277872
## 54440         -0.3136904           0.08730534           0.1612509 -1.4797107
## 79781         -0.3136904           0.08730534           0.1612509  0.9640406
## 87997         -0.3136904          -0.06591079           0.1612509  0.5880789
## 67886         -0.3136904          -2.67058510          -1.7553096  1.1520215
##       AccMonthNN AccWeekdayNN  AccTimeNN   RepDelayNN Marital1 Marital2
## 96757  0.7709056   -1.3844942  0.1196021 -0.149961094        1        0
## 43824  0.1801416   -1.3844942 -1.1920949  0.001846787        0        0
## 54440  1.0662876   -0.7940759  0.3819415  2.582580779        0        1
## 79781  1.0662876   -0.7940759  0.1196021 -0.605384740        0        1
## 87997  0.4755236   -1.3844942  0.3819415 -0.415624888        0        0
## 67886  1.6570516   -0.7940759 -0.4050767 -0.074057153        0        0
##       Marital3      RandNN      RandUN
## 96757        0 -0.50295965 -0.08155268
## 43824        1  0.13090483 -0.90017020
## 54440        0 -0.07959023 -0.20039753
## 79781        0  0.88632647  1.66112994
## 87997        1  0.11634170 -0.04664537
## 67886        1  0.31804537 -0.08624419
```

```
str(dat)
```

```
## 'data.frame':    89332 obs. of  44 variables:
##  $ ClaimNumber         : chr  "WC9730090" "WC5305468" "WC6197732" "WC8318363" ...
##  $ Age                 : num  53 37 18 34 26 26 28 63 49 29 ...
##  $ Gender              : Factor w/ 2 levels "F","M": 1 1 2 2 2 2 2 2 2 1 ...
##  $ MaritalStatus       : Factor w/ 3 levels "M","S","U": 1 3 2 2 3 3 2 2 1 2 ...
##  $ DependentChildren   : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ DependentsOther     : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ WeeklyPay           : num  683 535 250 544 435 ...
##  $ PartTimeFullTime    : Factor w/ 2 levels "F","P": 1 1 1 1 1 1 1 1 1 1 ...
##  $ HoursWorkedPerWeek  : num  40 38 38 38 37 20 40 38 38 60 ...
##  $ DaysWorkedPerWeek   : num  5 5 5 5 5 4 5 5 5 5 ...
##  $ ClaimDescription    : chr  "BUFFING OUT DISC CUTTING RIGHT UPPER SWOLLEN HEAD" "FELL HEAD ON STEP
```

```
##  $ InitialCaseEstimate : num  20000 7500 425 70000 25500 ...
##  $ Claim               : int  3139046 2895985 2049604 2023994 1669706 1610744 1452597 1423497 133946
##  $ AccDate             : Date, format: "2001-09-10" "1994-07-18" ...
##  $ AccDay              : int  5005 2394 1023 5762 4991 6189 6318 5565 5766 3529 ...
##  $ AccYear             : int  2001 1994 1990 2003 2001 2004 2005 2003 2003 1997 ...
##  $ AccMonth            : int  9 7 10 10 8 12 4 3 10 8 ...
##  $ AccWeekday          : num  1 1 2 2 1 2 5 1 6 2 ...
##  $ AccTime             : int  12 7 13 12 13 10 17 11 16 12 ...
##  $ RepDate             : Date, format: "2001-10-08" "1994-08-19" ...
##  $ RepDay              : int  5033 2426 1165 5778 5012 6219 6330 5575 5782 3539 ...
##  $ RepYear             : int  2001 1994 1991 2003 2001 2005 2005 2003 2003 1997 ...
##  $ RepMonth            : int  10 8 3 10 9 1 4 4 10 9 ...
##  $ RepWeekday          : num  1 5 4 4 1 4 3 4 1 5 ...
##  $ RepDelay            : num  28 32 100 16 21 30 12 10 16 10 ...
##  $ Id                  : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ AgeNN               : num [1:89332, 1] 1.5196 0.2189 -1.3256 -0.0249 -0.6753 ...
##  $ GenderNN            : num [1:89332, 1] -1.813 -1.813 0.551 0.551 0.551 ...
##  $ DependentChildrenNN : num [1:89332, 1] -0.253 -0.253 -0.253 -0.253 -0.253 ...
##  $ DependentsOtherNN   : num [1:89332, 1] -0.0828 -0.0828 -0.0828 -0.0828 -0.0828 ...
##  $ WeeklyPayNN         : num [1:89332, 1] 1.097 0.472 -0.733 0.51 0.049 ...
##  $ PartTimeFullTimeNN  : num [1:89332, 1] -0.314 -0.314 -0.314 -0.314 -0.314 ...
##  $ HoursWorkedPerWeekNN: num [1:89332, 1] 0.3937 0.0873 0.0873 0.0873 -0.0659 ...
##  $ DaysWorkedPerWeekNN : num [1:89332, 1] 0.161 0.161 0.161 0.161 0.161 ...
##  $ AccYearNN           : num [1:89332, 1] 0.588 -0.728 -1.48 0.964 0.588 ...
##  $ AccMonthNN          : num [1:89332, 1] 0.771 0.18 1.066 1.066 0.476 ...
##  $ AccWeekdayNN        : num [1:89332, 1] -1.384 -1.384 -0.794 -0.794 -1.384 ...
##  $ AccTimeNN           : num [1:89332, 1] 0.12 -1.192 0.382 0.12 0.382 ...
##  $ RepDelayNN          : num [1:89332, 1] -0.14996 0.00185 2.58258 -0.60538 -0.41562 ...
##  $ Marital1            : int  1 0 0 0 0 0 0 0 1 0 ...
##  $ Marital2            : int  0 0 1 1 0 0 1 1 0 1 ...
##  $ Marital3            : int  0 1 0 0 1 1 0 0 0 0 ...
##  $ RandNN              : num [1:89332, 1] -0.503 0.1309 -0.0796 0.8863 0.1163 ...
##  $ RandUN              : num [1:89332, 1] -0.0816 -0.9002 -0.2004 1.6611 -0.0466 ...
```

```
summary(dat)
```

```
##  ClaimNumber             Age        Gender    MaritalStatus DependentChildren
##  Length:89332       Min.   :16.00   F:20831   M:35702       Min.   :0.00000
##  Class :character   1st Qu.:24.00   M:68501   S:43988       1st Qu.:0.00000
##  Mode  :character   Median :32.00             U: 9642       Median :0.00000
##                     Mean   :34.31                           Mean   :0.05994
##                     3rd Qu.:43.00                           3rd Qu.:0.00000
##                     Max.   :70.00                           Max.   :1.00000
##  DependentsOther      WeeklyPay      PartTimeFullTime HoursWorkedPerWeek
##  Min.   :0.000000   Min.   :   0.0   F:81329          Min.   : 1.00
##  1st Qu.:0.000000   1st Qu.: 200.0   P: 8003          1st Qu.:38.00
##  Median :0.000000   Median : 415.0                    Median :38.00
##  Mean   :0.006806   Mean   : 423.4                    Mean   :37.43
##  3rd Qu.:0.000000   3rd Qu.: 533.0                    3rd Qu.:40.00
##  Max.   :1.000000   Max.   :1200.0                    Max.   :60.00
##  DaysWorkedPerWeek ClaimDescription   InitialCaseEstimate     Claim
##  Min.   :1.000     Length:89332       Min.   :     1      Min.   :     20
##  1st Qu.:5.000     Class :character   1st Qu.:   730      1st Qu.:    237
##  Median :5.000     Mode  :character   Median :  2000      Median :    662
##  Mean   :4.916                        Mean   :  8986      Mean   :  12730
```

```
##  3rd Qu.:5.000                        3rd Qu.: 10000      3rd Qu.:   2766
##  Max.   :7.000                        Max.   :560000      Max.   :3139046
##     AccDate               AccDay        AccYear        AccMonth
##  Min.   :1988-01-01   Min.   :   4   Min.   :1988   Min.   : 1.00
##  1st Qu.:1994-01-24   1st Qu.:2219   1st Qu.:1994   1st Qu.: 3.00
##  Median :1998-11-07   Median :3968   Median :1998   Median : 6.00
##  Mean   :1998-05-11   Mean   :3788   Mean   :1998   Mean   : 6.39
##  3rd Qu.:2003-02-10   3rd Qu.:5523   3rd Qu.:2003   3rd Qu.: 9.00
##  Max.   :2006-07-20   Max.   :6779   Max.   :2006   Max.   :12.00
##    AccWeekday       AccTime         RepDate               RepDay
##  Min.   :1.000   Min.   : 0.00   Min.   :1988-01-07   Min.   :  10
##  1st Qu.:2.000   1st Qu.: 9.00   1st Qu.:1994-03-01   1st Qu.:2255
##  Median :3.000   Median :11.00   Median :1998-12-17   Median :4007
##  Mean   :3.345   Mean   :11.54   Mean   :1998-06-19   Mean   :3826
##  3rd Qu.:5.000   3rd Qu.:14.00   3rd Qu.:2003-03-17   3rd Qu.:5558
##  Max.   :7.000   Max.   :23.00   Max.   :2008-06-19   Max.   :7479
##     RepYear        RepMonth        RepWeekday       RepDelay
##  Min.   :1988   Min.   : 1.000   Min.   :1.000   Min.   :  0.00
##  1st Qu.:1994   1st Qu.: 4.000   1st Qu.:1.000   1st Qu.: 14.00
##  Median :1998   Median : 7.000   Median :2.000   Median : 22.00
##  Mean   :1998   Mean   : 6.524   Mean   :2.443   Mean   : 31.95
##  3rd Qu.:2003   3rd Qu.: 9.000   3rd Qu.:4.000   3rd Qu.: 41.00
##  Max.   :2008   Max.   :12.000   Max.   :5.000   Max.   :100.00
##        Id              AgeNN.V1           GenderNN.V1
##  Min.   :    1   Min.   :-1.4881808   Min.   :-1.8133890
##  1st Qu.:22334   1st Qu.:-0.8378468   1st Qu.: 0.5514475
##  Median :44666   Median :-0.1875128   Median : 0.5514475
##  Mean   :44666   Mean   : 0.0000000   Mean   : 0.0000000
##  3rd Qu.:66999   3rd Qu.: 0.7066964   3rd Qu.: 0.5514475
##  Max.   :89332   Max.   : 2.9015735   Max.   : 0.5514475
##  DependentChildrenNN.V1 DependentsOtherNN.V1   WeeklyPayNN.V1
##  Min.   :-0.252521      Min.   :-0.082781    Min.   :-1.789844
##  1st Qu.:-0.252521      1st Qu.:-0.082781    1st Qu.:-0.944417
##  Median :-0.252521      Median :-0.082781    Median :-0.035584
##  Mean   : 0.000000      Mean   : 0.000000    Mean   : 0.000000
##  3rd Qu.:-0.252521      3rd Qu.:-0.082781    3rd Qu.: 0.463218
##  Max.   : 3.960026      Max.   :12.079983    Max.   : 3.282715
##  PartTimeFullTimeNN.V1 HoursWorkedPerWeekNN.V1 DaysWorkedPerWeekNN.V1
##  Min.   :-0.313690     Min.   :-5.581692       Min.   :-7.504991
##  1st Qu.:-0.313690     1st Qu.: 0.087305       1st Qu.: 0.161251
##  Median :-0.313690     Median : 0.087305       Median : 0.161251
##  Mean   : 0.000000     Mean   : 0.000000       Mean   : 0.000000
##  3rd Qu.:-0.313690     3rd Qu.: 0.393738       3rd Qu.: 0.161251
##  Max.   : 3.187821     Max.   : 3.458060       Max.   : 3.994372
##      AccYearNN.V1        AccMonthNN.V1       AccWeekdayNN.V1
##  Min.   :-1.8556724   Min.   :-1.5921504   Min.   :-1.3844942
##  1st Qu.:-0.7277872   1st Qu.:-1.0013864   1st Qu.:-0.7940759
##  Median : 0.0241363   Median :-0.1152404   Median :-0.2036577
##  Mean   : 0.0000000   Mean   : 0.0000000   Mean   : 0.0000000
##  3rd Qu.: 0.9640406   3rd Qu.: 0.7709056   3rd Qu.: 0.9771788
##  Max.   : 1.5279832   Max.   : 1.6570516   Max.   : 2.1580153
##      AccTimeNN.V1        RepDelayNN.V1        Marital1         Marital2
##  Min.   :-3.0284708   Min.   :-1.2126163   Min.   :0.0000   Min.   :0.0000
##  1st Qu.:-0.6674161   1st Qu.:-0.6812887   1st Qu.:0.0000   1st Qu.:0.0000
```

```
##  Median :-0.1427373   Median :-0.3776729   Median :0.0000   Median :0.0000
##  Mean   : 0.0000000   Mean   : 0.0000000   Mean   :0.3997   Mean   :0.4924
##  3rd Qu.: 0.6442809   3rd Qu.: 0.3434145   3rd Qu.:1.0000   3rd Qu.:1.0000
##  Max.   : 3.0053356   Max.   : 2.5825808   Max.   :1.0000   Max.   :1.0000
##     Marital3           RandNN.V1           RandUN.V1
##  Min.   :0.0000   Min.   :-4.212677   Min.   :-1.7384595
##  1st Qu.:0.0000   1st Qu.:-0.671017   1st Qu.:-0.8693163
##  Median :0.0000   Median :-0.004631   Median : 0.0000425
##  Mean   :0.1079   Mean   : 0.000000   Mean   : 0.0000000
##  3rd Qu.:0.0000   3rd Qu.: 0.672313   3rd Qu.: 0.8656251
##  Max.   :1.0000   Max.   : 4.363558   Max.   : 1.7326854
```

Split train and test data

To perform a proper out-of-sample generalization analysis we partition our data into a learning data set $\mathcal{L}$ and a test data set $\mathcal{T}$. We hold on to the same partitioning throughout this notebook to have comparability across all methods studied. The learning data set $\mathcal{L}$ will be used for model fitting (this includes early stopping rules in gradient descent fitting), and the test data set $\mathcal{T}$ will only be used for out-of-sample predictive performance analyses.

The empirical analysis of our data has shown that the claim averages are non-stationary over time and that claim sizes follow a highly skewed distribution. To have learning and test data sets sharing similar properties w.r.t. claim sizes we partition the entire data in a stratifieed way into learning data set $\mathcal{L}$ and test data set $\mathcal{T}$ in a ratio $4:1$. That is, we order the claims w.r.t. their claim sizes. Then, we select at random 1 claim from the 5 biggest ones for $\mathcal{T}$, 1 claim from the 5 second biggest ones for $\mathcal{T}$, and so on, and the non-selected claims are allocated to $\mathcal{L}$. This random allocation provides us with a learning data set $\mathcal{L}$ and a test data set $\mathcal{T}$.

$\Rightarrow$ The general assumption is that all data (Y_i, x_i) are independent and identical distributed (i.i.d.), and we are going to determine the conditional distribution (in-sample) of Y_i, given x_i, which can then be used to forecast (out-of-sample) Y_t, given x_t.

```
idx <- sample(x = c(1:5), size = ceiling(nrow(dat) / 5), replace = TRUE)
idx <- (1:ceiling(nrow(dat) / 5) - 1) * 5 + idx

test <- dat[intersect(idx, 1:nrow(dat)), ]
learn <- dat[setdiff(1:nrow(dat), idx), ]

learn <- learn[sample(1:nrow(learn)), ]
test <- test[sample(1:nrow(test)), ]
```

```
# size of train/test
sprintf("Number of observations (learn): %s", nrow(learn))
```

```
## [1] "Number of observations (learn): 71466"
```

```
sprintf("Number of observations (test): %s", nrow(test))
```

```
## [1] "Number of observations (test): 17866"
```

```
# Claims average of learn/test
sprintf("Empirical claims average (learn): %s", round(sum(learn$Claim) / length(learn$Claim), 0))
```

```
## [1] "Empirical claims average (learn): 12733"
```

```
sprintf("Empirical claims average (test): %s", round(sum(test$Claim) / length(test$Claim), 0))
```

```
## [1] "Empirical claims average (test): 12714"
```

```
# Quantiles of learn/test
probs <- c(.1, .25, .5, .75, .9)
bind_rows(quantile(learn$Claim, probs = probs), quantile(test$Claim, probs = probs))
```

```
## # A tibble: 2 x 5
##   `10%` `25%` `50%` `75%`  `90%`
##   <dbl> <dbl> <dbl> <dbl>  <dbl>
## 1   110  237    662 2766  18370.
## 2   110  237.   662 2766. 18368
```

Exercise: Use the standard 80/20 split for the learning and test set and compare the result. Can you follow the reasons to split differently?

Store model results

As we are going to compare various models, we create a table which stores the metrics we are going to use for the comparison and the selection of the best model.

```
# initialize table to store all model results for comparison
df_cmp <- tibble(
 model = character(),
 learn_p2 = numeric(),
 learn_pp = numeric(),
 learn_p3 = numeric(),
 test_p2 = numeric(),
 test_pp = numeric(),
 test_p3 = numeric(),
 avg_size = numeric(),
)
```

In the following chapters, we are going to fit various models to the data. At the end, we will compare the performance and quality of the several fitted models. We compare them by using the metrics defined above.

Descriptive analysis

Reporting Delay summary statistics

This provides us claims all having occurred between 1988/01/01 and 2006/07/20.

```
range(dat$AccYear)
```

```
## [1] 1988 2006
```

These claims have been reported in the calendar years from 1998 to 2008.

```
range(dat$RepYear)
```

```
## [1] 1988 2008
```

We have limited the maximum reporting delay to 100 (see above).

```
range(dat$RepDelay)
```

```
## [1]   0 100
```

```
round(range(dat$RepDelay) / 365, 4)
```

```
## [1] 0.000 0.274
```

```
round(range(dat$RepDelay) / 7, 4)
```

```
## [1]  0.0000 14.2857
```

```
round(mean(dat$RepDelay) / 7, 4)
```

```
## [1] 4.5645
```

```
# define acc_week, rep_week and RepDelay_week
min_accDay <- min(dat$AccDay)
dat <- dat %>% mutate(
    acc_week = floor((AccDay - min_accDay) / 7),
    rep_week = floor((RepDay - min_accDay) / 7),
    RepDelay_week = rep_week - acc_week
)
```

```
quantile(dat$RepDelay_week, probs = c(.9, .98, .99))
```

```
## 90% 98% 99%
##  11  29  39
```

```
acc1 <- dat %>% group_by(acc_week, rep_week) %>% summarize(nr = n())
```

```
## `summarise()` has grouped output by 'acc_week'. You can override using the `.groups` argument.
```

```
acc2 <- dat %>% group_by(acc_week) %>% summarize(mm = mean(RepDelay_week))
```

```
head(acc1)
```

```
## # A tibble: 6 x 3
## # Groups:   acc_week [1]
##   acc_week rep_week    nr
##      <dbl>    <dbl> <int>
## 1        0        0     1
## 2        0        1     7
## 3        0        2     7
## 4        0        3     7
## 5        0        4     5
## 6        0        5     3
```

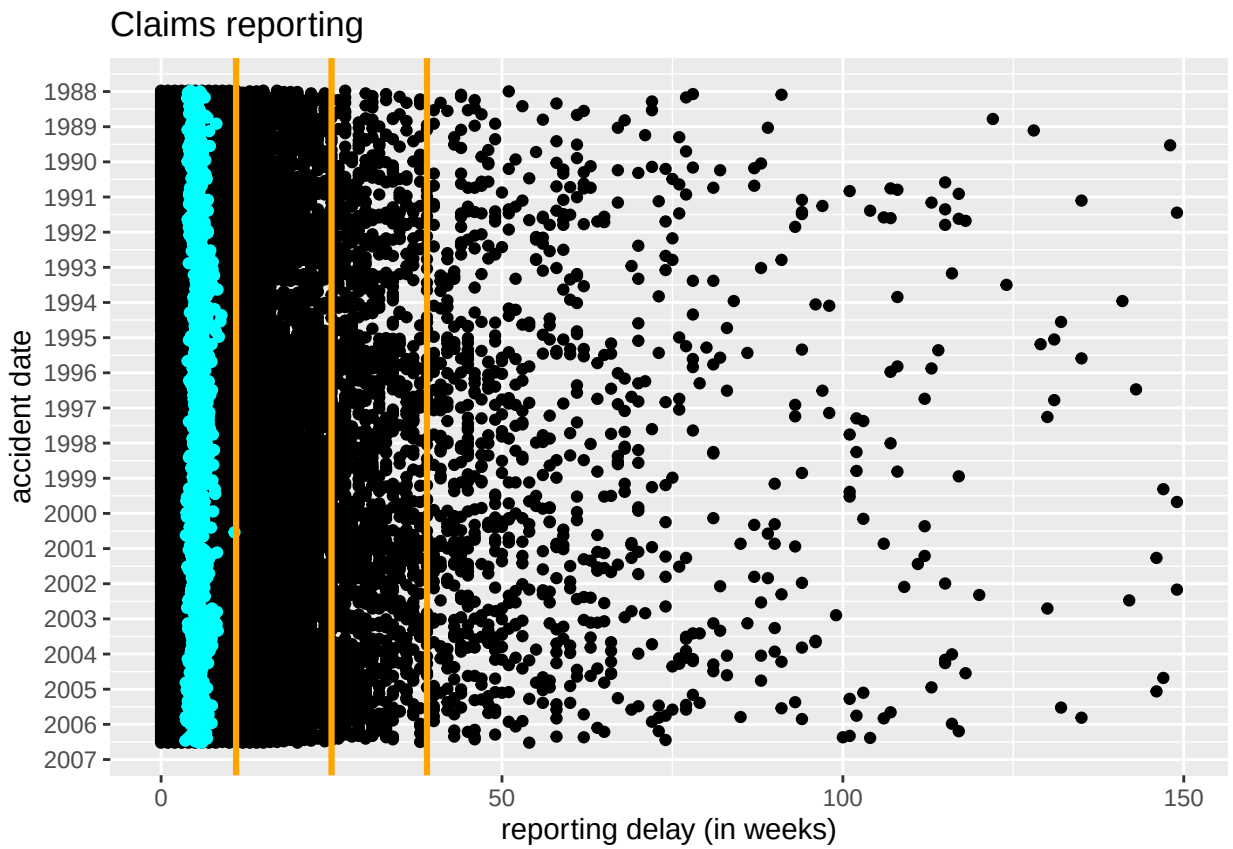
```
head(acc2)
```

```
## # A tibble: 6 x 2
##   acc_week    mm
##      <dbl> <dbl>
## 1        0  6
## 2        1  5.35
## 3        2  5
## 4        3  3.56
## 5        4  4.86
## 6        5  5.84
```

```
# to plot the quantiles
qq0 <- c(.9, .975, .99)
qq1 <- quantile(dat$RepDelay_week, probs = qq0)
qq1
```

```
##   90% 97.5%   99%
##    11    25    39
```

```
ggplot(acc1, aes(x = rep_week - acc_week, y = max(acc_week) - acc_week)) +
    geom_point() +
    geom_point(data = acc2, aes(x = mm, y = acc_week), color = "cyan") +
    geom_vline(xintercept = qq1, color = "orange", size = line_size) +
    scale_y_continuous(
      labels = rev(c(min(dat$AccYear):(max(dat$AccYear) + 1))),
      breaks = c(0:(max(dat$AccYear) - min(dat$AccYear) + 1)) * 365 / 7 - 25
    ) + labs(title = "Claims reporting", x = "reporting delay (in weeks)", y = "accident date")
```



The figure above shows the accident dates vs. the reporting delays of these claims (in weekly units), the cyan dots give the weekly averages; The orange vertical lines show the 90%, 97.5% and 99% quantiles at 11, 25 and 39 weeks. From this plot we cannot detect any trends in reporting delays, i.e., it suggests a stationary reporting behavior over time.

Claim size summary statistics

The claim amounts of these accident insurance claims range from 20 to 3'136'046.

```
range(dat$Claim)
```

```
## [1]      20 3139046
```

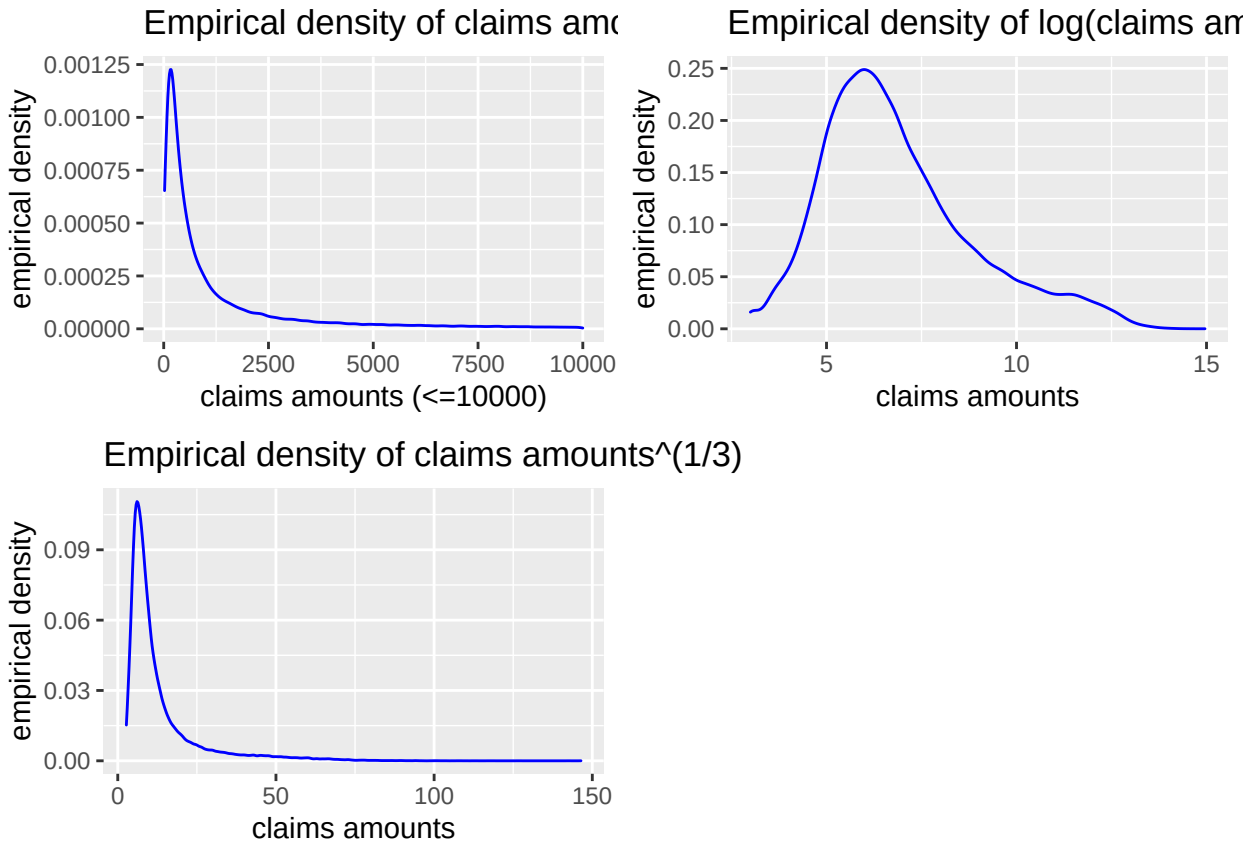
And the key statistics from the claim amount distribution are as follows.

```
summary(dat$Claim)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##      20     237     662   12730    2766 3139046
```

The figures below show empirical plots of these claim amounts, with two transformations applied to the claim amounts for better visualizations. The plots show that the empirical density is unimodal.

```
p1 <- ggplot(dat %>% filter(Claim <= 10000), aes(x = Claim)) + geom_density(colour = "blue") +
    labs(title = "Empirical density of claims amounts", x = "claims amounts (<=10000)", y = "empirical d

p2 <- ggplot(dat, aes(x = log(Claim))) + geom_density(colour = "blue") +
    labs(title = "Empirical density of log(claims amounts)", x = "claims amounts", y = "empirical densi

p3 <- ggplot(dat, aes(x = Claim^(1/3))) + geom_density(colour = "blue") +
    labs(title = "Empirical density of claims amounts^(1/3)", x = "claims amounts", y = "empirical dens

grid.arrange(p1, p2, p3, ncol = 2)
```



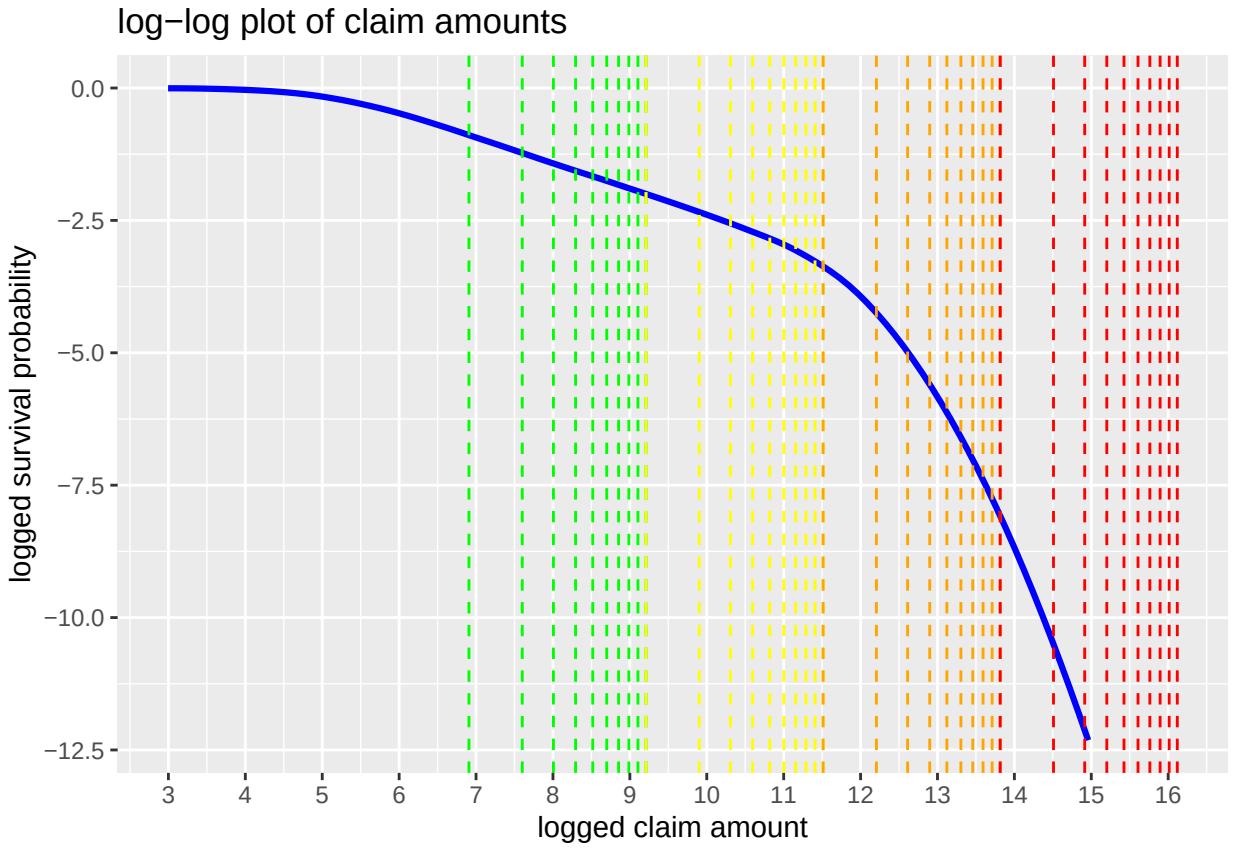
Log-log plot

Below we show the log-log plot indicating heavy-tailedness, i.e., resembling asymptotically a straight line with negative slope.

```
pp <- ecdf(dat$Claim)
xx <- min(log(dat$Claim)) + 0:100/100 * (max(log(dat$Claim)) - min(log(dat$Claim)))
ES <- predict(locfit(log(1.00001 - pp(dat$Claim)) ~ log(dat$Claim), alpha = 0.1, deg = 2), newdata = xx)
dat_loglog <- data.frame(xx = xx, ES = ES)
```

```
ggplot(dat_loglog, aes(x = xx, y = ES)) + geom_line(colour = "blue", size = line_size) +
    geom_vline(xintercept = log(c(1:10) * 1000), colour = "green", linetype = "dashed") +
    geom_vline(xintercept = log(c(1:10) * 10000), colour = "yellow", linetype = "dashed") +
    geom_vline(xintercept = log(c(1:10) * 100000), colour = "orange", linetype = "dashed") +
```

```
    geom_vline(xintercept = log(c(1:10) * 1000000), colour = "red", linetype = "dashed") +
    labs(title = "log-log plot of claim amounts", x = "logged claim amount", y = "logged survival probal
    scale_x_continuous(breaks = seq(3,16,1))
```



Marginal plots

The claims are supported by various covariates. We will not further discuss the meaning of these covariates as their names are rather self-explanatory. ClaimsDescription contains a short claim description, and InitialClaimsEstimate the initial case estimate.

Moreover, the accident date, accident daytime and the reporting date, allow us to calculate time related variables such as the reporting delay, but also the weekday of the accident, the accident month and the accident year. We are going to use this information as covariates for claim prediction because claims maybe influenced by the daytime, the weekday, the season expressed by the calendar months, and the accident year in case of non-stationarity.

```
col_names <- c("Age","Gender","MaritalStatus","DependentChildren","DependentsOther",
               "WeeklyPay","PartTimeFullTime","HoursWorkedPerWeek","DaysWorkedPerWeek",
               "AccYear","AccMonth","AccWeekday","AccTime","RepDelay")

global_avg <- mean(dat$Claim)
severity_limits <- c(0,40000)
```

```
dat_tmp <- dat
dat_tmp <- dat_tmp %>% mutate(
    WeeklyPay = pmin(1200, ceiling(WeeklyPay / 100) * 100),
    HoursWorkedPerWeek = pmin(60, HoursWorkedPerWeek),
    RepDelay = pmin(100, floor(RepDelay / 10) * 10)
```

```
)
```

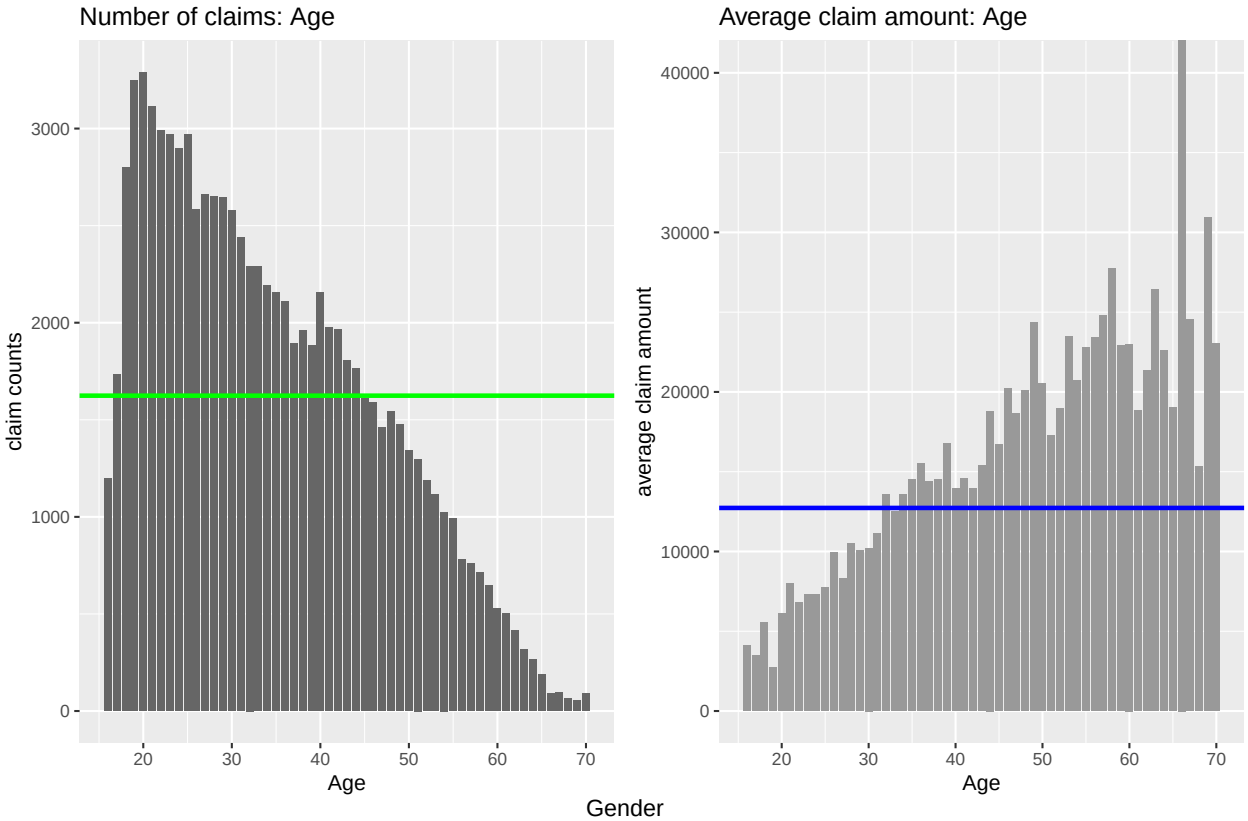
In the subsequent figures we show the number of claims and the average claim amounts per AccYear, AccMonth and AccWeekday; all plots of average claim amounts have the same y-scale `severity_limits` and the same blue overall average claim amount called `global_avg`. The horizontal green line is the average number of claims per class shown on the x-axis.

For the last year 2016 we only have data until July 20. We observe that claim averages are increasing over time. AccMonth is fairly uniform and we have less accidents on weekends, however, this does not significantly influence the claim averages. Accidents during night times are less frequent but more expensive, and reporting delay does not have a major impact on claim averages. We censor reporting delay at 100 days, because of scarcity of data beyond and because of right-skewness of reporting delays.

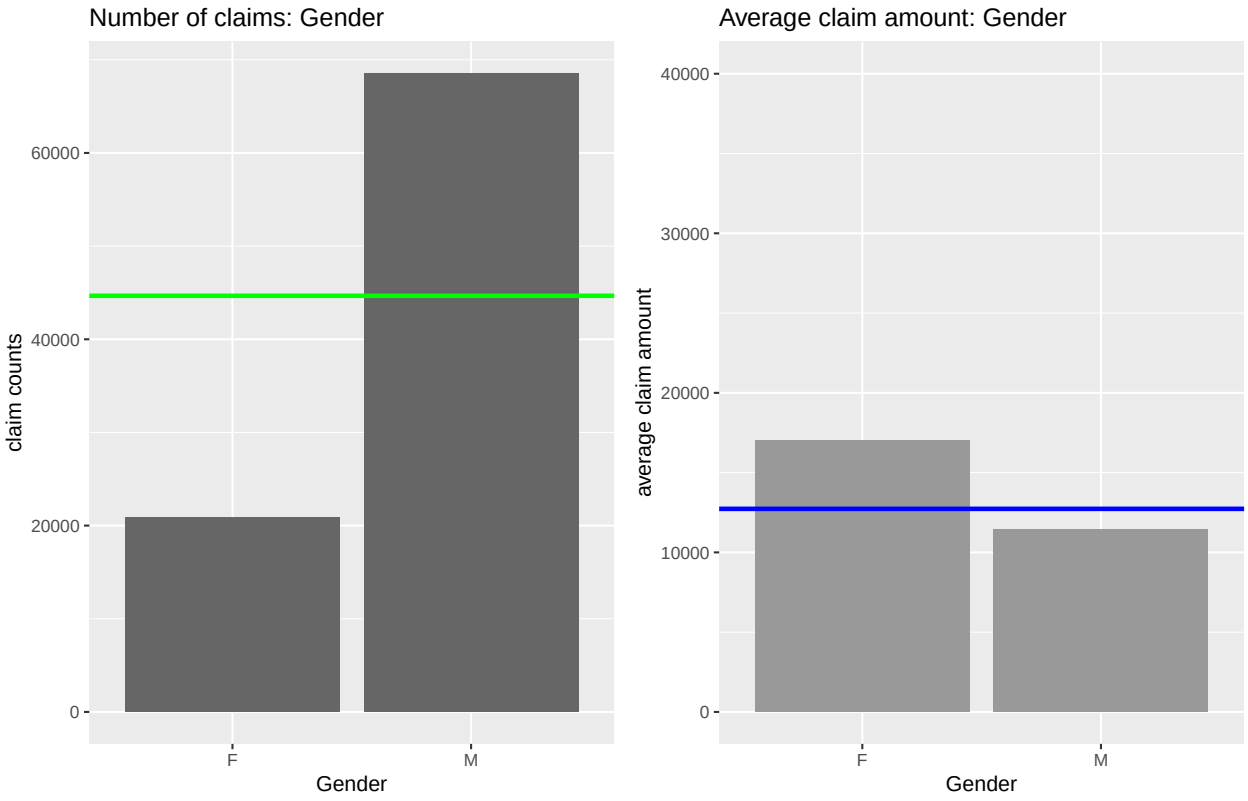
For regression modeling, we will censor Age at ages 16 and 70 because observations beyond these censoring points are scarce. We will censor DependentChildren and DependentsOther at 1, to differentiate 0 and 1, this provides us with binary variables. WeeklyPay is shown censored at 1'200. HoursWorkedPerWeek is censored at 60.

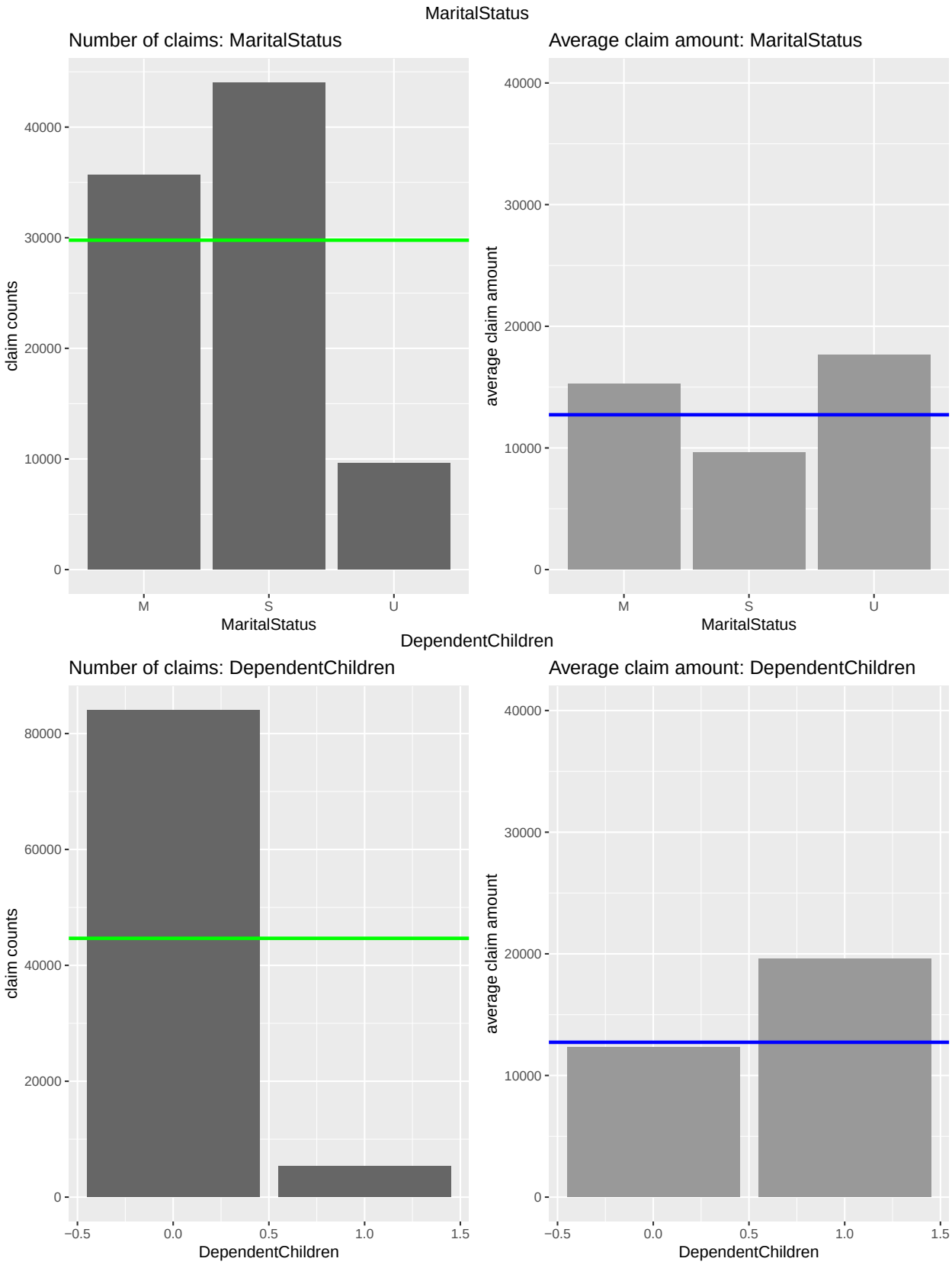
```
for (k in 1:length(col_names)) {
    xvar <- col_names[k]

    out <- dat_tmp %>% group_by(!!sym(xvar)) %>% summarize(vol = n(), avg = mean(Claim))

    tmp <- dat_tmp %>% select(!!sym(xvar))
    global_n <- nrow(dat_tmp) / length(levels(factor(tmp[[1]])))

    plt1 <- ggplot(out, aes(x = !!sym(xvar), group = 1)) + geom_bar(aes(weight = vol), fill = "gray40")
        geom_hline(yintercept = global_n, colour = "green", size = line_size) +
        labs(title = paste("Number of claims:", col_names[k]), x = col_names[k], y = "claim counts")

    plt2 <- ggplot(out, aes(x = !!sym(xvar), group = 1)) + geom_bar(aes(weight = avg), fill = "gray60")
        geom_hline(yintercept = global_avg, colour = "blue", size = line_size) +
        coord_cartesian(ylim = severity_limits) +
        labs(title = paste("Average claim amount:", col_names[k]), x = col_names[k], y = "average claim

    grid.arrange(plt1, plt2, ncol = 2, top = col_names[k])
}
```

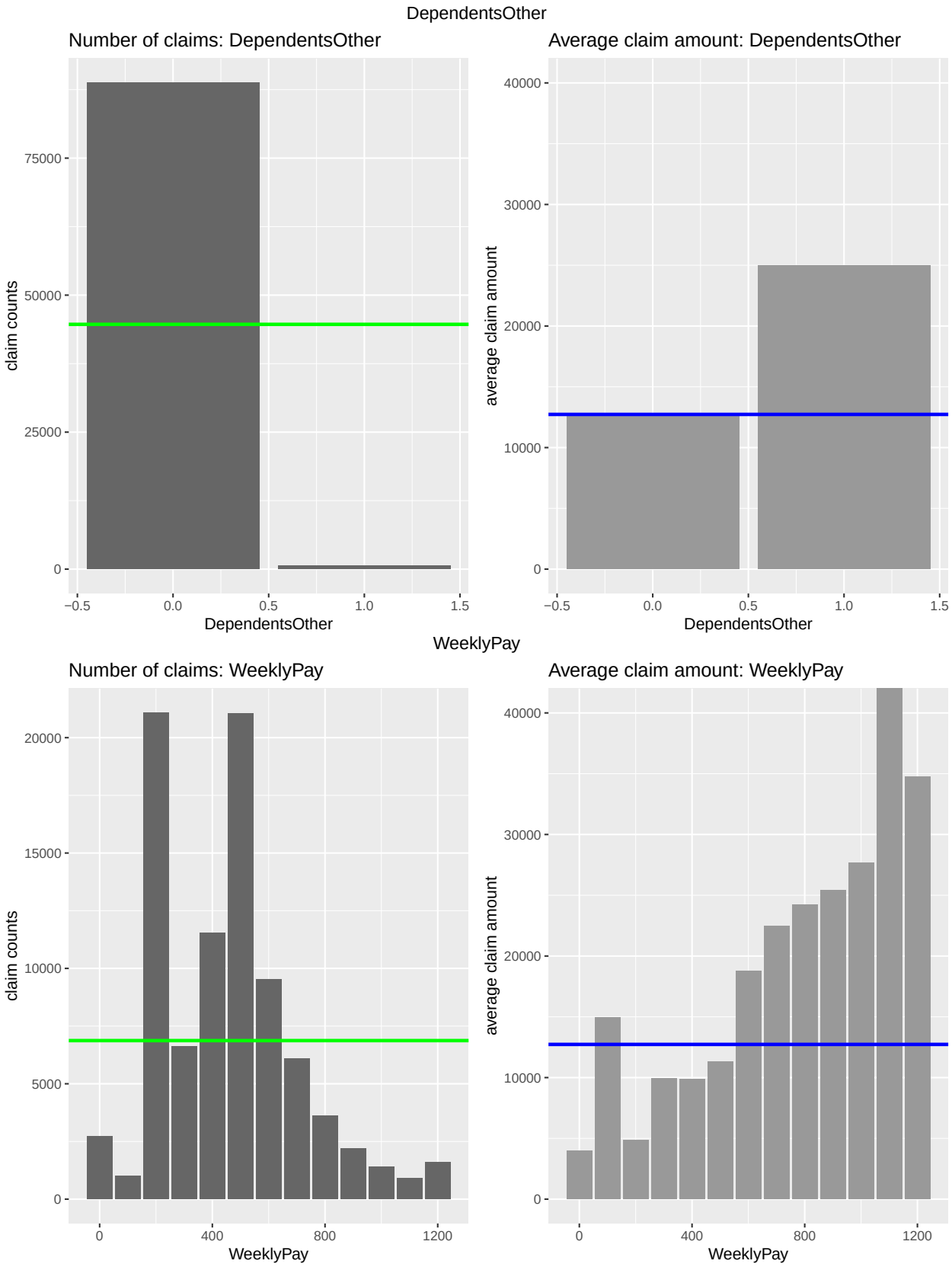
Age

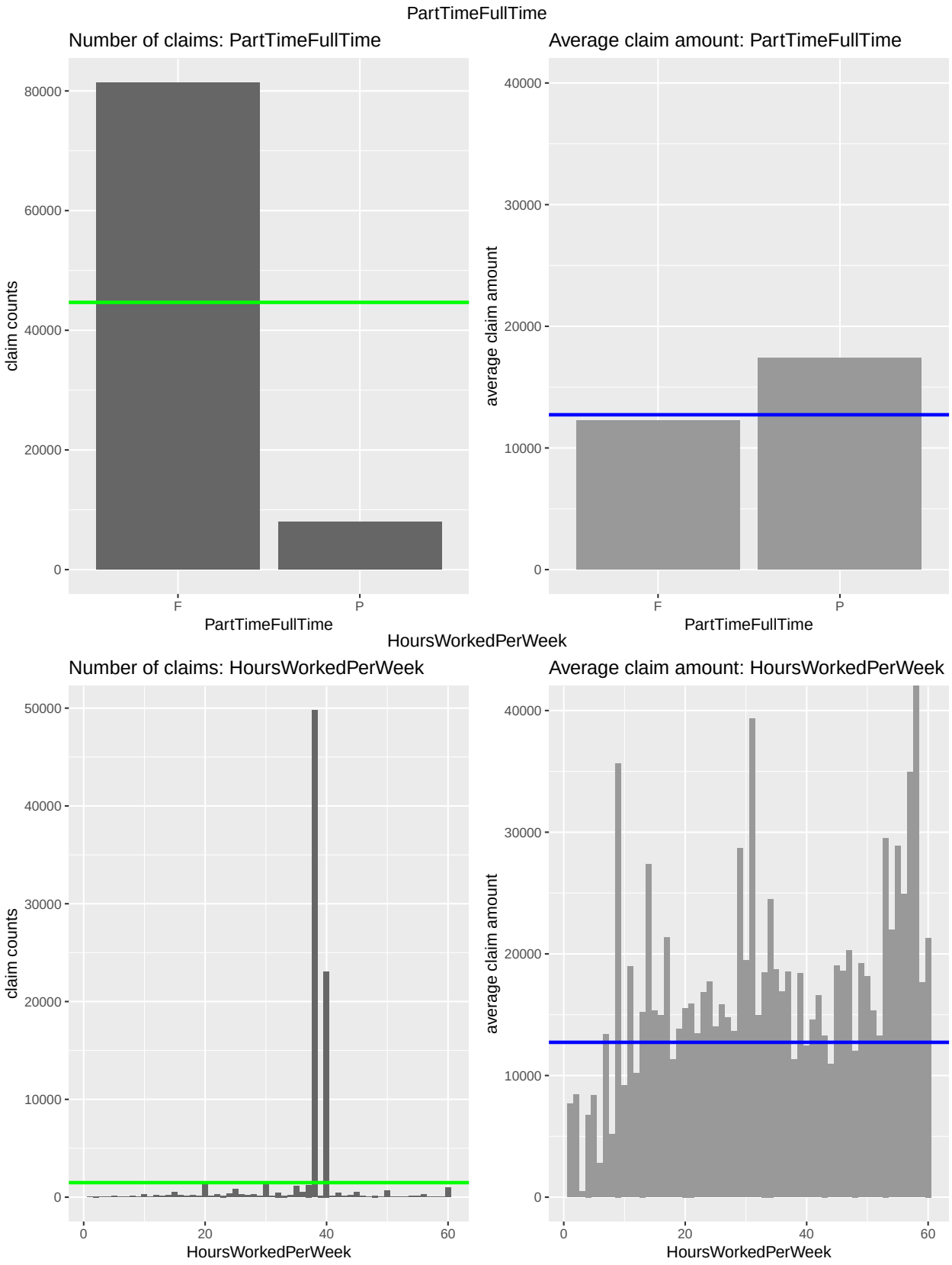

Number of claims: Age
claim counts
3000
2000
1000
0
20
30
40
50
60
70
Age
Average claim amount: Age
average claim amount
40000
30000
20000
10000
0
20
30
40
50
60
70
Age

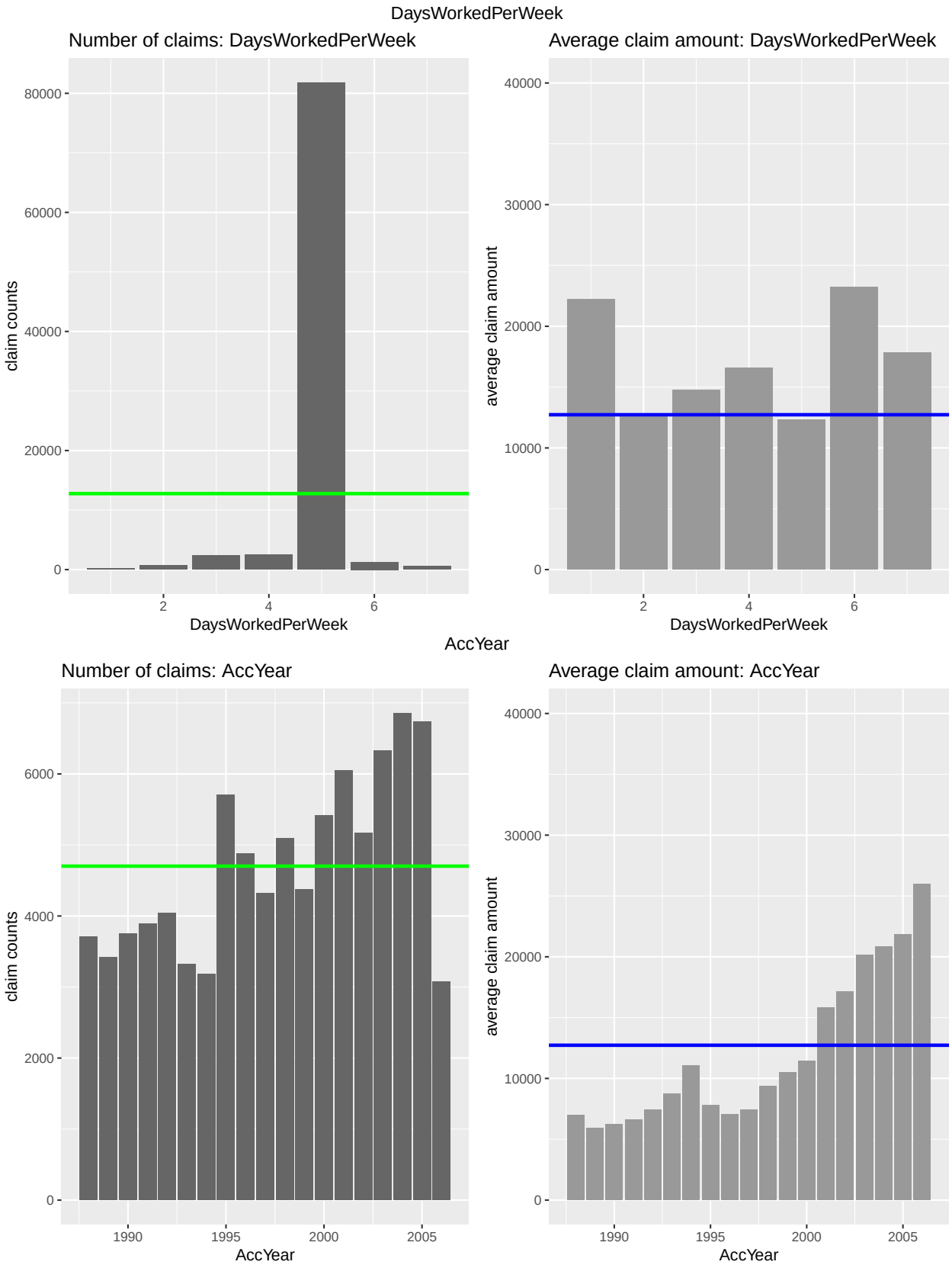
Gender

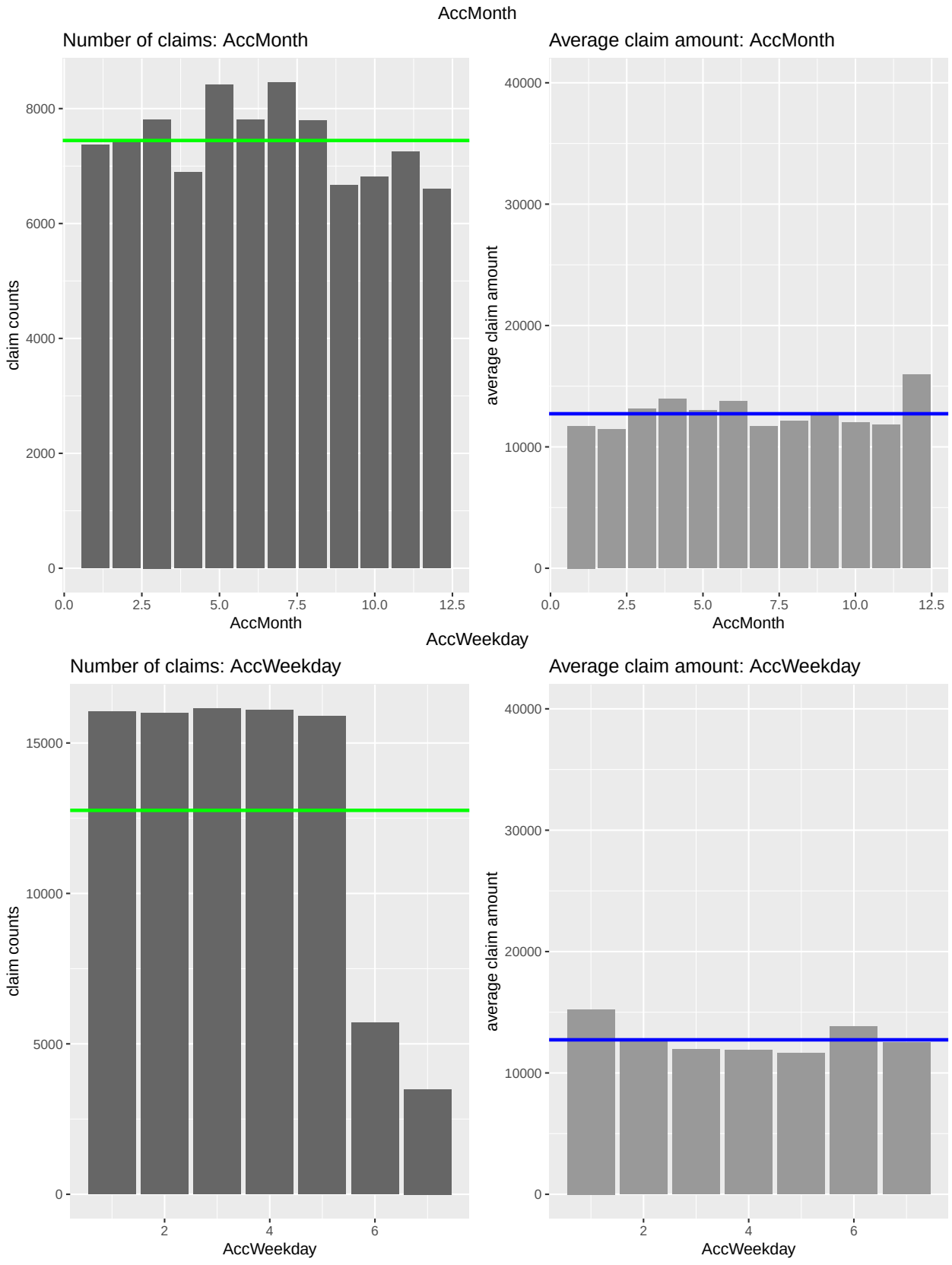

Number of claims: Gender
claim counts
60000
40000
20000
0
F
M
Gender
Average claim amount: Gender
average claim amount
40000
30000
20000
10000
0
F
M
Gender

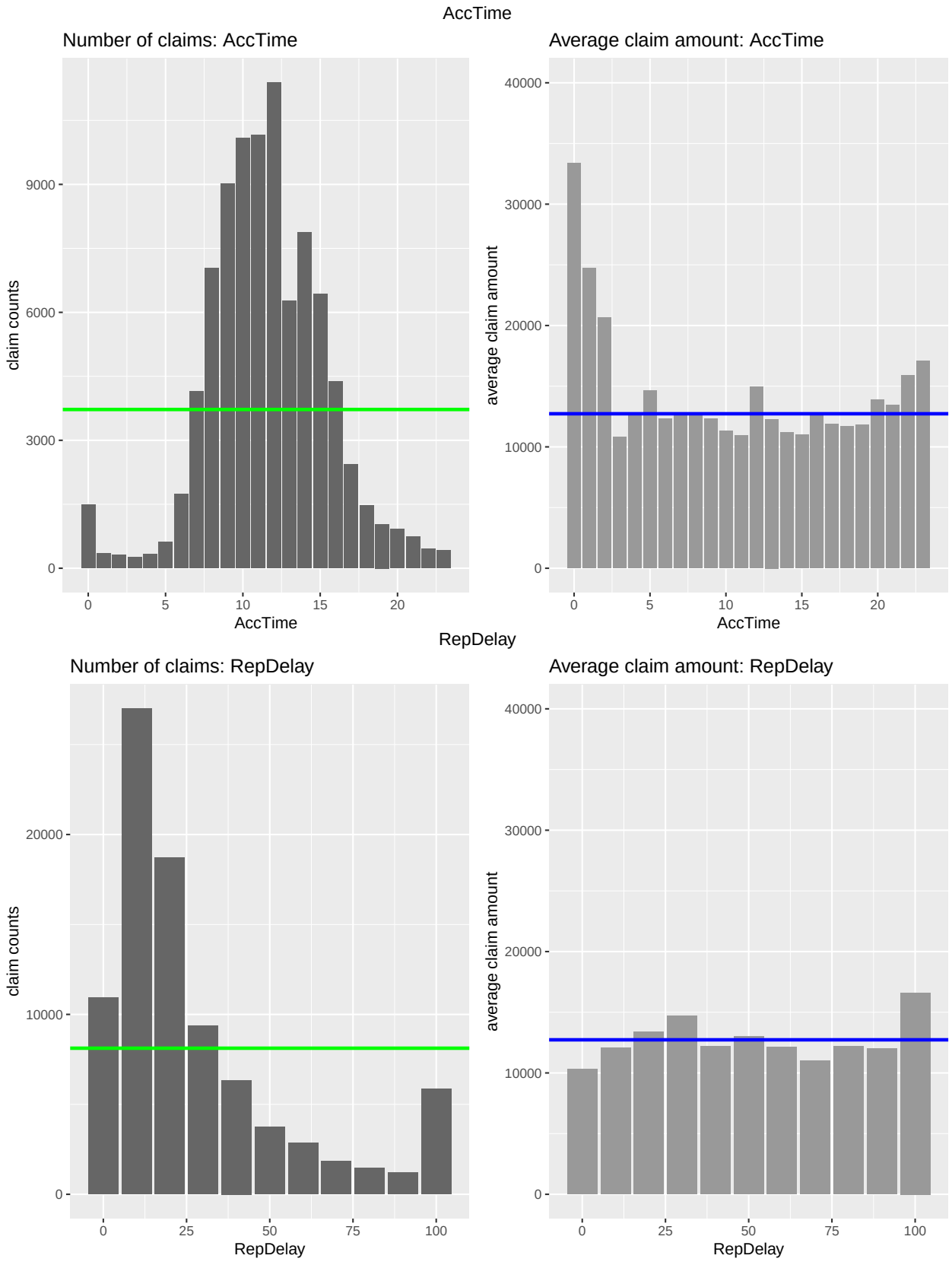

MaritalStatus
Number of claims: MaritalStatus
claim counts
40000
30000
20000
10000
0
M
S
U
MaritalStatus
Average claim amount: MaritalStatus
average claim amount
40000
30000
20000
10000
0
M
S
U
MaritalStatus
DependentChildren
Number of claims: DependentChildren
claim counts
80000
60000
40000
20000
0
−0.5
0.0
0.5
1.0
1.5
DependentChildren
Average claim amount: DependentChildren
average claim amount
40000
30000
20000
10000
0
−0.5
0.0
0.5
1.0
1.5
DependentChildren


DependentsOther
Number of claims: DependentsOther
claim counts
75000
50000
25000
0
−0.5
0.0
0.5
1.0
1.5
DependentsOther
Average claim amount: DependentsOther
average claim amount
40000
30000
20000
10000
0
−0.5
0.0
0.5
1.0
1.5
DependentsOther
WeeklyPay
Number of claims: WeeklyPay
claim counts
20000
15000
10000
5000
0
0
400
800
1200
WeeklyPay
Average claim amount: WeeklyPay
average claim amount
40000
30000
20000
10000
0
0
400
800
1200
WeeklyPay


PartTimeFullTime
Number of claims: PartTimeFullTime
claim counts
80000
60000
40000
20000
0
F
P
PartTimeFullTime
Average claim amount: PartTimeFullTime
average claim amount
40000
30000
20000
10000
0
F
P
PartTimeFullTime
HoursWorkedPerWeek
Number of claims: HoursWorkedPerWeek
claim counts
50000
40000
30000
20000
10000
0
0
20
40
60
HoursWorkedPerWeek
Average claim amount: HoursWorkedPerWeek
average claim amount
40000
30000
20000
10000
0
0
20
40
60
HoursWorkedPerWeek


DaysWorkedPerWeek
Number of claims: DaysWorkedPerWeek
Average claim amount: DaysWorkedPerWeek
claim counts
average claim amount
DaysWorkedPerWeek
DaysWorkedPerWeek
AccYear
Number of claims: AccYear
Average claim amount: AccYear
claim counts
average claim amount
AccYear
AccYear


AccMonth
Number of claims: AccMonth
claim counts
AccMonth
Average claim amount: AccMonth
average claim amount
AccMonth
AccWeekday
Number of claims: AccWeekday
claim counts
AccWeekday
Average claim amount: AccWeekday
average claim amount
AccWeekday



The levels of these covariates show some sensitivities in claim averages, and our goal is to fit a regression model to the individual claim sizes to understand systematic effects in the claim amounts and the significance

of these differences.

Feature correlation

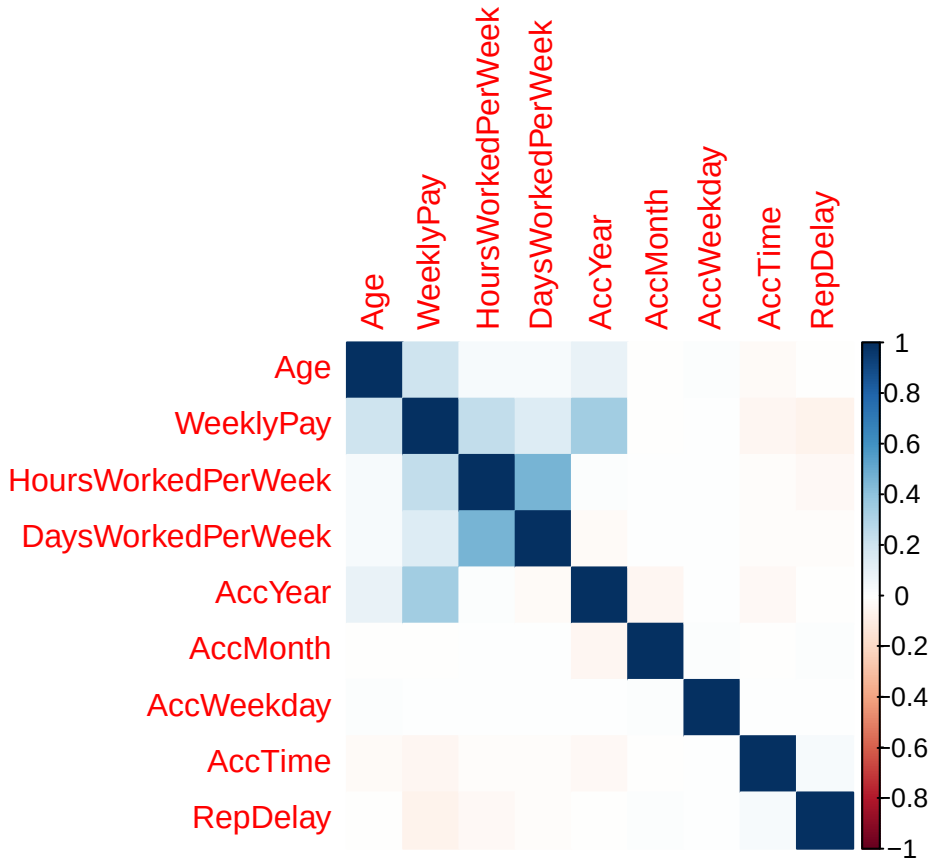
Analyzing dependence between covariate components, see table below, we find positive correlation between WeeklyPay, Age, DaysWorkedPerWeek and HoursWorkedPerWeek, which, of course, makes perfect sense from a practical point of view. We also find positive correlation between AccYear and HoursWorkedPerWeek. A further analysis of this positive correlation shows that it comes from the fact that in the early years from 1988 to 1994 roughly 1'500 claims per year have HoursWorkedPerWeek equal to 0, and after 1994 such zeros have almost completely disappeared. Probably such zeros in early years are missing data; to not distort our analysis, we have simply dropped these claims.

```
sel_col <- c("Age","WeeklyPay","HoursWorkedPerWeek","DaysWorkedPerWeek",
             "AccYear","AccMonth","AccWeekday","AccTime","RepDelay")
dat_tmp <- dat[, sel_col]
```

```
corrMat <- round(cor(dat_tmp, method = "pearson"), 2)
corrMat
```

```
##                       Age WeeklyPay HoursWorkedPerWeek DaysWorkedPerWeek AccYear
## Age                  1.00      0.20               0.03              0.03    0.09
## WeeklyPay            0.20      1.00               0.24              0.15    0.34
## HoursWorkedPerWeek   0.03      0.24               1.00              0.46    0.01
## DaysWorkedPerWeek    0.03      0.15               0.46              1.00   -0.03
## AccYear              0.09      0.34               0.01             -0.03    1.00
## AccMonth            -0.01     -0.01               0.00              0.00   -0.05
## AccWeekday           0.01      0.00               0.00              0.00    0.00
## AccTime             -0.03     -0.05              -0.02             -0.02   -0.04
## RepDelay            -0.01     -0.07              -0.04             -0.02   -0.01
##                     AccMonth AccWeekday AccTime RepDelay
## Age                    -0.01       0.01   -0.03    -0.01
## WeeklyPay              -0.01       0.00   -0.05    -0.07
## HoursWorkedPerWeek      0.00       0.00   -0.02    -0.04
## DaysWorkedPerWeek       0.00       0.00   -0.02    -0.02
## AccYear                -0.05       0.00   -0.04    -0.01
## AccMonth                1.00       0.01   -0.01     0.01
## AccWeekday              0.01       1.00    0.00     0.00
## AccTime                -0.01       0.00    1.00     0.03
## RepDelay                0.01       0.00    0.03     1.00
```

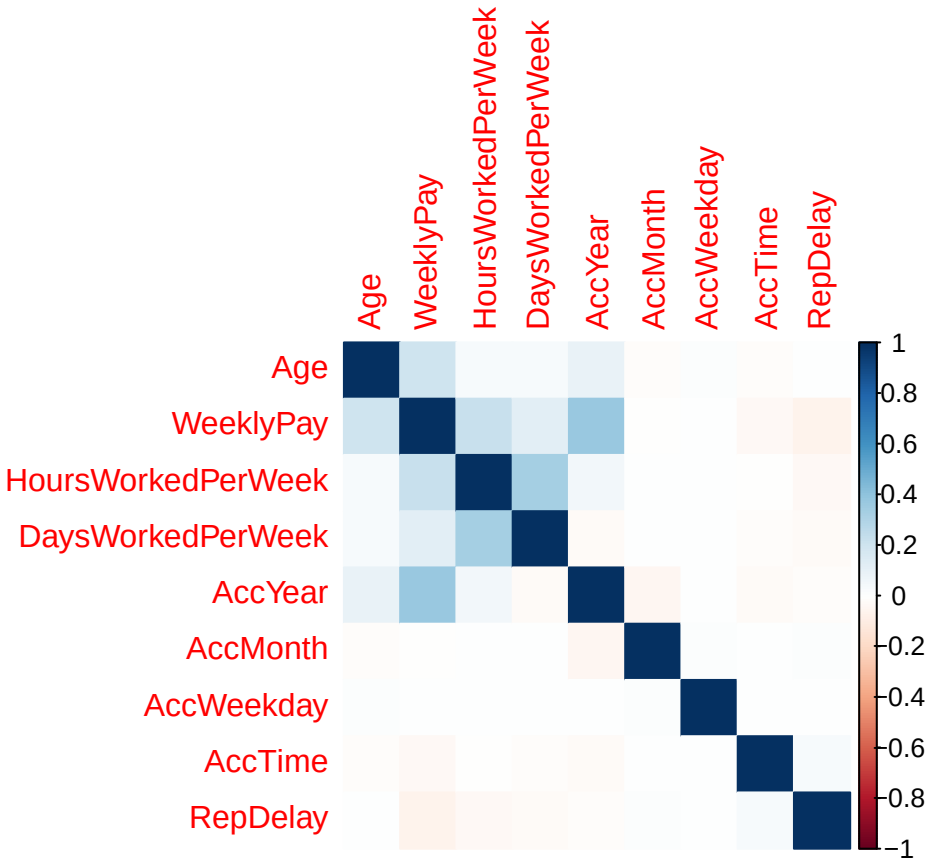
```
corrplot(corrMat, method = "color")
```




```
corrMat <- round(cor(dat_tmp, method = "spearman"), 2)
corrMat
```

```
##                       Age WeeklyPay HoursWorkedPerWeek DaysWorkedPerWeek AccYear
## Age                  1.00      0.20                0.03               0.03    0.09
## WeeklyPay            0.20      1.00                0.22               0.12    0.37
## HoursWorkedPerWeek   0.03      0.22                1.00               0.33    0.05
## DaysWorkedPerWeek    0.03      0.12                0.33               1.00   -0.03
## AccYear              0.09      0.37                0.05              -0.03    1.00
## AccMonth            -0.02     -0.01                0.00               0.00   -0.05
## AccWeekday           0.01      0.00                0.00               0.00    0.00
## AccTime             -0.02     -0.04               -0.01              -0.02   -0.03
## RepDelay             0.00     -0.07               -0.04              -0.03   -0.02
##                     AccMonth AccWeekday AccTime RepDelay
## Age                    -0.02       0.01   -0.02     0.00
## WeeklyPay              -0.01       0.00   -0.04    -0.07
## HoursWorkedPerWeek      0.00       0.00   -0.01    -0.04
## DaysWorkedPerWeek       0.00       0.00   -0.02    -0.03
## AccYear                -0.05       0.00   -0.03    -0.02
## AccMonth                1.00       0.01    0.00     0.01
## AccWeekday              0.01       1.00    0.00     0.00
## AccTime                 0.00       0.00    1.00     0.03
## RepDelay                0.01       0.00    0.03     1.00
```

```
corrplot(corrMat, method = "color")
```



Exercise: Do not exclude the 1'500 claims with zero HoursWokredPerWeek and follow our rationales for excluding them.

Theory: Claim size modelling

Below, we show some screenshots from the tutorial on the tweedie distribution for claim size modelling. See the tutorial for additional theoretical considerations.

3 Tweedie's family for claim size modeling

3.1 Tweedie's distributions

The most popular family of distribution functions for regression modeling is the exponential dispersion family (EDF). An attractive sub-family within the EDF is Tweedie's family [28] which is extensively studied in Jørgensen [9, 10]; we also refer to Chapter 2 in Wüthrich–Merz [31]. Tweedie's family contains, among others, the Gaussian, the Poisson, the gamma and the inverse Gaussian models. The density within Tweedie's family takes the following general form

$$Y \sim f(y;\theta,v/\varphi) = \exp\left\{\frac{y\theta-\kappa_p(\theta)}{\varphi/v}+a(y;v/\varphi)\right\}, \tag{3.1}$$

with exposure $v>0$, dispersion parameter $\varphi>0$, canonical parameter $\theta\in\boldsymbol{\Theta}_p$ in the effective domain $\boldsymbol{\Theta}_p\subseteq\mathbb{R}$, normalization $a(\cdot;\cdot)$ *not* depending on the canonical parameter θ, and cumulant function $\kappa_p:\boldsymbol{\Theta}_p\to\mathbb{R}$ given by

$$\kappa_p(\theta)=\begin{cases}\frac{1}{2-p}\left((1-p)\theta\right)^{\frac{2-p}{1-p}} & \text{for } p\in\mathbb{R}\setminus((0,1]\cup\{2\}),\\ \exp\{\theta\} & \text{for } p=1,\\ -\log(-\theta) & \text{for } p=2.\end{cases} \tag{3.2}$$

For $p\in(0,1)$ there do not exist any Tweedie's distribution functions, see Theorem 2 in Jørgensen [9]. Tweedie's family is summarized in Table 1.

p	distribution	support of Y	$\boldsymbol{\Theta}_p$
$p<0$	generated by extreme stable distributions	$\mathbb{R}$	$[0,\infty)$
$p=0$	Gaussian distribution	$\mathbb{R}$	$\mathbb{R}$
$p=1$	Poisson distribution	$\mathbb{N}_0$	$\mathbb{R}$
$1<p<2$	Tweedie's compound Poisson distribution	$[0,\infty)$	$(-\infty,0)$
$p=2$	gamma distribution	$(0,\infty)$	$(-\infty,0)$
$p>2$	generated by positive stable distributions	$(0,\infty)$	$(-\infty,0]$
$p=3$	inverse Gaussian distribution	$(0,\infty)$	$(-\infty,0]$

Table 1: Tweedie's family; this table is taken from Table 4.1 in Jørgensen [10].

The characterizing property of Tweedie's family is that a Tweedie distributed random variable Y has the first two moments for canonical parameter $\theta\in\boldsymbol{\Theta}_p$ given by

$$\mu=\mu(\theta)=\mathbb{E}[Y]=\kappa_p'(\theta) \qquad \text{and} \qquad \mathrm{Var}(Y)=\frac{\varphi}{v}\kappa_p''(\theta)=\frac{\varphi}{v}\mu^p=\frac{\varphi}{v}V(\mu)>0, \tag{3.3}$$

the latter describes the meaning of parameter p and it motivates the definition of the variance function $\mu\mapsto V_p(\mu)=\mu^p$ within Tweedie's family. For this reason, Tweedie's family is said to have power variance functions with power variance parameter $p\in\mathbb{R}\setminus(0,1)$.

[2] The i.i.d. assumption applies to $(Y_i,x_i)_i$. The responses Y_i, given covariates x_i, are only independent, but not identically distributed because the distribution typically differs in different covariates x_i.

☞ For claim size modeling power variance parameters $p \geq 2$ are most suitable; this includes the (absolutely continuous) gamma and the inverse Gaussian models. For claim size modeling we set exposure $v = 1$. This provides us with coefficient of variation (CoV)

$$\mathrm{CoV}(Y) = \frac{\sqrt{\mathrm{Var}(Y)}}{\mathbb{E}[Y]} = \varphi \mu^{p/2-1}.$$

Thus, for $\mu > 1$ the coefficient of variation is increasing in power variance parameter $p \geq 2$. Nevertheless, all members of Tweedie's family are light-tailed as we have exponentially decaying survival function, that is, the survival function $1 - F(y; \theta, v/\varphi)$ of Y decays exponentially for $y \to \infty$, see Remarks 2.11 and Section 2.2.5 in Wüthrich–Merz [31].

General modelling: parameters

First, we define the two parameters containing the features for the regression modeling, differentiating between the name of the feature and the (standardized) features in the data.

```
# used/selected features
col_features <- c("AgeNN","GenderNN","DependentChildrenNN","DependentsOtherNN",
                  "WeeklyPayNN","PartTimeFullTimeNN","HoursWorkedPerWeekNN",
                  "DaysWorkedPerWeekNN","AccYearNN","AccMonthNN","AccWeekdayNN",
                  "AccTimeNN","RepDelayNN","Marital1","Marital2","Marital3")
col_names <- c("Age","Gender","DependentChildren","DependentsOther","WeeklyPay",
               "PartTimeFullTime","HoursWorkedPerWeek","DaysWorkedPerWeek",
               "AccYear","AccMonth","AccWeekday","AccTime","RepDelay",
               "Marital1","Marital2","Marital3")
```

We choose a value $2 < p < 3$ as proposed in the theoretical part above.

```
# select p in [2,3]
p <- 2.5
```

Model(s) 0: Null model

We fit the null model for power variance parameters $p = 2$ (gamma model), $p = 2.5$ and $p = 3$ (inverse Gaussian model) to the accident insurance claim data.

In the null model the MLE is easily obtained , and it does not depend on the specific choice of p.

```
# homogeneous model (learn)
(size_hom <- round(mean(learn$Claim)))
```

```
## [1] 12733
```

```
log_size_hom <- log(size_hom)
```

```
df_cmp %<>% bind_rows(
  data.frame(
    model = "Null model",
    learn_p2 = round(gamma_loss(learn$Claim, size_hom), 4),
    learn_pp = round(p_loss(learn$Claim, size_hom, p) * 10, 4),
    learn_p3 = round(ig_loss(learn$Claim, size_hom) * 1000, 4),
    test_p2 = round(gamma_loss(test$Claim, size_hom), 4),
    test_pp = round(p_loss(test$Claim, size_hom, p) * 10, 4),
    test_p3 = round(ig_loss(test$Claim, size_hom) * 1000, 4),
```

```
    avg_size = round(size_hom, 0)
  ))
df_cmp
```

```
## # A tibble: 1 x 8
##   model      learn_p2 learn_pp learn_p3 test_p2 test_pp test_p3 avg_size
##   <chr>         <dbl>    <dbl>    <dbl>   <dbl>   <dbl>   <dbl>    <dbl>
## 1 Null model     5.13     1.02     3.59    5.13    1.02    3.59    12733
```

The table shows the resulting in-sample and out-of-sample deviance losses for power variance parameters $p = 2, 2.5, 3$. We use these results as benchmarks for the subsequent models. We remark that the losses in this table live on different scales for different power variance parameters p because the unit deviances take different functional forms for different p's.

Designing Neural Networks

We do not provide much details in this notebook on the choice of a particular network architecture and its calibration. We refer to the notebook and tutorial on Feed-Forward Neural Networks (FNN) for the French Motor Third Party Liability Claims for the details, see here (tutorial) or here (notebook)

Epochs and batches

Epochs indicates how many times we go through the entire learning data $\mathcal{D}$, and batch size indicates the size of the subsamples considered in each Gradient Descent Method (GDM) step. Thus, if the batch size is equal to the number of observations n we do exactly one GDM step in one epoch, if the batch size is equal to 1 then we do n GDM steps in one epoch until we have seen the entire learning data $\mathcal{D}$. Note that smaller batches are needed for big data because it is not feasible to simultaneously calculate the gradient on all data efficiently if we have many observations. Therefore, we partition the entire data at random into (mini-) batches in the application of the GDM. Note that this partitioning of the data is of particular interest if we work with big data because it allows us to explore the data sequentially.

Concretely, for the maximal batch size n we can do exactly one GDM step in one epoch, for batch size k we can do n/k GDM steps in one epoch. For the maximal batch size we need to calculate the gradient on the entire data $\mathcal{D}$. The latter is, of course, much faster but on the other hand we need to calculate n/k gradients to run through the entire data (in an epoch).

The partitioning of the data $\mathcal{D}$ into batches is done at random, and it may happen that several potential outliers lie in the same batch. This happens especially if the chosen batch size is small and the expected frequency is low (class imbalance problem).

Model(s) 1: Plain-vanilla Neural Networks

We do not discuss here the definition of feed-forward neural networks (FFN) and details for the fitting, see the corresponding tutorial and the references therein for further details.

We fit a FFN network architecture to the different deviance losses L_p with power variance parameters $p = 2$ (gamma model), $p = 2.5$ and $p = 3$ (inverse Gaussian (IG) model) to the accident insurance claim data. As depth of the FNN network we choose $d = 3$ having $(q_1, q_2, q_3) = (20, 15, 10)$ neurons in the FFN layers. We choose hyperbolic tangent activation function for ϕ_m, $1 \leq m \leq d$, and log-link for g. The latter is not the canonical link for power variance parameters $p \geq 2$, but it ensures that range and domain of the link function g match the effective domain and the means $\mu(\theta)$ for power variance parameters $p \geq 2$.

Common neural network specifications

Below we define the network parameters.

```
# Size of input for neural networks
q0 <- length(col_features)
qqq <- c(q0, c(20,15,10), 1)

sprintf("Neural network with K=3 hidden layer")
```

```
## [1] "Neural network with K=3 hidden layer"
```

```
sprintf("Input feature dimension: q0 = %s", q0)
```

```
## [1] "Input feature dimension: q0 = 16"
```

```
sprintf("Number of hidden neurons first layer: q1 = %s", qqq[2])
```

```
## [1] "Number of hidden neurons first layer: q1 = 20"
```

```
sprintf("Number of hidden neurons second layer: q2 = %s", qqq[3])
```

```
## [1] "Number of hidden neurons second layer: q2 = 15"
```

```
sprintf("Number of hidden neurons third layer: q3 = %s", qqq[4])
```

```
## [1] "Number of hidden neurons third layer: q3 = 10"
```

```
sprintf("Output dimension: %s", qqq[5])
```

```
## [1] "Output dimension: 1"
```

Below we define the response, learning and testing matrix required as input to the `keras` functions.

```
# matrices
YY <- as.matrix(as.numeric(learn$Claim))
XX <- as.matrix(learn[, col_features])
TT <- as.matrix(test[, col_features])
```

Plain-vanilla Gamma Neural Network (p=2)

Definition

In this notebook we do not provide further details on the layers used and the structure, we refer to other notebooks or the references at the end of this notebook. A good reference is the keras cheat sheet.

```
Design  <- layer_input(shape = c(qqq[1]), dtype = 'float32', name = 'design')

Output <- Design %>%
    layer_dense(units = qqq[2], activation = 'tanh', name = 'layer1') %>%
    layer_dense(units = qqq[3], activation = 'tanh', name = 'layer2') %>%
    layer_dense(units = qqq[4], activation = 'tanh', name = 'layer3') %>%
    layer_dense(units = 1, activation = 'exponential', name = 'output',
                weights = list(array(0, dim = c(qqq[4], 1)), array(log_size_hom, dim = c(1))))

model_p2 <- keras_model(inputs = list(Design), outputs = c(Output))
```

Compilation

Let us compile the model, using the tweedie deviance loss function as objective function, and nadam as the optimizer, and we provide a summary of the network structure.

For further details, we refer to the help file of compile here.

```
model_p2 %>% compile(
    loss = k_gamma_loss,
    optimizer = 'nadam'
)

summary(model_p2)
```

```
## Model: "model"
## _________________________________________________________________________
## Layer (type)                 Output Shape              Param #
## =========================================================================
## design (InputLayer)          [(None, 16)]              0
## _________________________________________________________________________
## layer1 (Dense)               (None, 20)                340
## _________________________________________________________________________
## layer2 (Dense)               (None, 15)                315
## _________________________________________________________________________
## layer3 (Dense)               (None, 10)                160
## _________________________________________________________________________
## output (Dense)               (None, 1)                 11
## =========================================================================
## Total params: 826
## Trainable params: 826
## Non-trainable params: 0
## _________________________________________________________________________
```

This summary is crucial for a good understanding of the fitted model. It contains the total number of parameters and shows what the exposure is included as an offset (without training the corresponding weight).

Fitting

For fitting a keras model and its arguments, we refer to the help of `fit` here. There you find details about the `validation_split` and the `verbose` argument.

If validation_split>0, then the training set is further subdivided into a new training and a validation set. The new training set is used for fitting the model and the validation set is used for (out-of-sample) validation. We emphasize that the validation set is chosen disjointly from the test data, as this latter data may still be used later for the choice of the optimal model (if, for instance, we need to decide between several networks). With validation_split=0.2 we split the learning data 8:2 into training set and validation set. We fit the network on the training set and we out-of-sample validate it on the validation set.

We choose a mini-batch size of 5'000 and the maximum number of epochs to be 100.

```
# set hyperparameters
epochs <- 100
batch_size <- 5000
validation_split <- 0.2 # set to >0 to see train/validation loss in plot(fit)
verbose <- 1
```

The number of epochs defines how many times we go through the entire learning data $\mathcal{D}$. Hence the best result is not achieved by the network with the maximum number of epochs, but rather an epoch value below the maximum. As a consequence, we only need the model with the minium validation loss on the validation set. This is achieved through the `callback_model_checkpoint` function here. More details about save and store models is provided here.

```
# store and use only the best model
cp_path <- paste("./Networks/model_p2")
```

```
cp_callback <- callback_model_checkpoint(
    filepath = cp_path,
    monitor = "val_loss",
    save_weights_only = TRUE,
    save_best_only = TRUE,
    verbose = 0
)
```

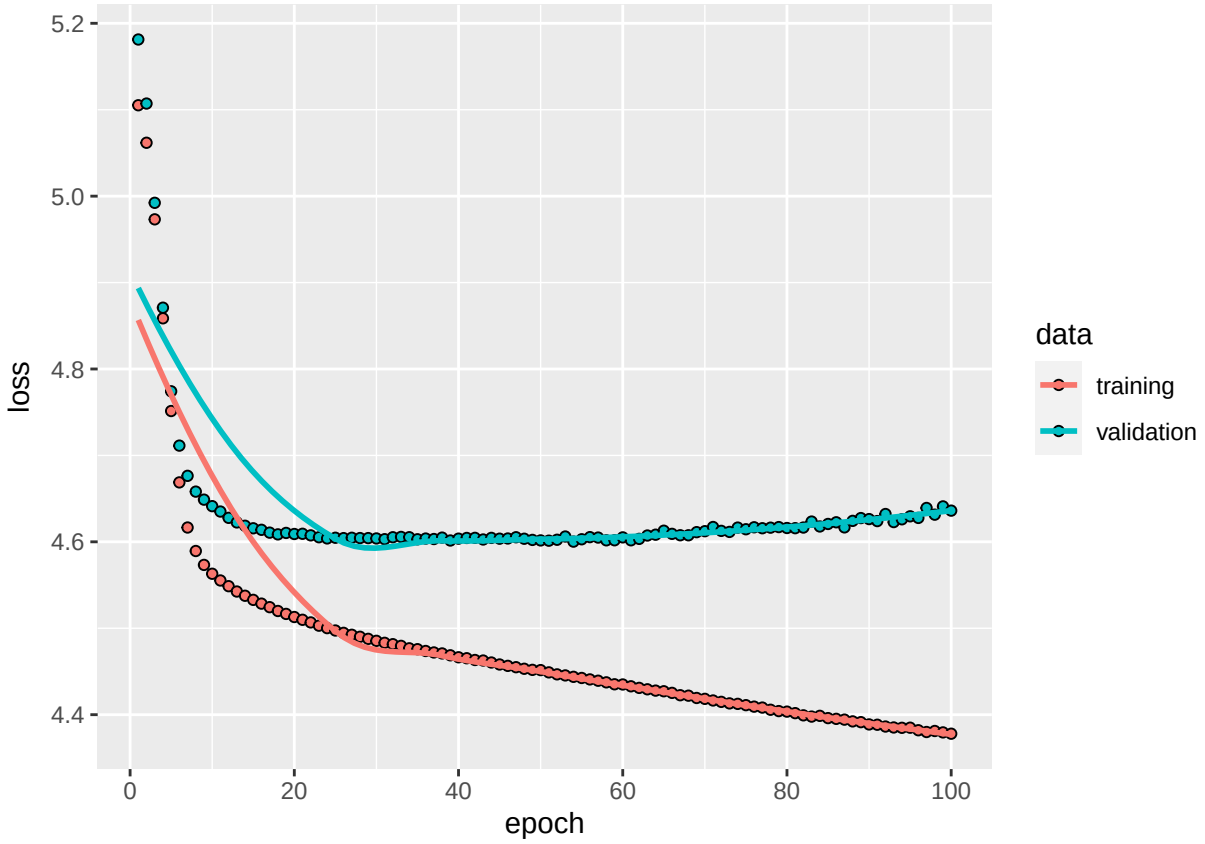
```
fit_p2 <- model_p2 %>%
  fit(
    list(XX), list(YY),
    validation_split = validation_split,
    epochs = epochs,
    batch_size = batch_size,
    callbacks = list(cp_callback),
    verbose = verbose
  )
```

Let us illustrate the decrease of loss during the gradient descent algorithm. We provide two charts below * First one: Depending on the argument validation_split you see one curve or two curves (training and validation). * Second one: Illustrated the optimal model (which is stored above)

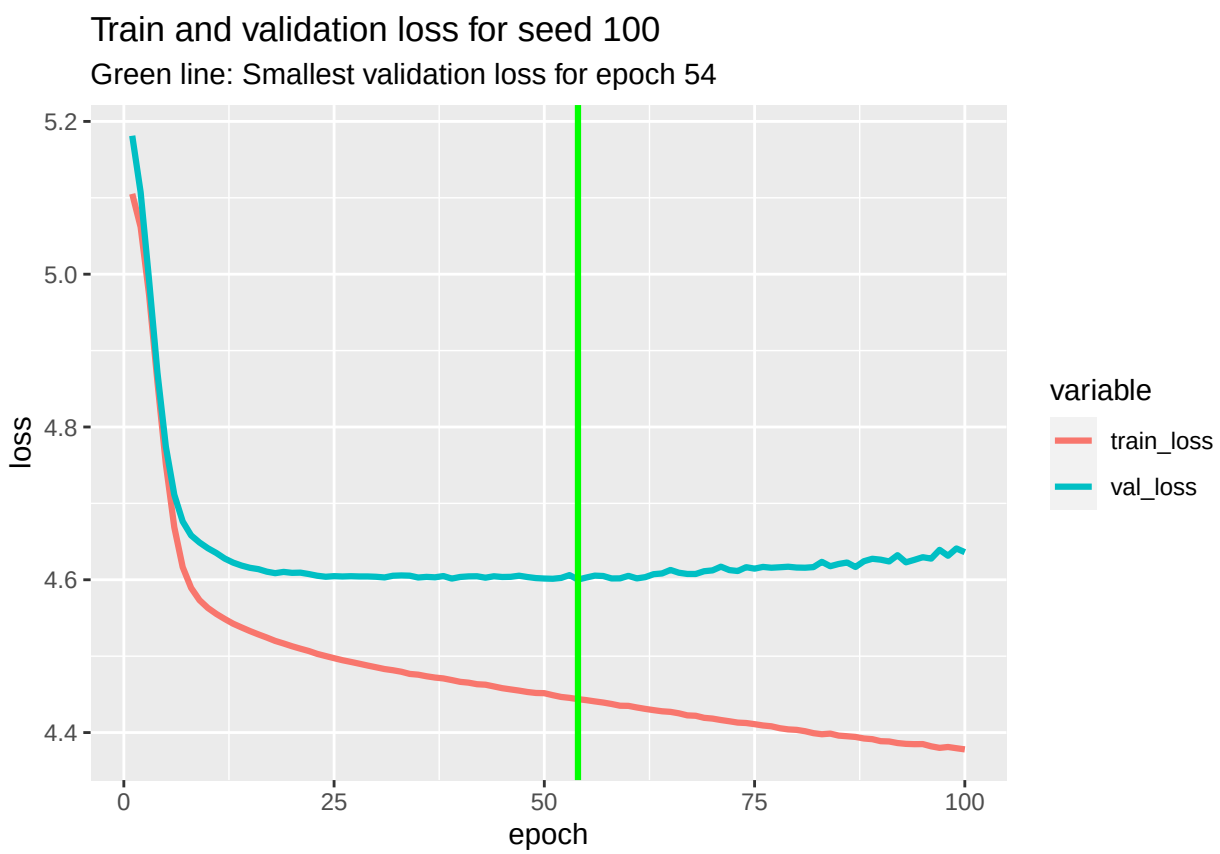
The plots help to determine the optimal number of epochs.

```
plot(fit_p2)
```

```
## `geom_smooth()` using formula 'y ~ x'
```



```
keras_plot_loss_min(fit_p2, seed)
```




```
load_model_weights_hdf5(model_p2, cp_path)
```

Validation

```
# calculating the predictions
learn$fitshp2 <- as.vector(model_p2 %>% predict(list(XX)))
test$fitshp2 <- as.vector(model_p2 %>% predict(list(TT)))

# average in-sample and out-of-sample losses (in 10^(0))
sprintf("Gamma deviance shallow network (train): %s", round(gamma_loss(learn$Claim, learn$fitshp2), 4))
```

```
## [1] "Gamma deviance shallow network (train): 4.4718"
```

```
sprintf("Gamma deviance shallow network (test): %s", round(gamma_loss(test$Claim, test$fitshp2), 4))
```

```
## [1] "Gamma deviance shallow network (test): 4.5896"
```

```
# average claims size
sprintf("Average size (test): %s", round(mean(test$fitshp2), 1))
```

```
## [1] "Average size (test): 12728.8"
```

```
df_cmp %<>% bind_rows(
  data.frame(model = "Plain-vanilla p2 (gamma)",
             learn_p2 = round(gamma_loss(learn$Claim, learn$fitshp2), 4),
             learn_pp = round(p_loss(learn$Claim, learn$fitshp2, p) * 10, 4),
             learn_p3 = round(ig_loss(learn$Claim, learn$fitshp2) * 1000, 4),
```

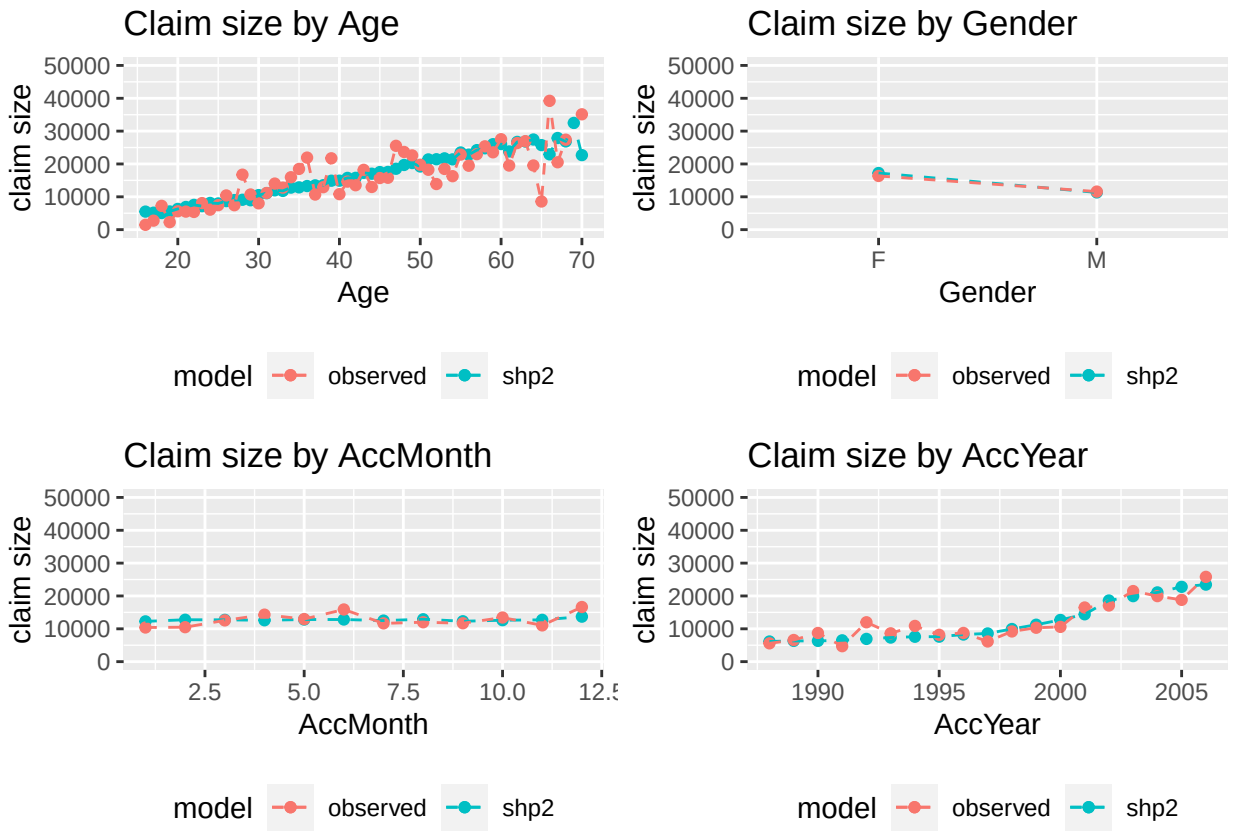
```
            test_p2 = round(gamma_loss(test$Claim, test$fitshp2), 4),
            test_pp = round(p_loss(test$Claim, test$fitshp2, p) * 10, 4),
            test_p3 = round(ig_loss(test$Claim, test$fitshp2) * 1000, 4),
            avg_size = round(mean(test$fitshp2), 0)
  ))
df_cmp
```

```
## # A tibble: 2 x 8
##   model              learn_p2 learn_pp learn_p3 test_p2 test_pp test_p3 avg_size
##   <chr>                 <dbl>    <dbl>    <dbl>   <dbl>   <dbl>   <dbl>    <dbl>
## 1 Null model             5.13    1.02      3.59    5.13    1.02     3.59    12733
## 2 Plain-vanilla p2 ~     4.47    0.952     3.51    4.59    0.965    3.53    12729
```

Calibration

```
# Age
plt1 <- plot_size(test, "Age", "Claim size by Age", "shp2", "fitshp2")
# Gender
plt2 <- plot_size(test, "Gender", "Claim size by Gender", "shp2", "fitshp2")
# AccMonth
plt3 <- plot_size(test, "AccMonth", "Claim size by AccMonth", "shp2", "fitshp2")
# AccYear
plt4 <- plot_size(test, "AccYear", "Claim size by AccYear", "shp2", "fitshp2")

grid.arrange(plt1, plt2, plt3, plt4)
```

```
## Warning: Removed 1 rows containing missing values (geom_point).
```



Is worthwhile to remark that the fit is quite close to the observations and that the neural networks smooths out some jumps in the observations which is in line with expectations.

Below, we do not provide the generally valid comments again below, only if there are specific comments to the model.

$2 < p < 3$ Neural Network

Definition

```
Design  <- layer_input(shape = c(qqq[1]), dtype = 'float32', name = 'design')

Output <- Design %>%
    layer_dense(units=qqq[2], activation='tanh', name='layer1') %>%
    layer_dense(units=qqq[3], activation='tanh', name='layer2') %>%
    layer_dense(units=qqq[4], activation='tanh', name='layer3') %>%
    layer_dense(units=1, activation='exponential', name='output',
                weights=list(array(0, dim=c(qqq[4],1)), array(log_size_hom, dim=c(1))))

model_pp <- keras_model(inputs = list(Design), outputs = c(Output))
```

Compilation

```
model_pp %>% compile(
    loss = k_p_loss,
    optimizer = 'nadam'
)

summary(model_pp)
```

```
## Model: "model_1"
## _________________________________________________________________________
## Layer (type)                 Output Shape              Param #
## =========================================================================
## design (InputLayer)          [(None, 16)]              0
## _________________________________________________________________________
## layer1 (Dense)               (None, 20)                340
## _________________________________________________________________________
## layer2 (Dense)               (None, 15)                315
## _________________________________________________________________________
## layer3 (Dense)               (None, 10)                160
## _________________________________________________________________________
## output (Dense)               (None, 1)                 11
## =========================================================================
## Total params: 826
## Trainable params: 826
## Non-trainable params: 0
## _________________________________________________________________________
```

Fitting

```
# set hyperparameters
epochs <- 100
batch_size <- 5000
```

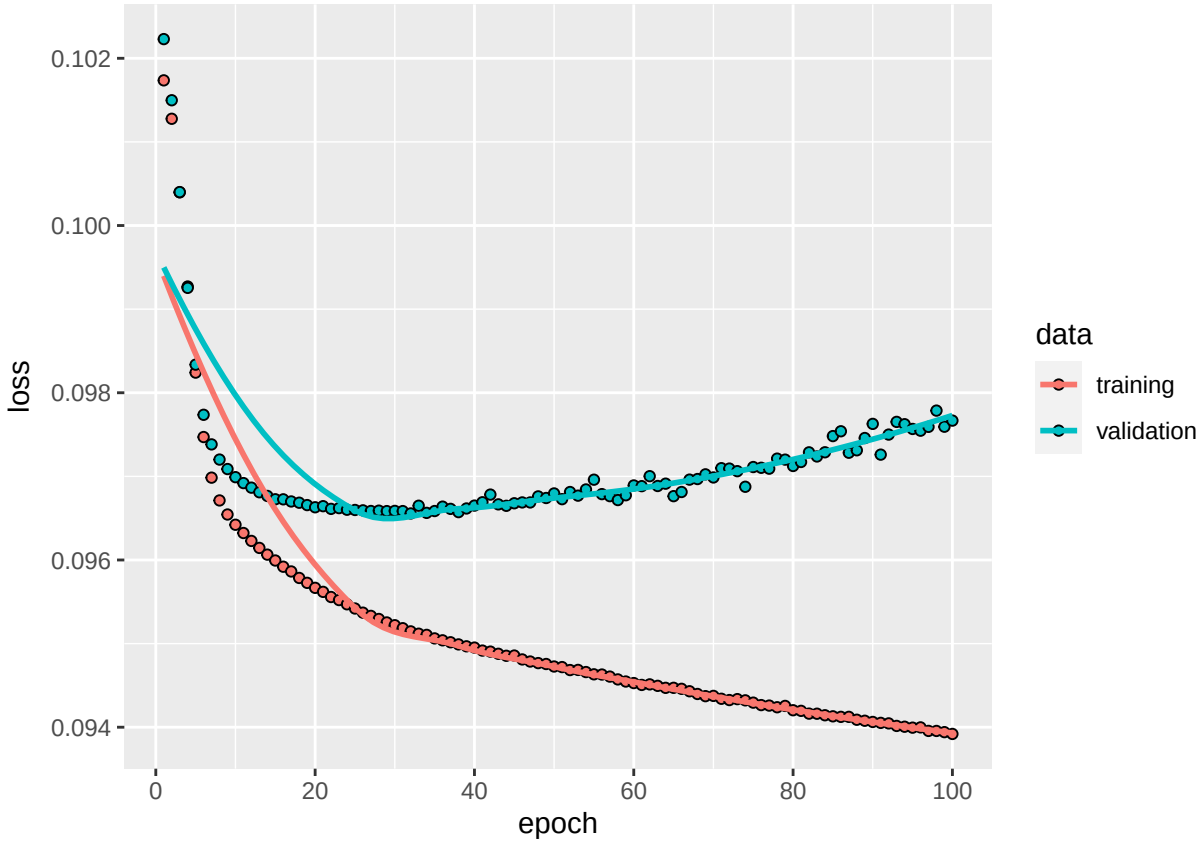
```
validation_split <- 0.2 # set to >0 to see train/validation loss in plot(fit)
verbose <- 1
```

```
# store and use only the best model
cp_path <- paste("./Networks/model_pp")

cp_callback <- callback_model_checkpoint(
    filepath = cp_path,
    monitor = "val_loss",
    save_weights_only = TRUE,
    save_best_only = TRUE,
    verbose = 0
)
```

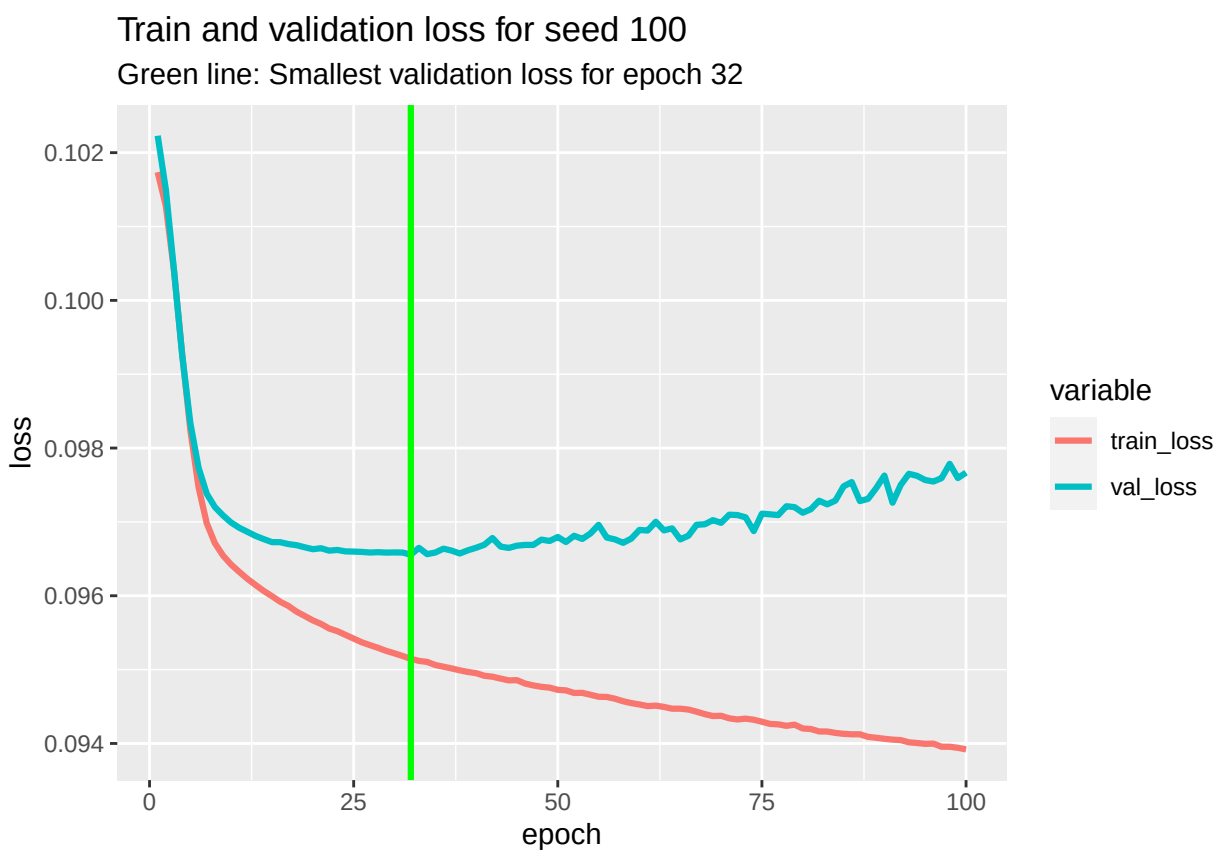
```
fit_pp <- model_pp %>% fit(
    list(XX), list(YY),
    validation_split = validation_split,
    epochs = epochs,
    batch_size = batch_size,
    callbacks = list(cp_callback),
    verbose = verbose
)
```

```
plot(fit_pp)
```

```
## `geom_smooth()` using formula 'y ~ x'
```



```
keras_plot_loss_min(fit_pp, seed)
```




```
load_model_weights_hdf5(model_pp, cp_path)
```

Validation

```
# calculating the predictions
learn$fitshpp <- as.vector(model_pp %>% predict(list(XX)))
test$fitshpp <- as.vector(model_pp %>% predict(list(TT)))

# average in-sample and out-of-sample losses (in 10^(0))
sprintf("p-loss deviance shallow network (train): %s", round(p_loss(learn$Claim, learn$fitshpp, p), 4))
```

```
## [1] "p-loss deviance shallow network (train): 0.0954"
```

```
sprintf("p-loss deviance shallow network (test): %s", round(p_loss(test$Claim, test$fitshpp, p), 4))
```

```
## [1] "p-loss deviance shallow network (test): 0.0969"
```

```
# average claims size
sprintf("Average size (test): %s", round(mean(test$fitshpp), 1))
```

```
## [1] "Average size (test): 12953.8"
```

```
df_cmp %<>% bind_rows(
  data.frame(model = paste0("Plain-vanilla pp (p=", p,")"),
             learn_p2 = round(gamma_loss(learn$Claim, learn$fitshpp), 4),
             learn_pp = round(p_loss(learn$Claim, learn$fitshpp, p) * 10, 4),
             learn_p3 = round(ig_loss(learn$Claim, learn$fitshpp) * 1000, 4),
```

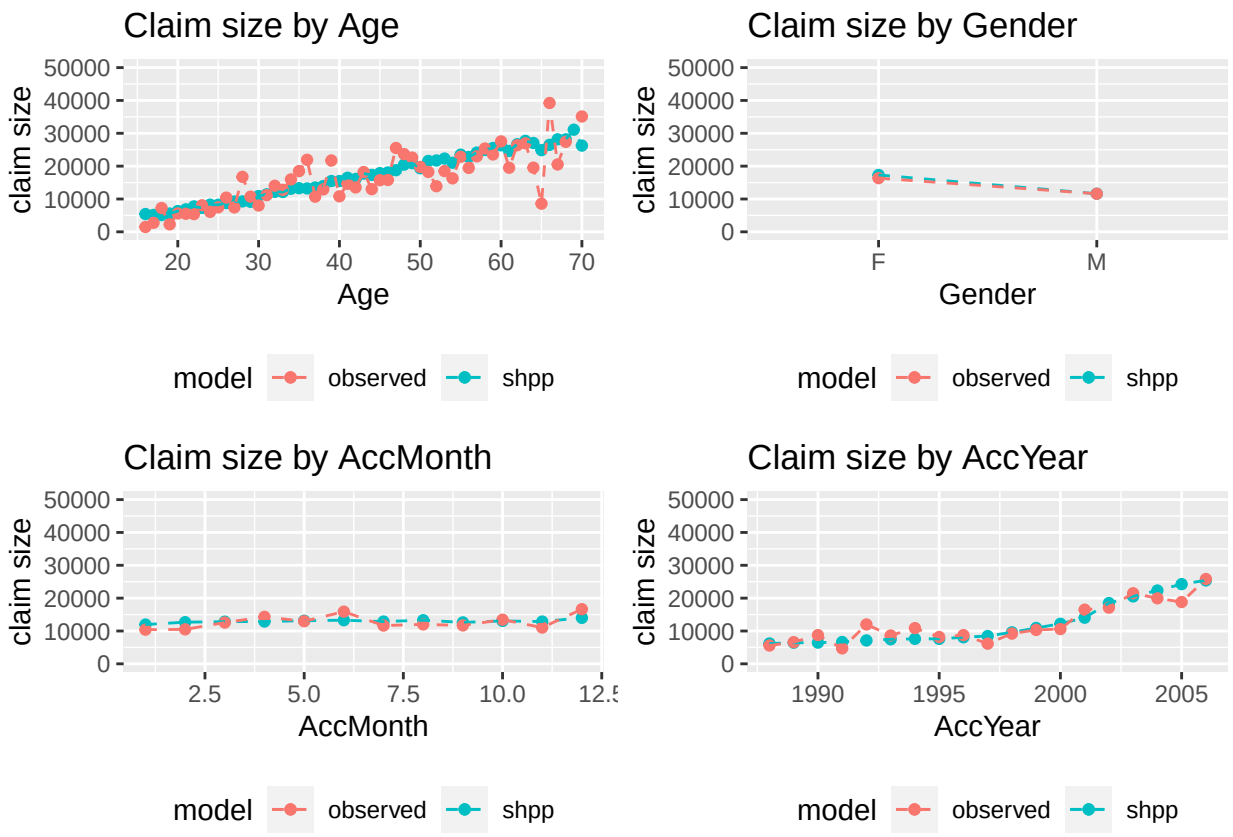
```
            test_p2 = round(gamma_loss(test$Claim, test$fitshpp), 4),
            test_pp = round(p_loss(test$Claim, test$fitshpp, p) * 10, 4),
            test_p3 = round(ig_loss(test$Claim, test$fitshpp) * 1000, 4),
            avg_size = round(mean(test$fitshpp), 0)
  ))
df_cmp
```

```
## # A tibble: 3 x 8
##   model              learn_p2 learn_pp learn_p3 test_p2 test_pp test_p3 avg_size
##   <chr>                 <dbl>    <dbl>    <dbl>   <dbl>   <dbl>   <dbl>    <dbl>
## 1 Null model             5.13    1.02      3.59    5.13   1.02     3.59    12733
## 2 Plain-vanilla p2 ~     4.47    0.952     3.51    4.59   0.965    3.53    12729
## 3 Plain-vanilla pp ~     4.50    0.954     3.52    4.62   0.969    3.54    12954
```

Calibration

```
# Age
plt1 <- plot_size(test, "Age", "Claim size by Age", "shpp", "fitshpp")
# Gender
plt2 <- plot_size(test, "Gender", "Claim size by Gender", "shpp", "fitshpp")
# AccMonth
plt3 <- plot_size(test, "AccMonth", "Claim size by AccMonth", "shpp", "fitshpp")
# AccYear
plt4 <- plot_size(test, "AccYear", "Claim size by AccYear", "shpp", "fitshpp")

grid.arrange(plt1, plt2, plt3, plt4)
```

```
## Warning: Removed 1 rows containing missing values (geom_point).
```



Plain-vanilla Inverse Gaussian Neural Network (p=3)

Definition

```
Design  <- layer_input(shape = c(qqq[1]), dtype = 'float32', name = 'design')

Output <- Design %>%
    layer_dense(units=qqq[2], activation='tanh', name='layer1') %>%
    layer_dense(units=qqq[3], activation='tanh', name='layer2') %>%
    layer_dense(units=qqq[4], activation='tanh', name='layer3') %>%
    layer_dense(units=1, activation='exponential', name='output',
                weights=list(array(0, dim=c(qqq[4],1)), array(log_size_hom, dim=c(1))))

model_p3 <- keras_model(inputs = list(Design), outputs = c(Output))
```

Compilation

```
model_p3 %>% compile(
    loss = k_ig_loss,
    optimizer = 'nadam'
)

summary(model_p3)
```

```
## Model: "model_2"
## ________________________________________________________________________________
## Layer (type)                        Output Shape                    Param #
```

```
## ================================================================================
## design (InputLayer)                 [(None, 16)]                0
## ________________________________________________________________________________
## layer1 (Dense)                      (None, 20)                  340
## ________________________________________________________________________________
## layer2 (Dense)                      (None, 15)                  315
## ________________________________________________________________________________
## layer3 (Dense)                      (None, 10)                  160
## ________________________________________________________________________________
## output (Dense)                      (None, 1)                   11
## ================================================================================
## Total params: 826
## Trainable params: 826
## Non-trainable params: 0
## ________________________________________________________________________________
```

Fitting

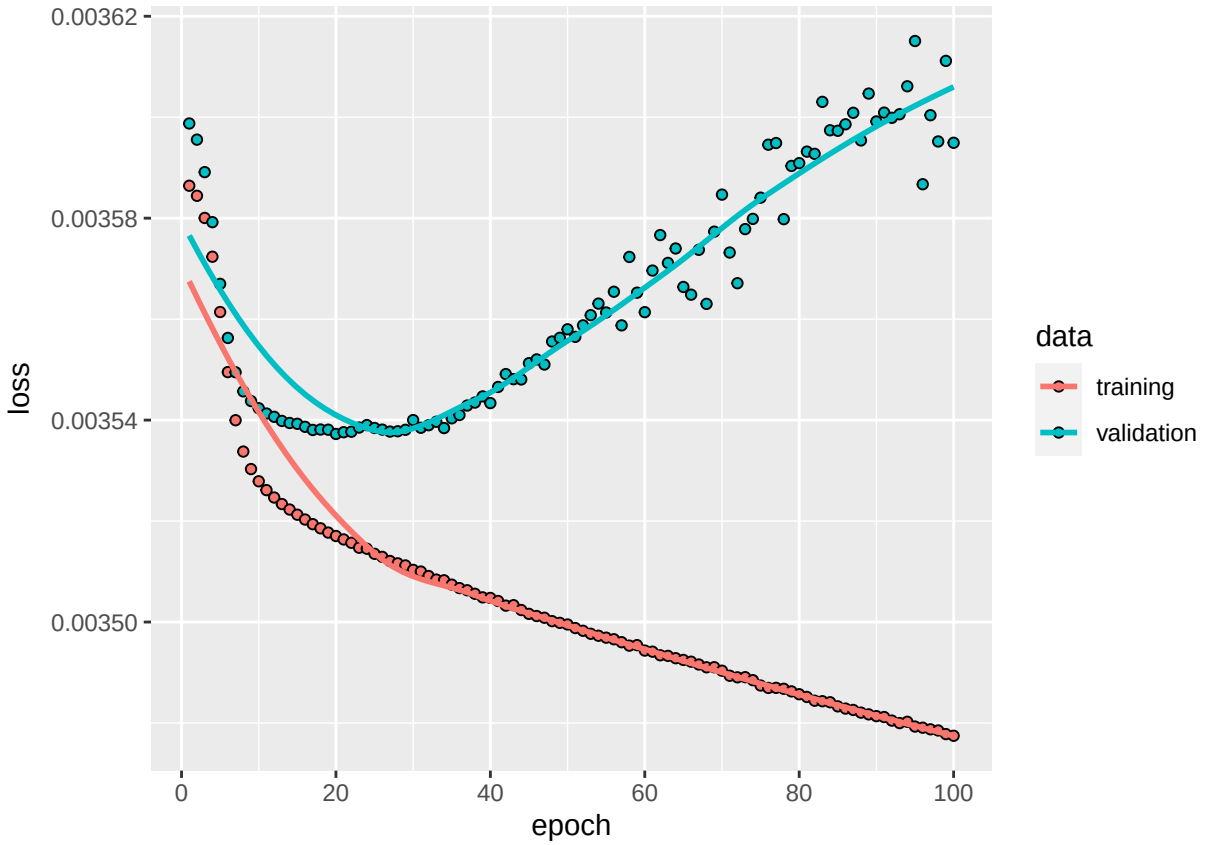
```
# set hyperparameters
epochs <- 100
batch_size <- 5000
validation_split <- 0.2 # set to >0 to see train/validation loss in plot(fit)
verbose <- 1
```

```
# store and use only the best model
cp_path <- paste("./Networks/model_p3")

cp_callback <- callback_model_checkpoint(
    filepath = cp_path,
    monitor = "val_loss",
    save_weights_only = TRUE,
    save_best_only = TRUE,
    verbose = 0
)
```

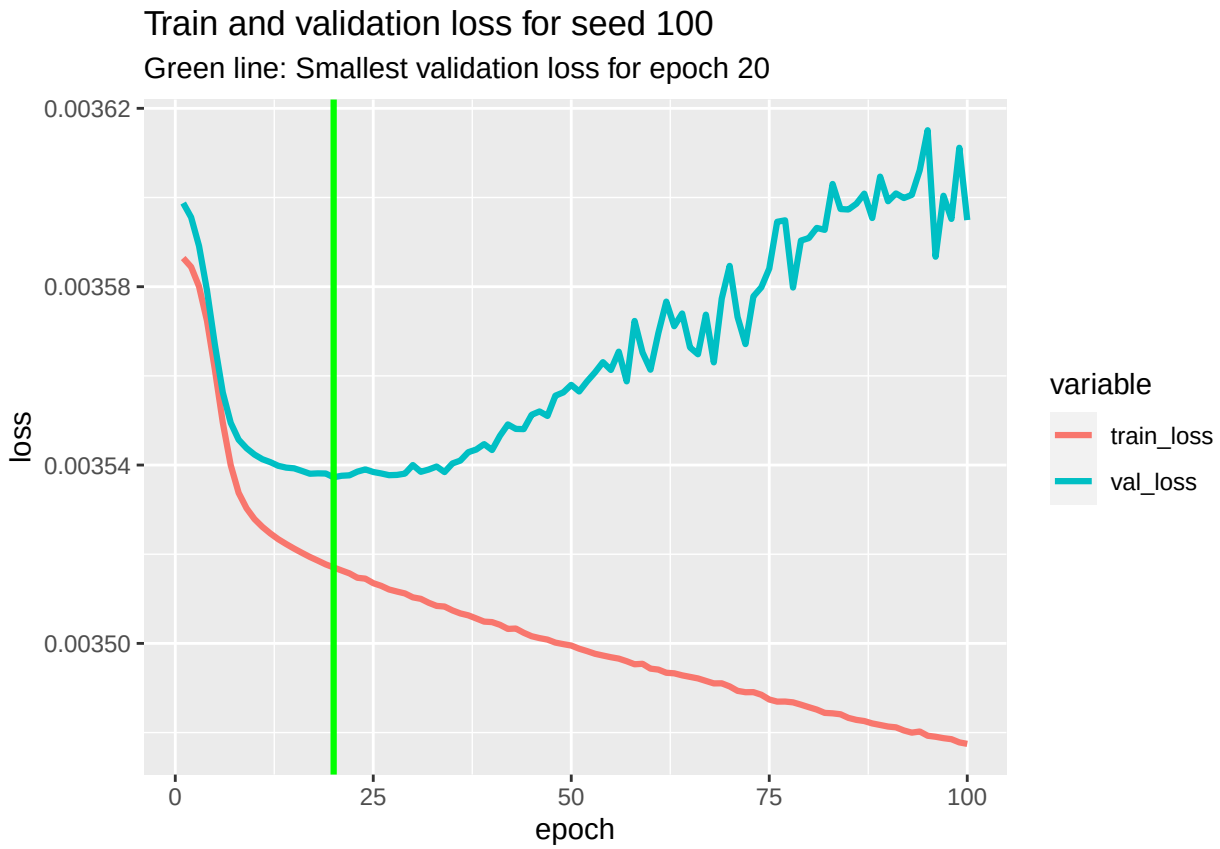
```
fit_p3 <- model_p3 %>% fit(
    list(XX), list(YY),
    validation_split = validation_split,
    epochs = epochs,
    batch_size = batch_size,
    callbacks = list(cp_callback),
    verbose = verbose
)
```

```
plot(fit_p3)
```

```
## `geom_smooth()` using formula 'y ~ x'
```




```
keras_plot_loss_min(fit_p3, seed)
```




```
load_model_weights_hdf5(model_p3, cp_path)
```

Validation

```
# calculating the predictions
learn$fitshp3 <- as.vector(model_p3 %>% predict(list(XX)))
test$fitshp3 <- as.vector(model_p3 %>% predict(list(TT)))

# average in-sample and out-of-sample losses (in 10^(0))
sprintf("IG deviance shallow network (train): %s", round(ig_loss(learn$Claim, learn$fitshp3), 4))
```

```
## [1] "IG deviance shallow network (train): 0.0035"
```

```
sprintf("IG deviance shallow network (test): %s", round(ig_loss(test$Claim, test$fitshp3), 4))
```

```
## [1] "IG deviance shallow network (test): 0.0035"
```

```
# average claims size
sprintf("Average size (test): %s", round(mean(test$fitshp3), 1))
```

```
## [1] "Average size (test): 14216.1"
```

```
df_cmp %<>% bind_rows(
  data.frame(model = "Plain-vanilla p3 (inverse gaussian)",
             learn_p2 = round(gamma_loss(learn$Claim, learn$fitshp3), 4),
             learn_pp = round(p_loss(learn$Claim, learn$fitshp3, p) * 10, 4),
             learn_p3 = round(ig_loss(learn$Claim, learn$fitshp3) * 1000, 4),
             test_p2 = round(gamma_loss(test$Claim, test$fitshp3), 4),
             test_pp = round(p_loss(test$Claim, test$fitshp3, p) * 10, 4),
```

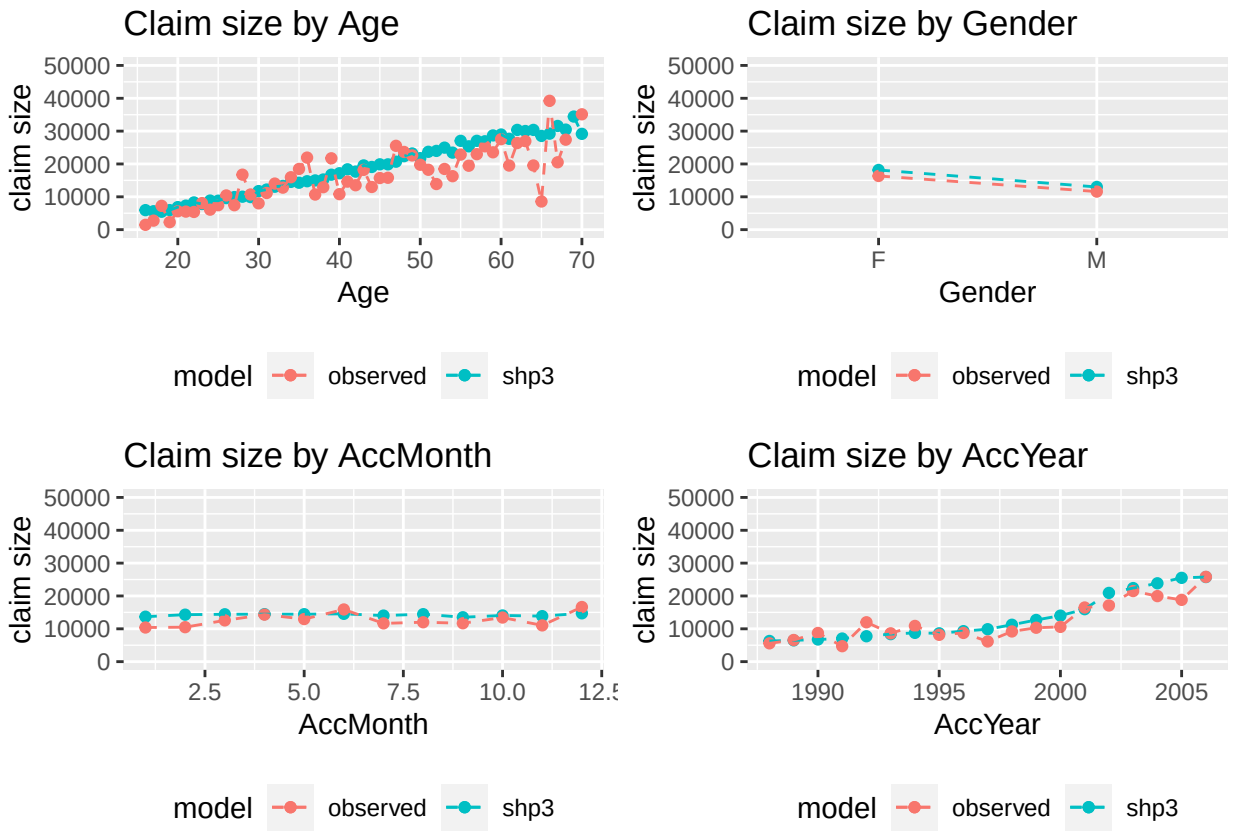
```
            test_p3 = round(ig_loss(test$Claim, test$fitshp3) * 1000, 4),
            avg_size = round(mean(test$fitshp3), 0)
  ))
df_cmp
```

```
## # A tibble: 4 x 8
##   model              learn_p2 learn_pp learn_p3 test_p2 test_pp test_p3 avg_size
##   <chr>                 <dbl>    <dbl>    <dbl>   <dbl>   <dbl>   <dbl>    <dbl>
## 1 Null model             5.13     1.02     3.59    5.13    1.02     3.59    12733
## 2 Plain-vanilla p2 ~     4.47    0.952     3.51    4.59   0.965     3.53    12729
## 3 Plain-vanilla pp ~     4.50    0.954     3.52    4.62   0.969     3.54    12954
## 4 Plain-vanilla p3 ~     4.52    0.957     3.52    4.59   0.964     3.53    14216
```

Calibration

```
# Age
plt1 <- plot_size(test, "Age", "Claim size by Age", "shp3", "fitshp3")
# Gender
plt2 <- plot_size(test, "Gender", "Claim size by Gender", "shp3", "fitshp3")
# AccMonth
plt3 <- plot_size(test, "AccMonth", "Claim size by AccMonth", "shp3", "fitshp3")
# AccYear
plt4 <- plot_size(test, "AccYear", "Claim size by AccYear", "shp3", "fitshp3")

grid.arrange(plt1, plt2, plt3, plt4)
```

```
## Warning: Removed 1 rows containing missing values (geom_point).
```



Conclusions

The results of the various plain-vanilla neural network models are as follows.

```
df_cmp
```

```
## # A tibble: 4 x 8
##   model              learn_p2 learn_pp learn_p3 test_p2 test_pp test_p3 avg_size
##   <chr>                 <dbl>    <dbl>    <dbl>   <dbl>   <dbl>   <dbl>    <dbl>
## 1 Null model             5.13    1.02      3.59    5.13   1.02     3.59    12733
## 2 Plain-vanilla p2 ~     4.47    0.952     3.51    4.59   0.965    3.53    12729
## 3 Plain-vanilla pp ~     4.50    0.954     3.52    4.62   0.969    3.54    12954
## 4 Plain-vanilla p3 ~     4.52    0.957     3.52    4.59   0.964    3.53    14216
```

We can draw the following conclusions: * Fitting these networks takes in average ~30 epochs, thus, fitting is very fast here. * We give preference to the gamma model $p = 2$. However, these solutions would need further analysis to perform a thorough model selection, e.g., one can study Tukey{Anscombe plots, dispersion parameters, etc. We refrain from doing so because this is not the main purpose of this notebook. * We remark that the inverse Gaussian model does not have the lowest in-sample loss on $L_{p=3}$. This seems counter-intuitive, but it is caused by the fact that we exercise early stopping. In fact, the inverse Gaussian model is more sensitive in fitting and, typically, this results in an earlier stopping time. Here, it uses less than 30 epochs, whereas the other two cases use more than 30 epochs. In general, the inverse Gaussian model is more difficult to fit.

Exercise: Compare the number of epochs to the number of epochs in the Frnech Motor Third Party Liability tutorials and notebooks.

Theory: LocalGLMnet

The FFN network architectures of the table above often provide good predictive power, however, they are neither easily interpretable nor do they allow for variable selection. I.e., we cannot answer the question whether we should keep, say, variable HoursWorkedPerWeek in the model or not. For this reason we present the LocalGLMnet which allows us to make such decisions.

Below some extracts from the tutorial here.

5.1 Definition of generalized linear models

A classical GLM can be obtained as a special case of a FFN network, namely, if we set depth $d = 0$, we receive GLM regression function

$$\boldsymbol{x} \;\mapsto\; \mu(\boldsymbol{x}) = g^{-1}\left(\beta_0 + \langle \boldsymbol{\beta}, \boldsymbol{x} \rangle\right), \tag{5.1}$$

with regression parameter $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_q)^\top \in \mathbb{R}^q$ and bias $\beta_0 \in \mathbb{R}$. The key idea of LocalGLMnets is to replace the (constant) *regression parameters* $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_q)^\top$ by covariate dependent *regression attentions* $\boldsymbol{\beta}(\boldsymbol{x}) = (\beta_1(\boldsymbol{x}), \ldots, \beta_q(\boldsymbol{x}))^\top$, which are modeled by FFN networks.

5.2 Definition of the LocalGLMnet architecture

We now make regression parameter $\boldsymbol{\beta}$ in (5.1) covariate $\boldsymbol{x}$-dependent.

Assumptions 5.1 (LocalGLMnet) *Choose a FFN network architecture of depth $d \in \mathbb{N}$ with input and output dimensions being equal to $q_0 = q_d = q$ to model the regression attentions*

$$\begin{aligned} \boldsymbol{\beta} : \mathbb{R}^q &\to \mathbb{R}^q \\ \boldsymbol{x} &\mapsto \boldsymbol{\beta}(\boldsymbol{x}) = \boldsymbol{z}^{(d:1)}(\boldsymbol{x}) = \left(\boldsymbol{z}^{(d)} \circ \cdots \circ \boldsymbol{z}^{(1)}\right)(\boldsymbol{x}). \end{aligned} \tag{5.2}$$

The LocalGLMnet is defined by the additive decomposition

$$\boldsymbol{x} \;\mapsto\; \mu(\boldsymbol{x}) = g^{-1}\left(\beta_0 + \langle \boldsymbol{\beta}(\boldsymbol{x}), \boldsymbol{x} \rangle\right). \tag{5.3}$$

Remarks 5.2

- This regression function is called LocalGLMnet, because locally around a given $\boldsymbol{x}$, regression function (5.3) can be understood as a GLM (supposed that $\boldsymbol{\beta}(\boldsymbol{x})$ is more or less constant in this environment).
- $\boldsymbol{\beta}(\boldsymbol{x})$ are called regression attentions because they provide more or less attention to specific values of the components of $\boldsymbol{x}$ in (5.3). This analogy is drawn from the attention layers introduced by Bahdanau et al. [1] and Vaswani et al. [29]. Additive decomposition (5.3) is also similar to the structure of Shapely values [24] (after applying the link function g) which allows, yet, for another interpretation, we refer to Lundberg–Lee [13], Sundararajan–Najmi [26], Sundararajan–Najmi [26] and Lorentzen–Mayer [12] for SHapley Additive exPlanation (SHAP).

Remarks 5.3 (interpretation of the LocalGLMnet regression function (5.3)) The additive decomposition (5.3) allows for an intuitive interpretation. Select one component $1 \leq j \leq q$ and study the terms

$$\beta_j(\boldsymbol{x})x_j. \tag{5.4}$$

(1) If $\beta_j(\boldsymbol{x}) \equiv \beta_j$ ($\neq 0$) is not covariate dependent (and different from zero), we receive a GLM term in x_j.

(2) Condition $\beta_j(\boldsymbol{x}) \equiv 0$ says that the term x_j should not be included.

(3) Property $\beta_j(\boldsymbol{x}) = \beta_j(x_j)$ says that we have a term $\beta_j(x_j)x_j$ that does not interact with other terms. Sensitivities of $\beta_j(\boldsymbol{x})$ in the components of $\boldsymbol{x}$ can be obtained by the gradient

$$\nabla\beta_j(\boldsymbol{x}) = \left(\partial_{x_1}\beta_j(\boldsymbol{x}), \ldots, \partial_{x_q}\beta_j(\boldsymbol{x})\right)^\top \in \mathbb{R}^q. \tag{5.5}$$

The j-th component $\partial_{x_j}\beta_j(\boldsymbol{x})$ of this gradient $\nabla\beta_j(\boldsymbol{x})$ explores whether we have a linear term in x_j, and the components different from j quantify the interactions.

(4) We do not have identifiability as we may still receive

$$\beta_j(\boldsymbol{x})x_j = x_{j'},$$

by learning a regression attention $\beta_j(\boldsymbol{x}) = x_{j'}/x_j$. In our examples, we did not experience these difficulties.

We emphasize in item (2) that $\beta_j(\boldsymbol{x}) \equiv 0$ indicates that the 'term' x_j in the additive decomposition (5.3) should be dropped. The 'covariate' x_j may still need to be kept, as it may play an important role in attention weights $\beta_{j'}(\boldsymbol{x})$ for some $j' \neq j$.

Model(s) 2: LocalGLMnet

We proceed completely analogously to the previous chapter and fit a LocalGLMnet to the accident insurance data. As depth of the LocalGLMnet we choose $d = 4$ having $(q_1, q_2, q_3, q_4) = (20, 15, 10, 16)$ neurons in the FFN layers. Note that $q_4 = q_0 = q = 16$ corresponds to the input dimension. We choose hyperbolic tangent activation function for ϕ_m, $1 \leq m \leq d-1$, the linear function $\phi_d(x) = x$, and log-link for g. The rest is done completely analogously to the previous chapter, namely, we fit this LocalGLMnet architecture using deviance loss functions L_p with $p = 2, 2.5, 3$. The explicit LocalGLMnet architecture is shown in Listing 5 in the appendix, and the results are presented in the tables below.

Common neural network specifications

```
# Size of input for neural networks
q0 <- length(col_features)
qqq <- c(q0, c(20, 15, 10), 1)

sprintf("Neural network with K=3 hidden layer")
```

```
## [1] "Neural network with K=3 hidden layer"
```

```
sprintf("Input feature dimension: q0 = %s", q0)
```

```
## [1] "Input feature dimension: q0 = 16"
```

```
sprintf("Number of hidden neurons first layer: q1 = %s", qqq[2])
```

```
## [1] "Number of hidden neurons first layer: q1 = 20"
```

```
sprintf("Number of hidden neurons second layer: q2 = %s", qqq[3])
```

```
## [1] "Number of hidden neurons second layer: q2 = 15"
```

```
sprintf("Number of hidden neurons third layer: q3 = %s", qqq[4])
```

```
## [1] "Number of hidden neurons third layer: q3 = 10"
```

```
sprintf("Number of hidden neurons third layer: q4 = %s", qqq[1])
```

```
## [1] "Number of hidden neurons third layer: q4 = 16"
```

```
sprintf("Output dimension: %s", qqq[5])
```

```
## [1] "Output dimension: 1"
```

Below we define the response, learning and testing matrix required as input to the `keras` functions.

```
# matrices
YY <- as.matrix(as.numeric(learn$Claim))
XX <- as.matrix(learn[, col_features])
TT <- as.matrix(test[, col_features])
```

```
# neural network structure
Design  <- layer_input(shape = c(qqq[1]), dtype = 'float32', name = 'design')

Attention <- Design %>%
    layer_dense(units=qqq[2], activation='tanh', name='layer1') %>%
    layer_dense(units=qqq[3], activation='tanh', name='layer2') %>%
    layer_dense(units=qqq[4], activation='tanh', name='layer3') %>%
    layer_dense(units=qqq[1], activation='linear', name='attention')

Output <- list(Design, Attention) %>% layer_dot(name='LocalGLM', axes=1) %>%
    layer_dense(
      units=1, activation='exponential', name='output',
      weights=list(array(0, dim=c(1,1)), array(log_size_hom, dim=c(1)))
    )
```

LocalGLMnet Gamma (p=2)

Definition

```
model_lgn_p2 <- keras_model(inputs = list(Design), outputs = c(Output))
```

Compilation

```
model_lgn_p2 %>% compile(
    loss = k_gamma_loss,
    optimizer = 'nadam'
)
summary(model_lgn_p2)
```

```
## Model: "model_3"
## ________________________________________________________________________________
## Layer (type)                Output Shape      Param #  Connected to
## ================================================================================
```

```
## design (InputLayer)       [(None, 16)]      0
## ________________________________________________________________________________
## layer1 (Dense)            (None, 20)        340       design[0][0]
## ________________________________________________________________________________
## layer2 (Dense)            (None, 15)        315       layer1[0][0]
## ________________________________________________________________________________
## layer3 (Dense)            (None, 10)        160       layer2[0][0]
## ________________________________________________________________________________
## attention (Dense)         (None, 16)        176       layer3[0][0]
## ________________________________________________________________________________
## LocalGLM (Dot)            (None, 1)         0         design[0][0]
##                                                       attention[0][0]
## ________________________________________________________________________________
## output (Dense)            (None, 1)         2         LocalGLM[0][0]
## ================================================================================
## Total params: 993
## Trainable params: 993
## Non-trainable params: 0
## ________________________________________________________________________________
```

Fitting

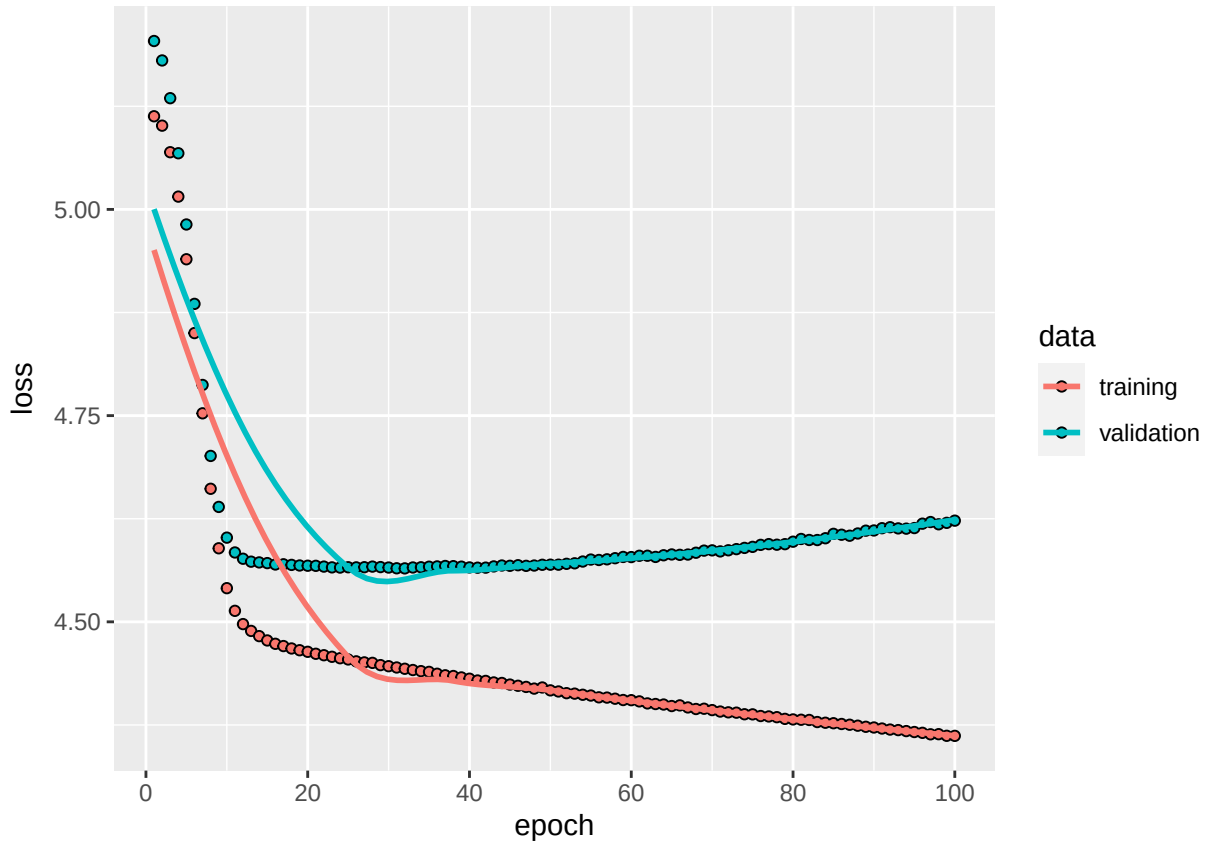
```
# set hyperparameters
epochs <- 100
batch_size <- 5000
validation_split <- 0.2 # set to >0 to see train/validation loss in plot(fit)
verbose <- 1
```

```
# store and use only the best model
cp_path <- paste("./Networks/model_lgn_p2")

cp_callback <- callback_model_checkpoint(
    filepath = cp_path,
    monitor = "val_loss",
    save_weights_only = TRUE,
    save_best_only = TRUE,
    verbose = 0
)
```

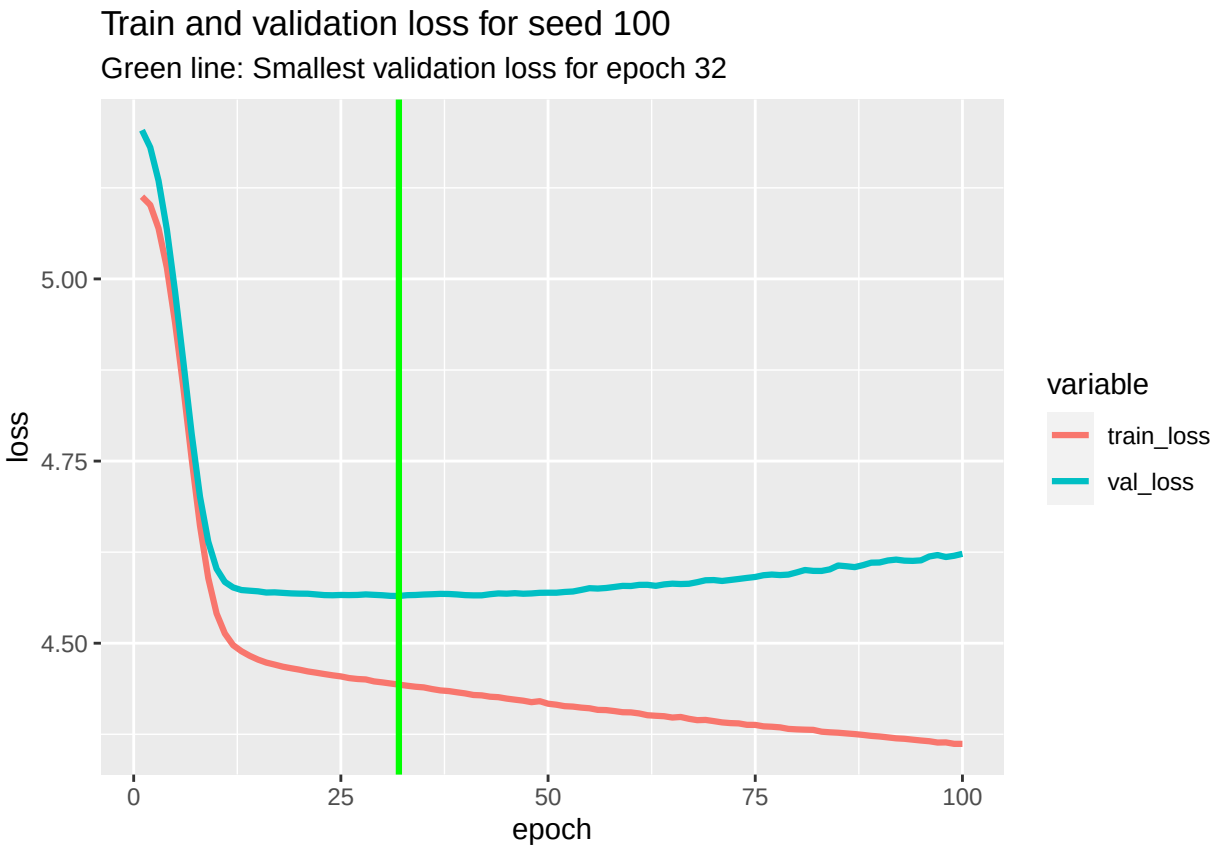
```
fit_lgn_p2 <- model_lgn_p2 %>% fit(
    list(XX), list(YY),
    validation_split = validation_split,
    epochs = epochs,
    batch_size = batch_size,
    callbacks = list(cp_callback),
    verbose = verbose
)
```

```
plot(fit_lgn_p2)
```

```
## `geom_smooth()` using formula 'y ~ x'
```




```
keras_plot_loss_min(fit_lgn_p2, seed)
```




```
load_model_weights_hdf5(model_lgn_p2, cp_path)
```

Validation

```
# calculating the predictions
learn$fitlgnp2 <- as.vector(model_lgn_p2 %>% predict(list(XX)))
test$fitlgnp2 <- as.vector(model_lgn_p2 %>% predict(list(TT)))

# average in-sample and out-of-sample losses (in 10^(0))
sprintf("Gamma deviance shallow network (train): %s", round(gamma_loss(learn$Claim, learn$fitlgnp2), 4))
```

```
## [1] "Gamma deviance shallow network (train): 4.4644"
```

```
sprintf("Gamma deviance shallow network (test): %s", round(gamma_loss(test$Claim, test$fitlgnp2), 4))
```

```
## [1] "Gamma deviance shallow network (test): 4.5854"
```

```
# average claims size
sprintf("Average size (test): %s", round(mean(test$fitlgnp2), 1))
```

```
## [1] "Average size (test): 12729.4"
```

```
df_cmp %<>% bind_rows(
  data.frame(model = "LocalGLMnet p2 (gamma)",
             learn_p2 = round(gamma_loss(learn$Claim, learn$fitlgnp2), 4),
             learn_pp = round(p_loss(learn$Claim, learn$fitlgnp2, p) * 10, 4),
             learn_p3 = round(ig_loss(learn$Claim, learn$fitlgnp2) * 1000, 4),
             test_p2 = round(gamma_loss(test$Claim, test$fitlgnp2), 4),
             test_pp = round(p_loss(test$Claim, test$fitlgnp2, p) * 10, 4),
```

```
            test_p3 = round(ig_loss(test$Claim, test$fitlgnp2) * 1000, 4),
            avg_size = round(mean(test$fitlgnp2), 0)
  ))
df_cmp
```

```
## # A tibble: 5 x 8
##   model              learn_p2 learn_pp learn_p3 test_p2 test_pp test_p3 avg_size
##   <chr>                 <dbl>    <dbl>    <dbl>   <dbl>   <dbl>   <dbl>    <dbl>
## 1 Null model             5.13    1.02      3.59    5.13   1.02     3.59    12733
## 2 Plain-vanilla p2 ~     4.47    0.952     3.51    4.59   0.965    3.53    12729
## 3 Plain-vanilla pp ~     4.50    0.954     3.52    4.62   0.969    3.54    12954
## 4 Plain-vanilla p3 ~     4.52    0.957     3.52    4.59   0.964    3.53    14216
## 5 LocalGLMnet p2 (g~     4.46    0.950     3.51    4.59   0.966    3.53    12729
```

Calibration

```
# Age
plt1 <- plot_size(test, "Age", "Claim size by Age", "lgnp2", "fitlgnp2")
# Gender
plt2 <- plot_size(test, "Gender", "Claim size by Gender", "lgnp2", "fitlgnp2")
# AccMonth
plt3 <- plot_size(test, "AccMonth", "Claim size by AccMonth", "lgnp2", "fitlgnp2")
# AccYear
plt4 <- plot_size(test, "AccYear", "Claim size by AccYear", "lgnp2", "fitlgnp2")

grid.arrange(plt1, plt2, plt3, plt4)
```

```
## Warning: Removed 1 rows containing missing values (geom_point).
```

LocalGlmnet $2 < p < 3$

Definition

```
model_lgn_pp <- keras_model(inputs = list(Design), outputs = c(Output))
```

Compilation

```
model_lgn_pp %>% compile(
    loss = k_p_loss,
    optimizer = 'nadam'
)
summary(model_lgn_pp)
```

```
## Model: "model_4"
## ____________________________________________________________________________
## Layer (type)                Output Shape      Param #  Connected to
## ============================================================================
## design (InputLayer)         [(None, 16)]      0
## ____________________________________________________________________________
## layer1 (Dense)              (None, 20)        340      design[0][0]
## ____________________________________________________________________________
## layer2 (Dense)              (None, 15)        315      layer1[0][0]
## ____________________________________________________________________________
## layer3 (Dense)              (None, 10)        160      layer2[0][0]
## ____________________________________________________________________________
## attention (Dense)           (None, 16)        176      layer3[0][0]
```

```
## ____________________________________________________________________________
## LocalGLM (Dot)            (None, 1)         0        design[0][0]
##                                                      attention[0][0]
## ____________________________________________________________________________
## output (Dense)            (None, 1)         2        LocalGLM[0][0]
## ============================================================================
## Total params: 993
## Trainable params: 993
## Non-trainable params: 0
## ____________________________________________________________________________
```

Fitting

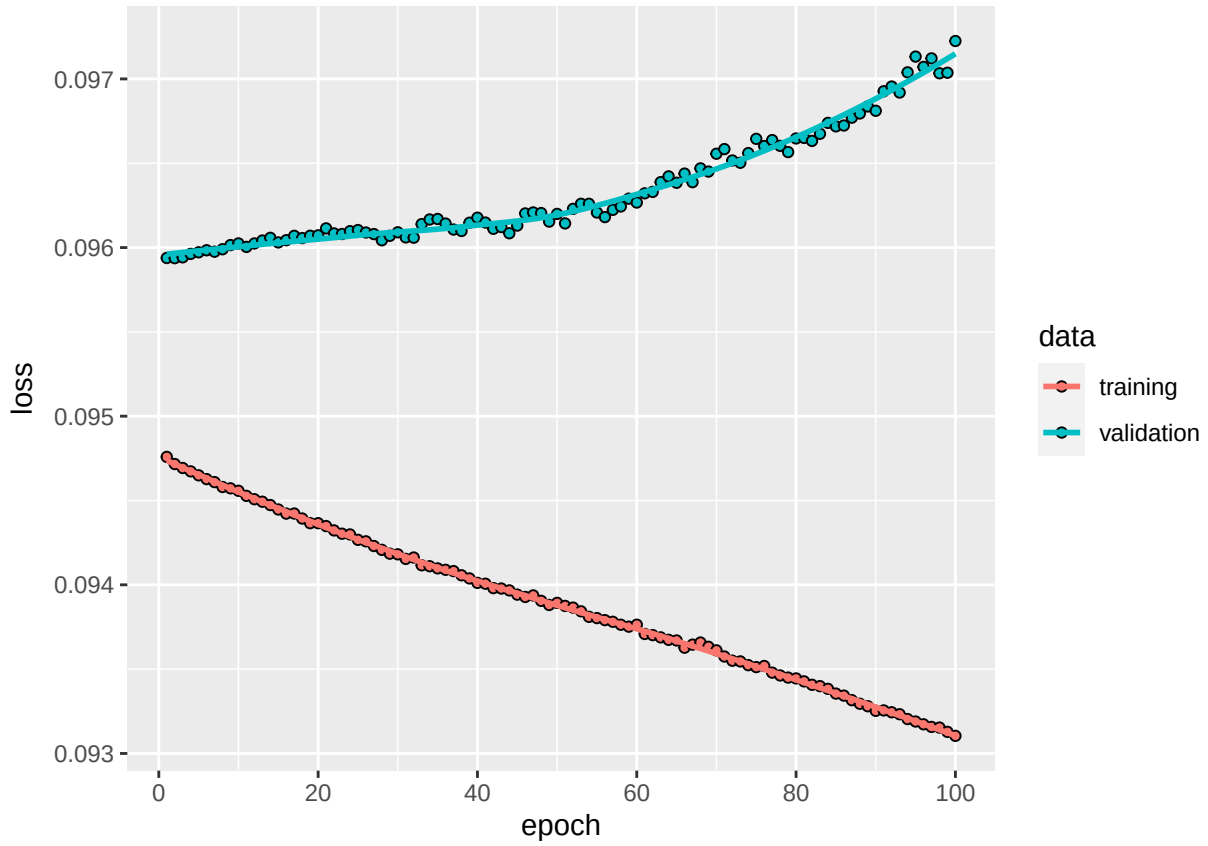
```
# set hyperparameters
epochs <- 100
batch_size <- 5000
validation_split <- 0.2 # set to >0 to see train/validation loss in plot(fit)
verbose <- 1
```

```
# store and use only the best model
cp_path <- paste("./Networks/model_lgn_pp")

cp_callback <- callback_model_checkpoint(
    filepath = cp_path,
    monitor = "val_loss",
    save_weights_only = TRUE,
    save_best_only = TRUE,
    verbose = 0
)
```

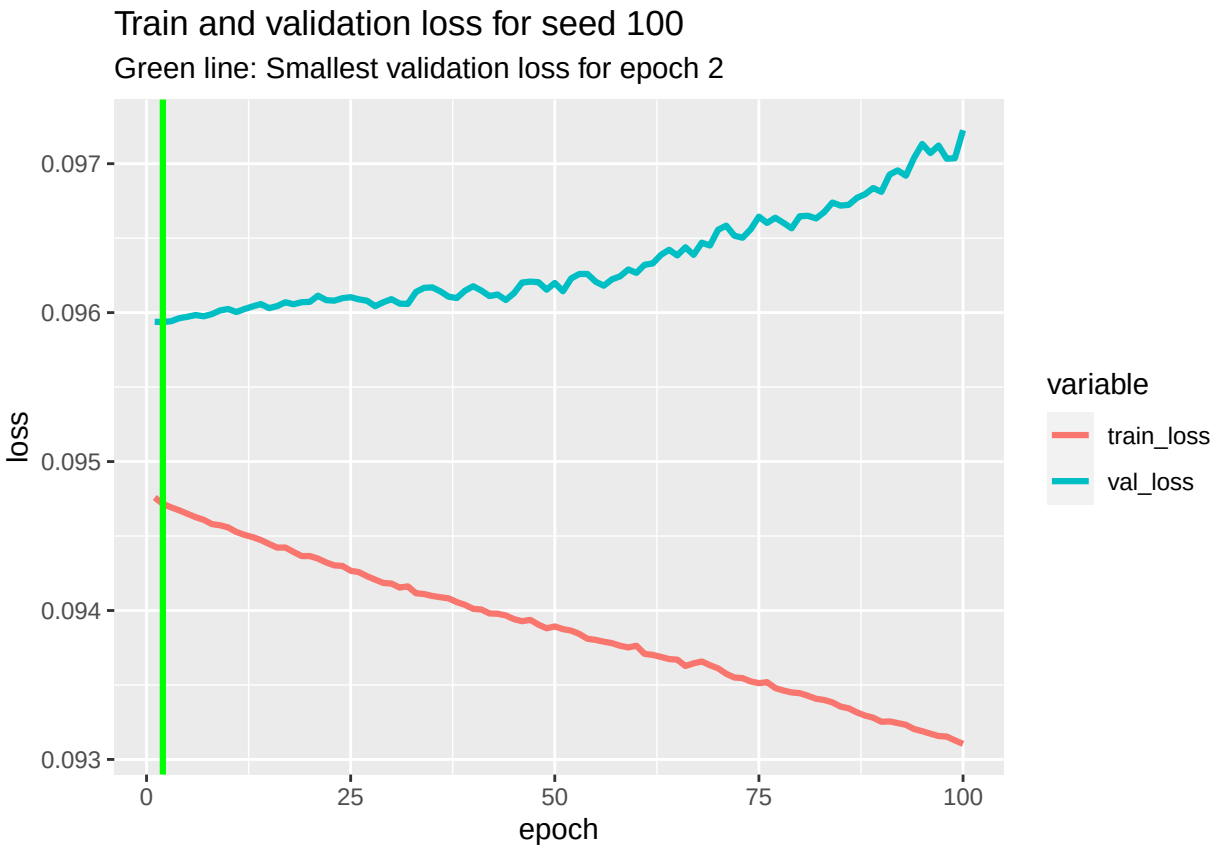
```
fit_lgn_pp <- model_lgn_pp %>% fit(
    list(XX), list(YY),
    validation_split = validation_split,
    epochs = epochs,
    batch_size = batch_size,
    callbacks = list(cp_callback),
    verbose = verbose
)
```

```
plot(fit_lgn_pp)
```

```
## `geom_smooth()` using formula 'y ~ x'
```




```
keras_plot_loss_min(fit_lgn_pp, seed)
```




```
load_model_weights_hdf5(model_lgn_pp, cp_path)
```

Validation

```
# calculating the predictions
learn$fitlgnpp <- as.vector(model_pp %>% predict(list(XX)))
test$fitlgnpp <- as.vector(model_pp %>% predict(list(TT)))

# average in-sample and out-of-sample losses (in 10^(0))
sprintf("p-loss deviance shallow network (train): %s", round(p_loss(learn$Claim, learn$fitlgnpp, p), 4))
```

```
## [1] "p-loss deviance shallow network (train): 0.0954"
```

```
sprintf("p-loss deviance shallow network (test): %s", round(p_loss(test$Claim, test$fitlgnpp, p), 4))
```

```
## [1] "p-loss deviance shallow network (test): 0.0969"
```

```
# average claims size
sprintf("Average size (test): %s", round(mean(test$fitlgnpp), 1))
```

```
## [1] "Average size (test): 12953.8"
```

```
df_cmp %<>% bind_rows(
  data.frame(model = "LocalGLMnet pp (p=2.5)",
             learn_p2 = round(gamma_loss(learn$Claim, learn$fitlgnpp), 4),
             learn_pp = round(p_loss(learn$Claim, learn$fitlgnpp, p) * 10, 4),
             learn_p3 = round(ig_loss(learn$Claim, learn$fitlgnpp) * 1000, 4),
             test_p2 = round(gamma_loss(test$Claim, test$fitlgnpp), 4),
             test_pp = round(p_loss(test$Claim, test$fitlgnpp, p) * 10, 4),
```

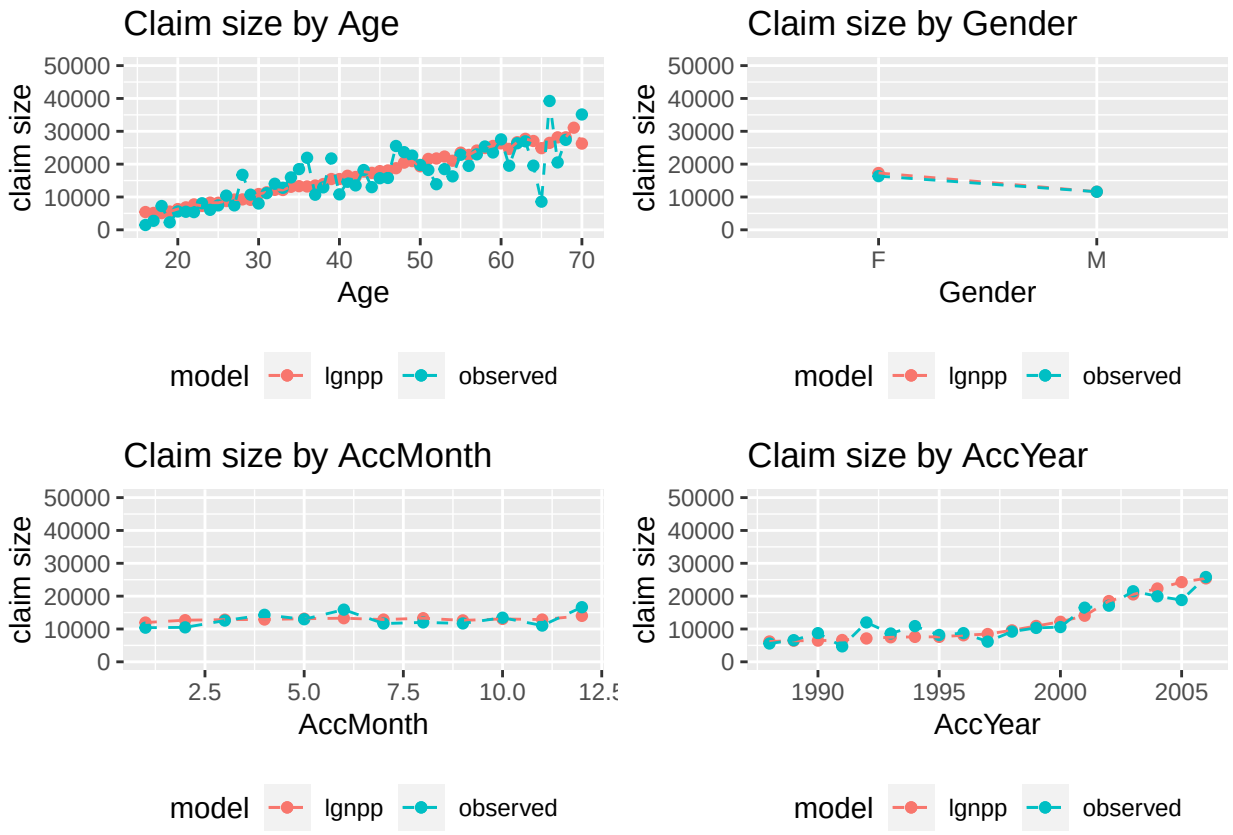
```
            test_p3 = round(ig_loss(test$Claim, test$fitlgnpp) * 1000, 4),
            avg_size = round(mean(test$fitlgnpp), 0)
  ))
df_cmp
```

```
## # A tibble: 6 x 8
##   model              learn_p2 learn_pp learn_p3 test_p2 test_pp test_p3 avg_size
##   <chr>                 <dbl>    <dbl>    <dbl>   <dbl>   <dbl>   <dbl>    <dbl>
## 1 Null model             5.13    1.02      3.59    5.13   1.02     3.59    12733
## 2 Plain-vanilla p2 ~     4.47    0.952     3.51    4.59   0.965    3.53    12729
## 3 Plain-vanilla pp ~     4.50    0.954     3.52    4.62   0.969    3.54    12954
## 4 Plain-vanilla p3 ~     4.52    0.957     3.52    4.59   0.964    3.53    14216
## 5 LocalGLMnet p2 (g~     4.46    0.950     3.51    4.59   0.966    3.53    12729
## 6 LocalGLMnet pp (p~     4.50    0.954     3.52    4.62   0.969    3.54    12954
```

Calibration

```
# Age
plt1 <- plot_size(test, "Age", "Claim size by Age", "lgnpp", "fitlgnpp")
# Gender
plt2 <- plot_size(test, "Gender", "Claim size by Gender", "lgnpp", "fitlgnpp")
# AccMonth
plt3 <- plot_size(test, "AccMonth", "Claim size by AccMonth", "lgnpp", "fitlgnpp")
# AccYear
plt4 <- plot_size(test, "AccYear", "Claim size by AccYear", "lgnpp", "fitlgnpp")

grid.arrange(plt1, plt2, plt3, plt4)
```

```
## Warning: Removed 1 rows containing missing values (geom_point).
```



LocalGLMnet Inverse Gaussian (p=3)

Definition

```
model_lgn_p3 <- keras_model(inputs = list(Design), outputs = c(Output))
```

Compilation

```
model_lgn_p3 %>% compile(
    loss = k_ig_loss,
    optimizer = 'nadam'
)
summary(model_lgn_p3)
```

```
## Model: "model_5"
## ________________________________________________________________________________
## Layer (type)                Output Shape      Param #  Connected to
## ================================================================================
## design (InputLayer)         [(None, 16)]      0
## ________________________________________________________________________________
## layer1 (Dense)              (None, 20)        340      design[0][0]
## ________________________________________________________________________________
## layer2 (Dense)              (None, 15)        315      layer1[0][0]
## ________________________________________________________________________________
## layer3 (Dense)              (None, 10)        160      layer2[0][0]
## ________________________________________________________________________________
## attention (Dense)           (None, 16)        176      layer3[0][0]
```

```
## ____________________________________________________________________________
## LocalGLM (Dot)            (None, 1)         0        design[0][0]
##                                                      attention[0][0]
## ____________________________________________________________________________
## output (Dense)            (None, 1)         2        LocalGLM[0][0]
## ============================================================================
## Total params: 993
## Trainable params: 993
## Non-trainable params: 0
## ____________________________________________________________________________
```

Fitting

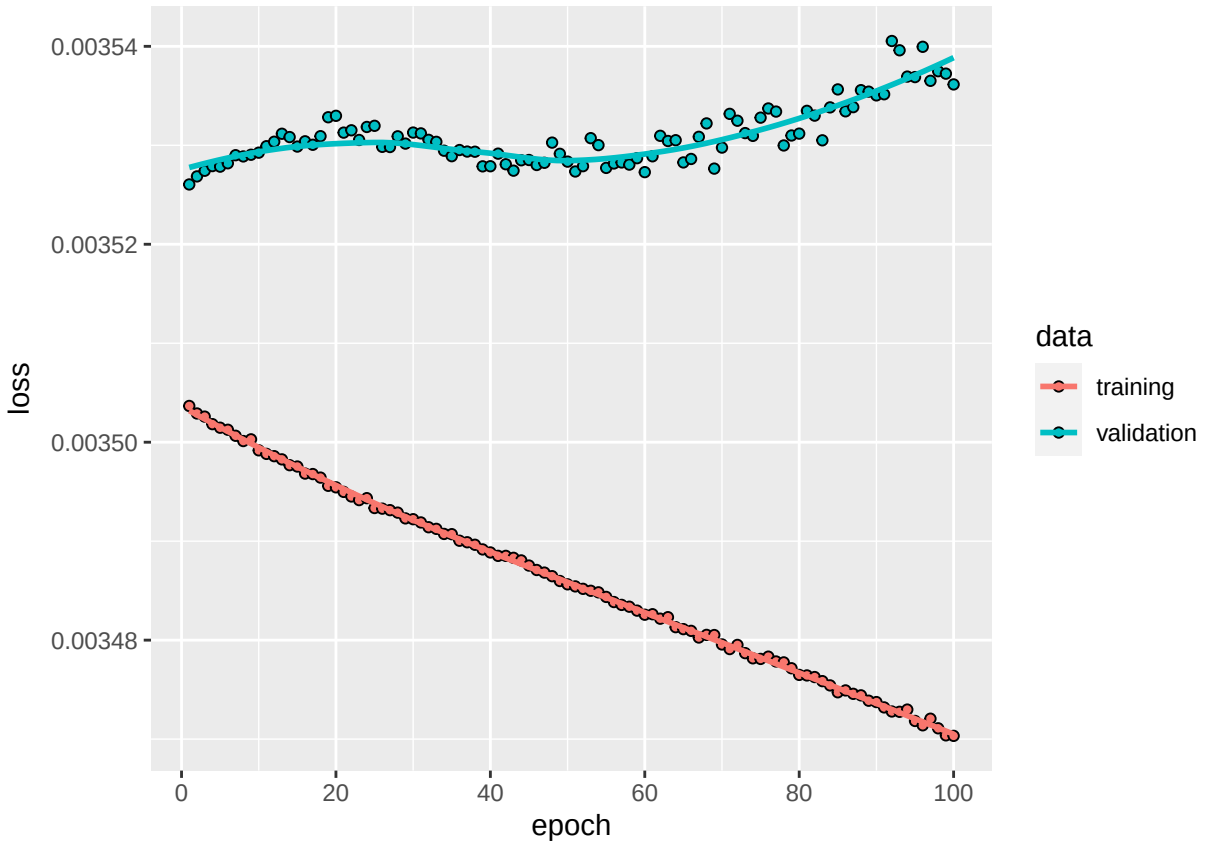
```
# set hyperparameters
epochs <- 100
batch_size <- 5000
validation_split <- 0.2 # set to >0 to see train/validation loss in plot(fit)
verbose <- 1
```

```
# store and use only the best model
cp_path <- paste("./Networks/model_lgn_p3")

cp_callback <- callback_model_checkpoint(
    filepath = cp_path,
    monitor = "val_loss",
    save_weights_only = TRUE,
    save_best_only = TRUE,
    verbose = 0
)
```

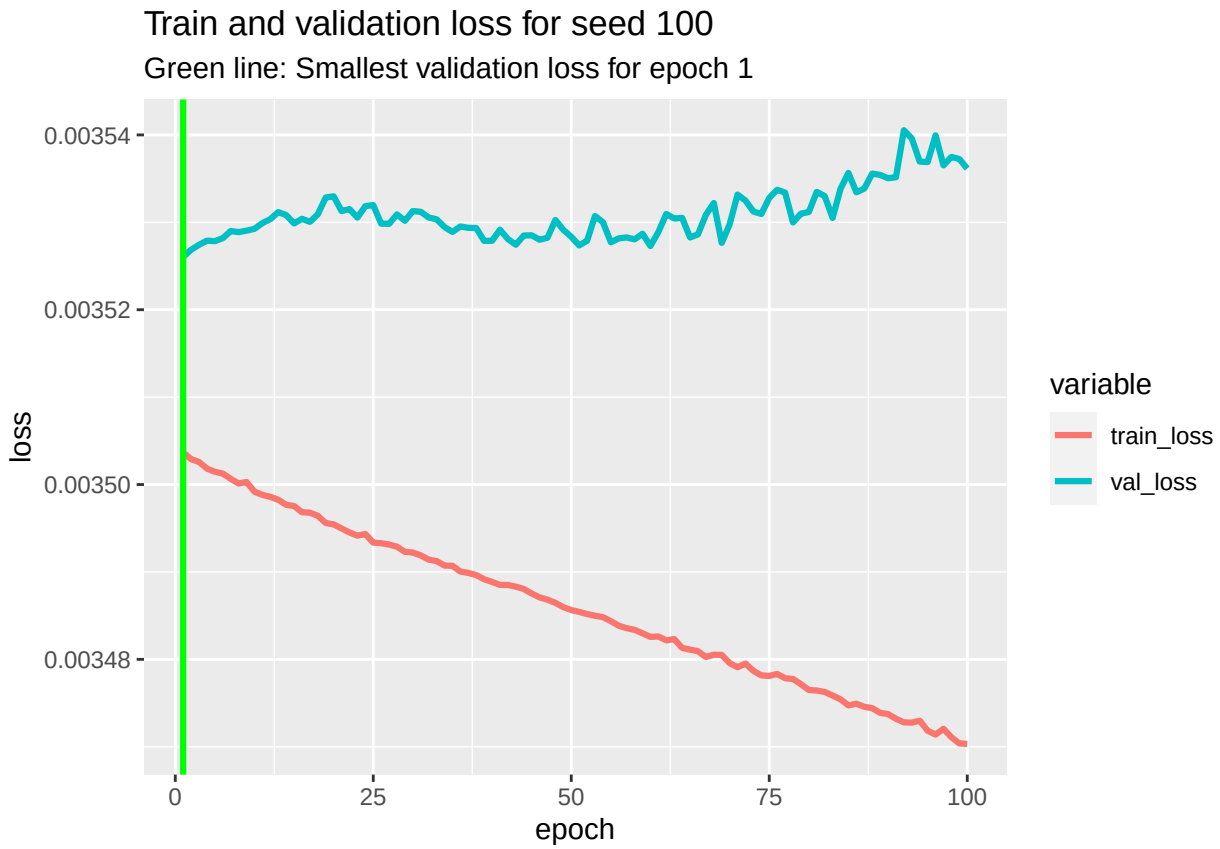
```
fit_lgn_p3 <- model_lgn_p3 %>% fit(
    list(XX), list(YY),
    validation_split = validation_split,
    epochs = epochs,
    batch_size = batch_size,
    callbacks = list(cp_callback),
    verbose = verbose
)
```

```
plot(fit_lgn_p3)
```

```
## `geom_smooth()` using formula 'y ~ x'
```




```
keras_plot_loss_min(fit_lgn_p3, seed)
```




```
load_model_weights_hdf5(model_lgn_p3, cp_path)
```

Validation

```
# calculating the predictions
learn$fitlgnp3 <- as.vector(model_lgn_p3 %>% predict(list(XX)))
test$fitlgnp3 <- as.vector(model_lgn_p3 %>% predict(list(TT)))

# average in-sample and out-of-sample losses (in 10^(0))
sprintf("IG deviance shallow network (train): %s", round(ig_loss(learn$Claim, learn$fitlgnp3), 4))
```

```
## [1] "IG deviance shallow network (train): 0.0035"
```

```
sprintf("IG deviance shallow network (test): %s", round(ig_loss(test$Claim, test$fitlgnp3), 4))
```

```
## [1] "IG deviance shallow network (test): 0.0035"
```

```
# average claims size
sprintf("Average size (test): %s", round(mean(test$fitlgnp3), 1))
```

```
## [1] "Average size (test): 12854.5"
```

```
df_cmp %<>% bind_rows(
  data.frame(model = "LocalGLMnet p3 (inverse gaussian)",
             learn_p2 = round(gamma_loss(learn$Claim, learn$fitlgnp3), 4),
             learn_pp = round(p_loss(learn$Claim, learn$fitlgnp3, p) * 10, 4),
             learn_p3 = round(ig_loss(learn$Claim, learn$fitlgnp3) * 1000, 4),
             test_p2 = round(gamma_loss(test$Claim, test$fitlgnp3), 4),
             test_pp = round(p_loss(test$Claim, test$fitlgnp3, p) * 10, 4),
```

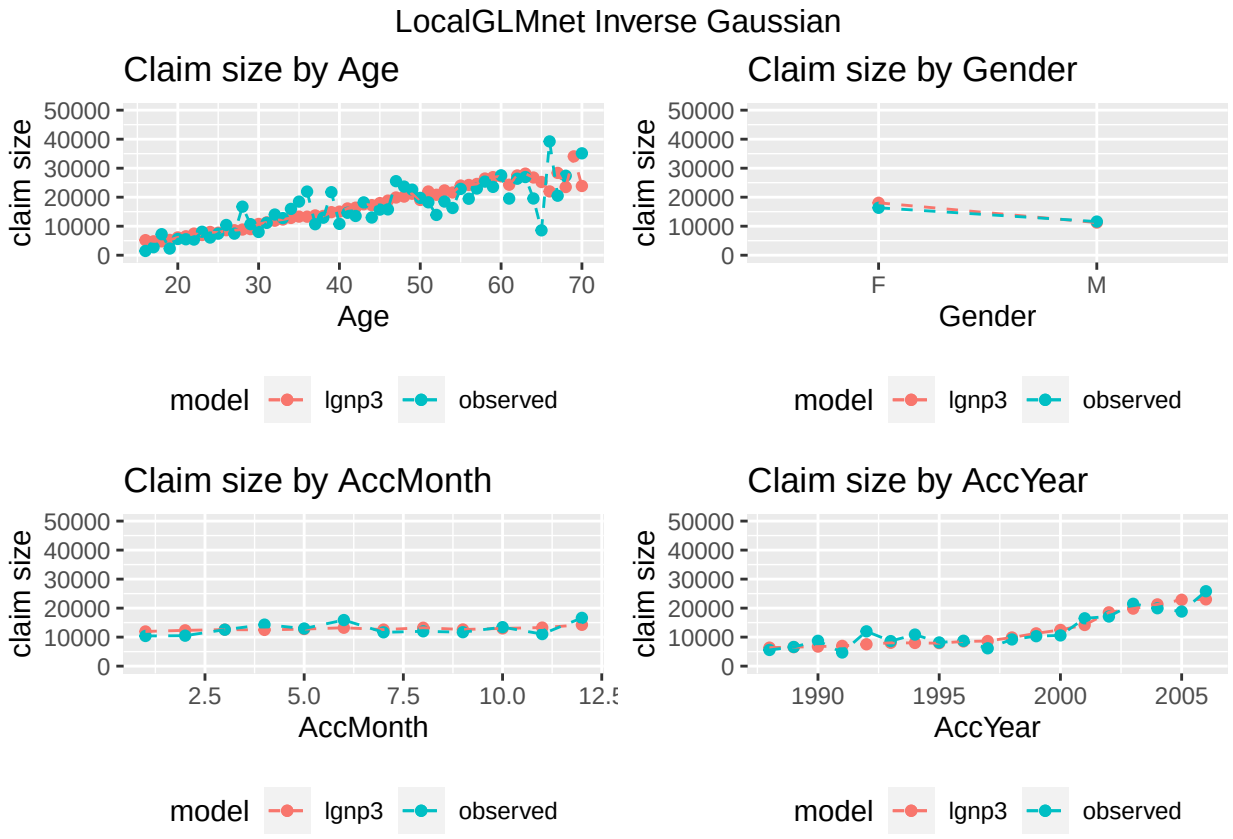
```
            test_p3 = round(ig_loss(test$Claim, test$fitlgnp3) * 1000, 4),
            avg_size = round(mean(test$fitlgnp3), 0)
  ))
df_cmp
```

```
## # A tibble: 7 x 8
##   model              learn_p2 learn_pp learn_p3 test_p2 test_pp test_p3 avg_size
##   <chr>                 <dbl>    <dbl>    <dbl>   <dbl>   <dbl>   <dbl>    <dbl>
## 1 Null model             5.13    1.02      3.59    5.13   1.02     3.59    12733
## 2 Plain-vanilla p2 ~     4.47    0.952     3.51    4.59   0.965    3.53    12729
## 3 Plain-vanilla pp ~     4.50    0.954     3.52    4.62   0.969    3.54    12954
## 4 Plain-vanilla p3 ~     4.52    0.957     3.52    4.59   0.964    3.53    14216
## 5 LocalGLMnet p2 (g~     4.46    0.950     3.51    4.59   0.966    3.53    12729
## 6 LocalGLMnet pp (p~     4.50    0.954     3.52    4.62   0.969    3.54    12954
## 7 LocalGLMnet p3 (i~     4.46    0.949     3.51    4.60   0.968    3.54    12855
```

Calibration

```
# Age
plt1 <- plot_size(test, "Age", "Claim size by Age", "lgnp3", "fitlgnp3")
# Gender
plt2 <- plot_size(test, "Gender", "Claim size by Gender", "lgnp3", "fitlgnp3")
# AccMonth
plt3 <- plot_size(test, "AccMonth", "Claim size by AccMonth", "lgnp3", "fitlgnp3")
# AccYear
plt4 <- plot_size(test, "AccYear", "Claim size by AccYear", "lgnp3", "fitlgnp3")

grid.arrange(plt1, plt2, plt3, plt4, top="LocalGLMnet Inverse Gaussian")
```

```
## Warning: Removed 1 rows containing missing values (geom_point).
```



Results of plain-vanilla and LocalGLMnet

The results of the plain-vanilla and LocalGLNnet models are as follows.

```
df_cmp
```

```
## # A tibble: 7 x 8
##   model              learn_p2 learn_pp learn_p3 test_p2 test_pp test_p3 avg_size
##   <chr>                 <dbl>    <dbl>    <dbl>   <dbl>   <dbl>   <dbl>    <dbl>
## 1 Null model             5.13    1.02      3.59    5.13   1.02     3.59    12733
## 2 Plain-vanilla p2 ~     4.47    0.952     3.51    4.59   0.965    3.53    12729
## 3 Plain-vanilla pp ~     4.50    0.954     3.52    4.62   0.969    3.54    12954
## 4 Plain-vanilla p3 ~     4.52    0.957     3.52    4.59   0.964    3.53    14216
## 5 LocalGLMnet p2 (g~     4.46    0.950     3.51    4.59   0.966    3.53    12729
## 6 LocalGLMnet pp (p~     4.50    0.954     3.52    4.62   0.969    3.54    12954
## 7 LocalGLMnet p3 (i~     4.46    0.949     3.51    4.60   0.968    3.54    12855
```

We can draw the following conclusions: * We again prefer the gamma model over the other power variance parameter models. * Moreover, for this particular data set the LocalGLMnet outperforms the deep FFN network. However, this is not the crucial point of introducing the LocalGLMnet, but the LocalGLMnet leads to interpretable predictions and it allows for variable selection as we are going to demonstrate in the next sections.

LocalGLMnet: Interpretations

In the following chapters, we are going to explore the various interpretations of the regression functions, which is the beauty and main benefit of LocalGLMnet!

Remarks 5.3 (interpretation of the LocalGLMnet regression function (5.3)) The additive decomposition (5.3) allows for an intuitive interpretation. Select one component $1 \leq j \leq q$ and study the terms

$$\beta_j(\boldsymbol{x})x_j. \tag{5.4}$$

(1) If $\beta_j(\boldsymbol{x}) \equiv \beta_j$ ($\neq 0$) is not covariate dependent (and different from zero), we receive a GLM term in x_j.

(2) Condition $\beta_j(\boldsymbol{x}) \equiv 0$ says that the term x_j should not be included.

(3) Property $\beta_j(\boldsymbol{x}) = \beta_j(x_j)$ says that we have a term $\beta_j(x_j)x_j$ that does not interact with other terms. Sensitivities of $\beta_j(\boldsymbol{x})$ in the components of $\boldsymbol{x}$ can be obtained by the gradient

$$\nabla\beta_j(\boldsymbol{x}) = \left(\partial_{x_1}\beta_j(\boldsymbol{x}), \ldots, \partial_{x_q}\beta_j(\boldsymbol{x})\right)^\top \in \mathbb{R}^q. \tag{5.5}$$

The j-th component $\partial_{x_j}\beta_j(\boldsymbol{x})$ of this gradient $\nabla\beta_j(\boldsymbol{x})$ explores whether we have a linear term in x_j, and the components different from j quantify the interactions.

(4) We do not have identifiability as we may still receive

$$\beta_j(\boldsymbol{x})x_j = x_{j'},$$

by learning a regression attention $\beta_j(\boldsymbol{x}) = x_{j'}/x_j$. In our examples, we did not experience these difficulties.

We emphasize in item (2) that $\beta_j(\boldsymbol{x}) \equiv 0$ indicates that the 'term' x_j in the additive decomposition (5.3) should be dropped. The 'covariate' x_j may still need to be kept, as it may play an important role in attention weights $\beta_{j'}(\boldsymbol{x})$ for some $j' \neq j$.

LocalGLMnet: Variable selection and variable importance

We are going to examine points (1) and (2) from the previous extract.

5.3.2 Variable selection and variable importance

The purpose of this section is to understand whether we need to include all additive terms into the regression function (5.3). We would like to explore this question by classical statistical testing. That is, we consider the null hypothesis $H_0 : \beta_j(\boldsymbol{x}) = 0$ for a given $1 \leq j \leq q$. If the estimates $\widehat{\beta}_j(\boldsymbol{x})$ differ too much from zero we reject this null hypothesis H_0. In classical statistics, one performs a Wald test or a likelihood ratio test (LRT) to answer such questions. Both tests are based on an asymptotic theory for MLE. Unfortunately, there is no similar asymptotic theory available for FNN networks. Therefore, we need to explore an empirical test.
In view of Remarks 5.3 (2) we need to understand whether for some components j we have $\beta_j(\boldsymbol{x}) \approx 0$ to analyze the null hypothesis H_0 for that j. In order to make this decision we extend the original covariates $\boldsymbol{x}_i$ by additional information that does *not* belong to our data. We therefore expand the original covariates $\boldsymbol{x}$ to $\boldsymbol{x}^+ = (\boldsymbol{x}^\top, x_{q+1}, x_{q+2})^\top \in \mathbb{R}^{q+2}$ and consider the extended LocalGLMnet regression function

$$\boldsymbol{x}^+ \;\mapsto\; \mu(\boldsymbol{x}^+) = g^{-1}\left(\beta_0 + \langle \boldsymbol{\beta}(\boldsymbol{x}^+), \boldsymbol{x}^+ \rangle\right). \tag{5.6}$$

These two additional components x_{q+1}, x_{q+2} will be taken *purely at random* and, thus, they cannot contain any systematic effects explaining response Y. For this reason we would like to see that the network finds estimated regression attentions $\widehat{\beta}_{q+1}(\boldsymbol{x}^+) \approx 0$ and $\widehat{\beta}_{q+2}(\boldsymbol{x}^+) \approx 0$ for these two additional terms. Network estimation will not provide them being exactly equal to zero as there is some noise in the data that also lets these regression attentions fluctuate around zero. The magnitude of these fluctuations will identify regression attention estimates $\widehat{\beta}_j(\boldsymbol{x}^+)$, $1 \leq j \leq q$, that are not sufficiently different from zero to justify inclusion in (5.6) and (5.3), respectively.
These two additional components x_{q+1} and x_{q+2} should be centered and have unit variance to live on the same scale as the other components in $\boldsymbol{x}$. For claims $1 \leq i \leq n$ we choose i.i.d. random variables $x_{i,q+1} \sim \text{Uniform}[-\sqrt{3}, \sqrt{3}]$, and independent i.i.d. random variables $x_{i,q+2} \sim \mathcal{N}(0,1)$. These variables are standardized, and we choose two different components to see whether the distributional choice influences our decision. We fit this model to the data using the gamma deviance loss L_2, the results are given in Table 5.

Data Preparation

Needs to be done as the two random features are added.

```
# used/selected features
col_features <- c("AgeNN","GenderNN","DependentChildrenNN","DependentsOtherNN",
                  "WeeklyPayNN","PartTimeFullTimeNN","HoursWorkedPerWeekNN",
                  "DaysWorkedPerWeekNN","AccYearNN","AccMonthNN","AccWeekdayNN",
                  "AccTimeNN","RepDelayNN","RandUN","RandNN","Marital1","Marital2","Marital3")
col_names <- c("Age","Gender","DependentChildren","DependentsOther","WeeklyPay",
               "PartTimeFullTime","HoursWorkedPerWeek","DaysWorkedPerWeek",
               "AccYear","AccMonth","AccWeekday","AccTime","RepDelay","RandUN",
               "RandNN","Marital1","Marital2","Marital3")
```

```
# Size of input for neural networks
q0 <- length(col_features)
```

```r
qqq <- c(q0, c(20, 15, 10), 1)

sprintf("Neural network with K=3 hidden layer")
```

```
## [1] "Neural network with K=3 hidden layer"
```

```r
sprintf("Input feature dimension: q0 = %s", q0)
```

```
## [1] "Input feature dimension: q0 = 18"
```

```r
sprintf("Number of hidden neurons first layer: q1 = %s", qqq[2])
```

```
## [1] "Number of hidden neurons first layer: q1 = 20"
```

```r
sprintf("Number of hidden neurons second layer: q2 = %s", qqq[3])
```

```
## [1] "Number of hidden neurons second layer: q2 = 15"
```

```r
sprintf("Number of hidden neurons third layer: q3 = %s", qqq[4])
```

```
## [1] "Number of hidden neurons third layer: q3 = 10"
```

```r
sprintf("Number of hidden neurons third layer: q4 = %s", qqq[1])
```

```
## [1] "Number of hidden neurons third layer: q4 = 18"
```

```r
sprintf("Output dimension: %s", qqq[5])
```

```
## [1] "Output dimension: 1"
```

```r
# matrices
YY <- as.matrix(as.numeric(learn$Claim))
XX <- as.matrix(learn[, col_features])
TT <- as.matrix(test[, col_features])
```

As $p = 2$ is our preferred model, we need to fit it including the two random variables.

Fit LocalGLMnet Gamma (p=2)

Definition

```r
# neural network structure
Design <- layer_input(shape = c(qqq[1]), dtype = 'float32', name = 'design')

Attention <- Design %>%
    layer_dense(units=qqq[2], activation='tanh', name='layer1') %>%
    layer_dense(units=qqq[3], activation='tanh', name='layer2') %>%
    layer_dense(units=qqq[4], activation='tanh', name='layer3') %>%
    layer_dense(units=qqq[1], activation='linear', name='attention')

Output <- list(Design, Attention) %>% layer_dot(name='LocalGLM', axes=1) %>%
    layer_dense(
      units=1, activation='exponential', name='output',
      weights=list(array(0, dim=c(1,1)), array(log_size_hom, dim=c(1))))
```

```r
model_var <- keras_model(inputs = list(Design), outputs = c(Output))
```

Compilation

```
model_var %>% compile(
    loss = k_gamma_loss,
    optimizer = 'nadam'
)
summary(model_var)
```

```
## Model: "model_6"
## ____________________________________________________________________________
## Layer (type)               Output Shape      Param #  Connected to
## ============================================================================
## design (InputLayer)        [(None, 18)]      0
## ____________________________________________________________________________
## layer1 (Dense)             (None, 20)        380      design[0][0]
## ____________________________________________________________________________
## layer2 (Dense)             (None, 15)        315      layer1[0][0]
## ____________________________________________________________________________
## layer3 (Dense)             (None, 10)        160      layer2[0][0]
## ____________________________________________________________________________
## attention (Dense)          (None, 18)        198      layer3[0][0]
## ____________________________________________________________________________
## LocalGLM (Dot)             (None, 1)         0        design[0][0]
##                                                       attention[0][0]
## ____________________________________________________________________________
## output (Dense)             (None, 1)         2        LocalGLM[0][0]
## ============================================================================
## Total params: 1,055
## Trainable params: 1,055
## Non-trainable params: 0
## ____________________________________________________________________________
```

Fitting

```
# set hyperparameters
epochs <- 100
batch_size <- 5000
validation_split <- 0.2 # set to >0 to see train/validation loss in plot(fit)
verbose <- 1
```

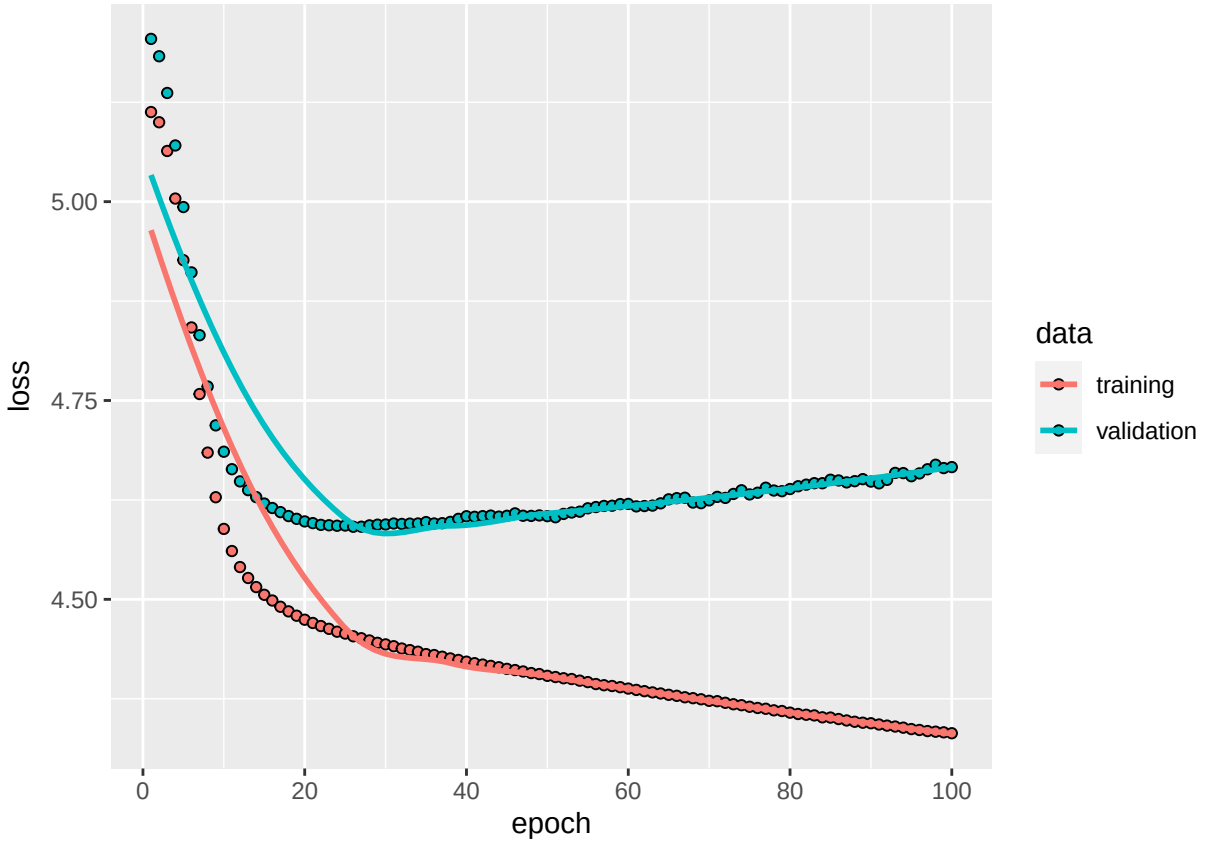
```
# store and use only the best model
cp_path <- paste("./Networks/model_var")

cp_callback <- callback_model_checkpoint(
    filepath = cp_path,
    monitor = "val_loss",
    save_weights_only = TRUE,
    save_best_only = TRUE,
    verbose = 0
)
```

```
fit_var <- model_var %>% fit(
    list(XX), list(YY),
    validation_split = validation_split,
    epochs = epochs,
```

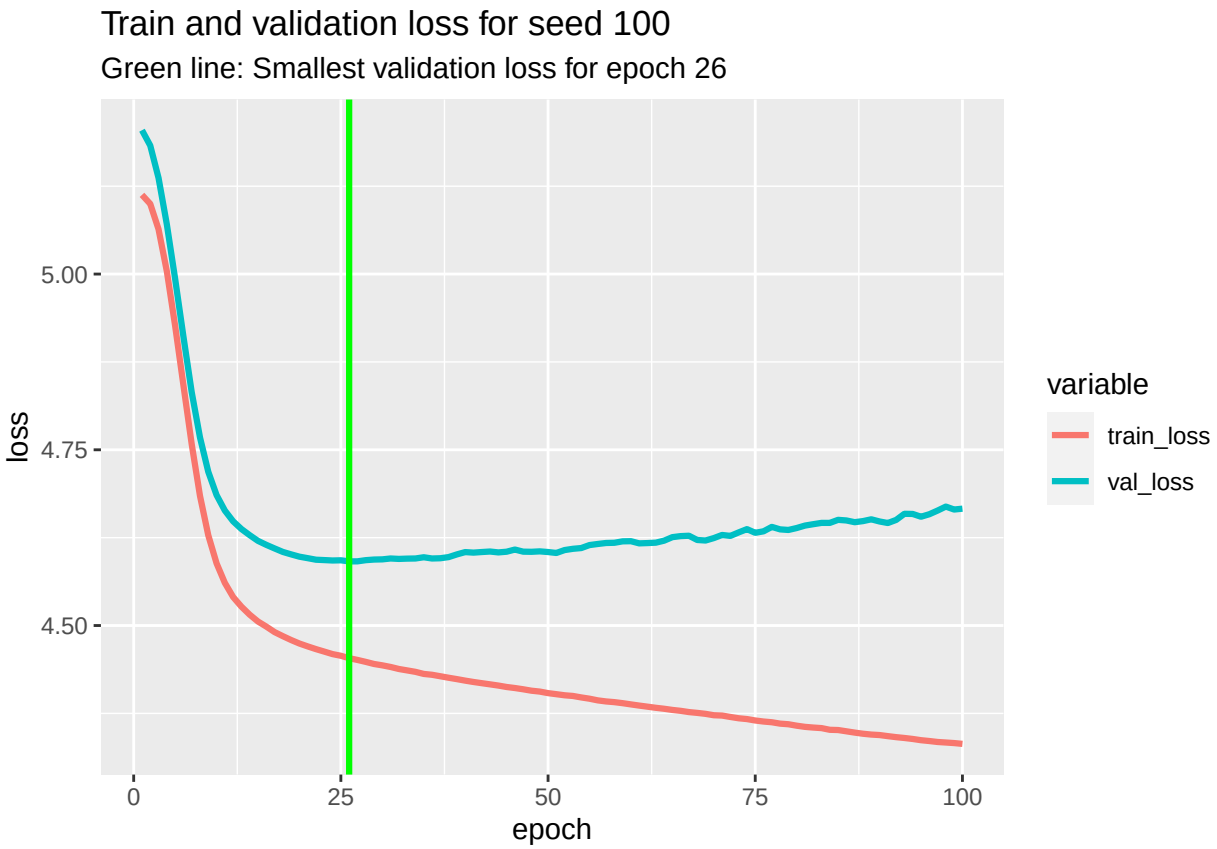
```
    batch_size = batch_size,
    callbacks = list(cp_callback),
    verbose = verbose
)
```

```
plot(fit_var)
```

```
## `geom_smooth()` using formula 'y ~ x'
```




```
keras_plot_loss_min(fit_var, seed)
```




```
load_model_weights_hdf5(model_var, cp_path)
```

Validation

```
# calculating the predictions
learn$fitvar <- as.vector(model_var %>% predict(list(XX)))
test$fitvar <- as.vector(model_var %>% predict(list(TT)))

# average in-sample and out-of-sample losses (in 10^(0))
sprintf("Gamma deviance: %s", round(gamma_loss(learn$Claim, learn$fitvar), 4))
```

```
## [1] "Gamma deviance: 4.4771"
```

```
sprintf("Gamma deviance: %s", round(gamma_loss(test$Claim, test$fitvar), 4))
```

```
## [1] "Gamma deviance: 4.6011"
```

```
# average claims size
sprintf("Average claim size (test): %s", round(mean(test$fitvar), 1))
```

```
## [1] "Average claim size (test): 12777.4"
```

⇒ The inclusion of the two additional (unrelated) components might slightly worsens the predictive performance. The reason is that these additional components give a higher over-fitting potential, and, typically, we stop the SGD algorithm more early.

Extract the regression attentions $\hat{\beta}_j(x_i)$

We extract the estimated covariate dependent regression attentions $\hat{\beta}_j(x_i)$ from the model `model_var` fitted previously. With that, we are able to test the hypothesis $H_0 : \beta_j(x) = 0$, as formulated above, based on the estimated regression attention values $\hat{\beta}_j(x_i)$. It allows to perform variable selection.

To the best of our knowledge, the extraction of the regression attentions values $\hat{\beta}_j(x_i)$ needs to be done "manually" in the sense that no single function is (yet) available to extract them. Main steps: * Predict the outcome of the corresponding layer in the network * Adjustment with the corresponding weight

First, we need to determine the (hidden) layer which provides the regression attentions $\hat{\beta}_j(x_i)$, we do not need the outcome, see the formula of the LocalGLMnet above.

```
# select submodel such that the prediction provides the unweighted regression attentions
zz <- keras_model(inputs = model_var$input, outputs = get_layer(model_var, 'attention')$output)
summary(zz)
```

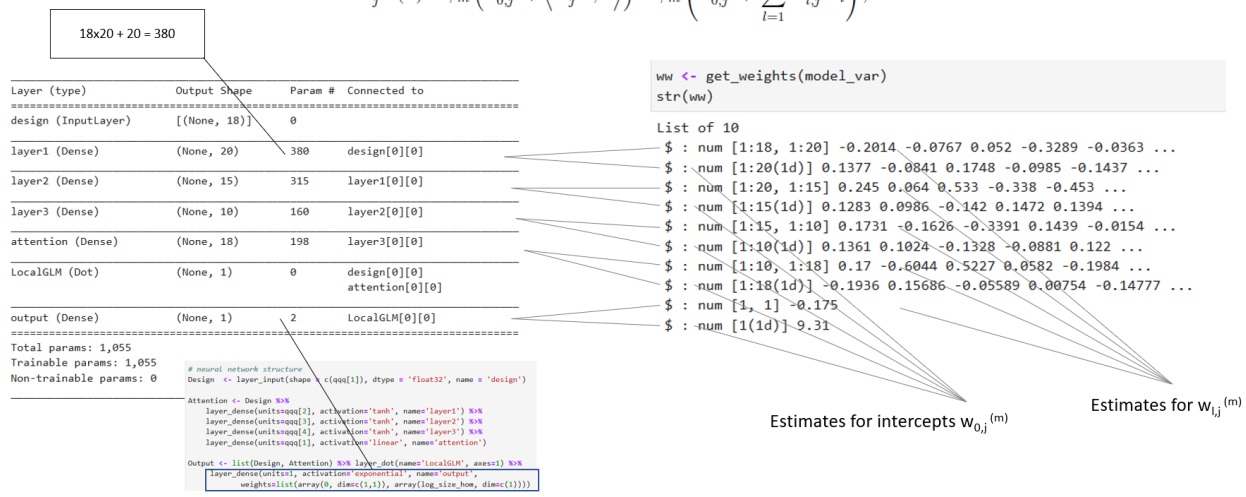
```
## Model: "model_7"
## _________________________________________________________________________________
## Layer (type)                         Output Shape                     Param #
## =================================================================================
## design (InputLayer)                  [(None, 18)]                     0
## _________________________________________________________________________________
## layer1 (Dense)                       (None, 20)                       380
## _________________________________________________________________________________
## layer2 (Dense)                       (None, 15)                       315
## _________________________________________________________________________________
## layer3 (Dense)                       (None, 10)                       160
## _________________________________________________________________________________
## attention (Dense)                    (None, 18)                       198
## =================================================================================
## Total params: 1,053
## Trainable params: 1,053
## Non-trainable params: 0
## _________________________________________________________________________________
```

```
# Calculate the regression attentions for model zz on the test set
beta_x <- data.frame(zz %>% predict(list(TT)))
names(beta_x) <- paste0("Beta", col_names)
```

Second, we need to adjust the predictions with the corresponding layer wieght to get the $\hat{\beta}_j(x_i)$.

Below an illustration which helps to understand the `get_weights` functions.

$$z_j^{(m)}(\boldsymbol{x}) = \phi_m\left(w_{0,j}^{(m)} + \left\langle \boldsymbol{w}_j^{(m)}, \boldsymbol{x}\right\rangle\right) = \phi_m\left(w_{0,j}^{(m)} + \sum_{l=1}^{q_{m-1}} w_{l,j}^{(m)} x_l\right),$$




```
# multiply with the corresponding weight
ww <- get_weights(model_var)
beta_x <- beta_x * as.numeric(ww[[9]])
str(beta_x)
```

```
## 'data.frame':    17866 obs. of  18 variables:
##  $ BetaAge                : num  0.42 0.383 0.209 0.337 0.367 ...
##  $ BetaGender             : num  -0.2477 -0.1558 -0.0601 -0.1568 -0.1877 ...
##  $ BetaDependentChildren  : num  0.1199 -0.0908 -0.073 -0.087 -0.0353 ...
##  $ BetaDependentsOther    : num  0.000942 0.185498 0.121862 0.09973 0.039315 ...
##  $ BetaWeeklyPay          : num  0.479 0.455 0.365 0.403 0.386 ...
##  $ BetaPartTimeFullTime   : num  0.12779 -0.00499 0.0088 -0.02077 -0.01373 ...
##  $ BetaHoursWorkedPerWeek: num  -0.01977 0.07154 0.03684 0.00982 0.00636 ...
##  $ BetaDaysWorkedPerWeek  : num  0.0384 -0.0566 -0.0868 0.0302 0.0601 ...
##  $ BetaAccYear            : num  0.369 0.264 0.204 0.142 0.167 ...
##  $ BetaAccMonth           : num  0.028 0.0296 0.121 0.0116 -0.0341 ...
##  $ BetaAccWeekday         : num  -0.00686 -0.04016 -0.02335 -0.06853 -0.02311 ...
##  $ BetaAccTime            : num  -0.07869 0.01317 -0.03985 0.02662 -0.00484 ...
##  $ BetaRepDelay           : num  0.0593 0.1701 0.0935 0.1817 0.1521 ...
##  $ BetaRandUN             : num  -0.0367 -0.01879 0.00843 -0.01604 -0.02632 ...
##  $ BetaRandNN             : num  0.0675 -0.1093 -0.047 -0.0924 -0.0514 ...
##  $ BetaMarital1           : num  -0.1388 -0.1443 -0.3015 0.0987 0.1721 ...
##  $ BetaMarital2           : num  -0.2548 -0.0672 -0.0721 -0.063 -0.1 ...
##  $ BetaMarital3           : num  0.3431 0.0921 0.0425 0.1768 0.2398 ...
```

`beta_x` are now the regression attentions $\hat{\beta}_j(x_i)$ of the test set $\mathcal{T}$.

The regression attentions $\hat{\beta}_j(x_i)$ on the learning set $\mathcal{L}$.

```
# attention beta(x) on the learning set
beta_xL <- data.frame(zz %>% predict(list(XX)))
names(beta_xL) <- paste0("Beta", col_names)
beta_xL <- beta_xL * as.numeric(ww[[9]])
```

Empirical hypothesis figures

For the empirical hypothesis testing, * We extract the estimated regression attentions $\hat{\beta}_{q+1}(x_i)$ and $\hat{\beta}_{q+2}(x_i)$ of the randomly added variables, and we calculate their empirical means and their empirical standard de-

viations * We calculate the 0.1% rejection region * We calculate the empirical coverage ratio for every feature

```
# mean and std.dev. of additional (randomly generated) components q+1 and q+2
mean_rand <- c(mean(beta_xL$BetaRandUN), mean(beta_xL$BetaRandNN))
sd_rand <- c(sd(beta_xL$BetaRandUN), sd(beta_xL$BetaRandNN))
sprintf("The means of the random variables regression attentions are:  %s", paste(round(mean_rand, 4), c
```

```
## [1] "The means of the random variables regression attentions are:  -0.0214  -0.007"
```

```
sprintf("The stdevs of the random variables regression attentions are:  %s", paste(round(sd_rand, 4), co
```

```
## [1] "The stdevs of the random variables regression attentions are:  0.023  0.0696"
```

```
# 0.1% rejection region
quant_rand <- mean(sd_rand) * abs(qnorm(0.0005))
cat("The 0.1% rejection region is 0 +/-", quant_rand)
```

```
## The 0.1% rejection region is 0 +/- 0.1522732
```

```
# in-sample coverage ratio for all features
num_names <- 1:(length(col_names)-3)
II <- data.frame(array(NA, c(1, length(num_names))))
names(II) <- col_names[num_names]
for (k1 in 1:length(num_names)) {
  II[1, k1] <- 1 - (sum(as.integer(-beta_xL[, num_names[k1]] > quant_rand)) +
                    sum(as.integer(beta_xL[, num_names[k1]] > quant_rand))) / nrow(beta_xL)
}
round(II, 4)
```

```
##      Age Gender DependentChildren DependentsOther WeeklyPay PartTimeFullTime
## 1 0.0099 0.2649            0.9647          0.9304    0.0066           0.9818
##   HoursWorkedPerWeek DaysWorkedPerWeek AccYear AccMonth AccWeekday AccTime
## 1                  1                 1  0.1407   0.9993          1       1
##   RepDelay RandUN RandNN
## 1   0.8015      1 0.9776
```

Plot the regression attentions

```
## merge covariates x with beta(x) on the test set
beta_x <- cbind(TT, beta_x)
```

We show the estimated regression attentions for all continuous and binary components j for 5000 randomly selected claims (we do not display all claims to not overload the plots).

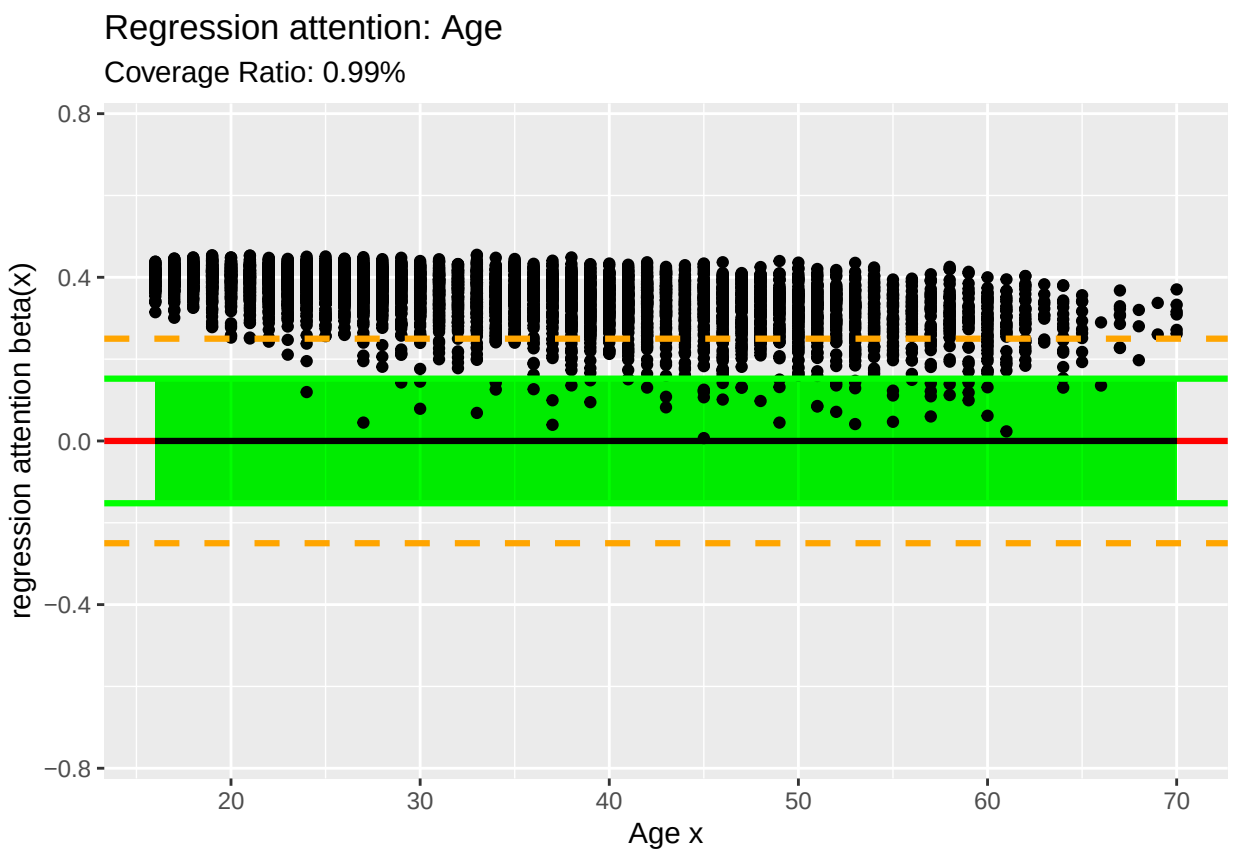
```
## select at random 5000 claims to not overload plots
nsample <- 5000
set.seed(seed)
idx <- sample(x = 1:nrow(test), size = nsample)
```

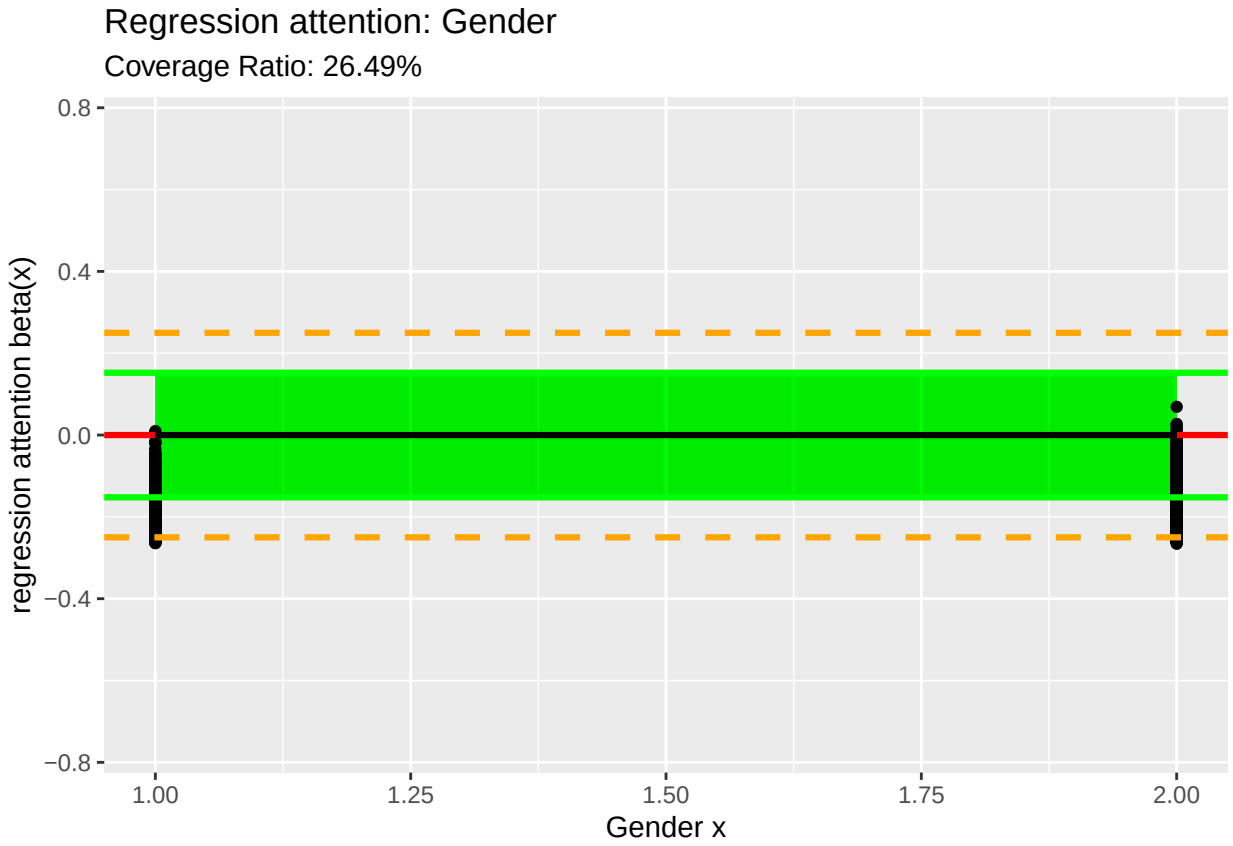
```
# data for plotting
beta_x_smp <- beta_x[idx, ]
test_smp <- test[idx, ]
test_smp$Gender <- as.numeric(test_smp$Gender)
test_smp$PartTimeFullTime <- as.numeric(test_smp$PartTimeFullTime)
```

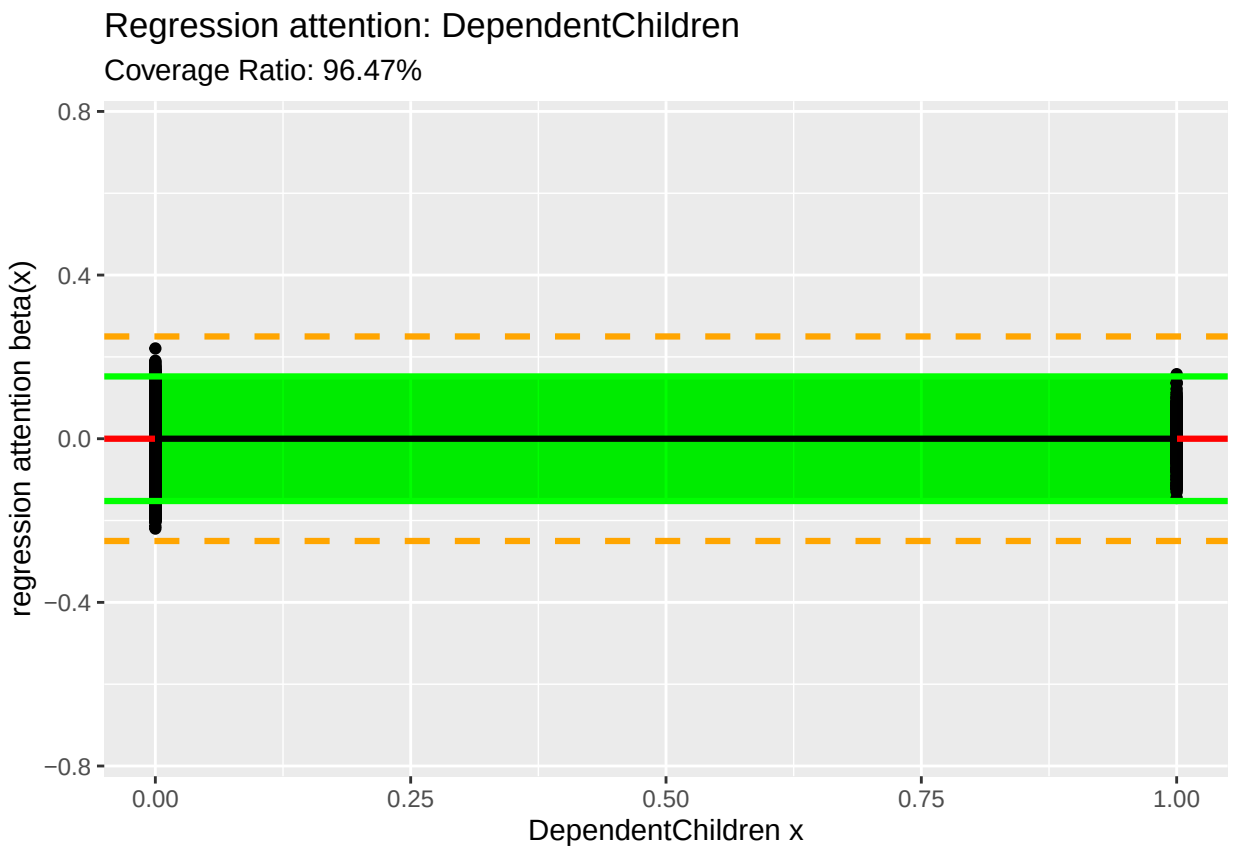
Below we show the regression attentions of 50000 randomly selected covariates. The shaded green area shows the interval $\mathcal{I}_{99{:}9\%}$, and the text in the subtitle shows the coverage ratio (CR) of this interval $\mathcal{I}_{99{:}9\%}$. The

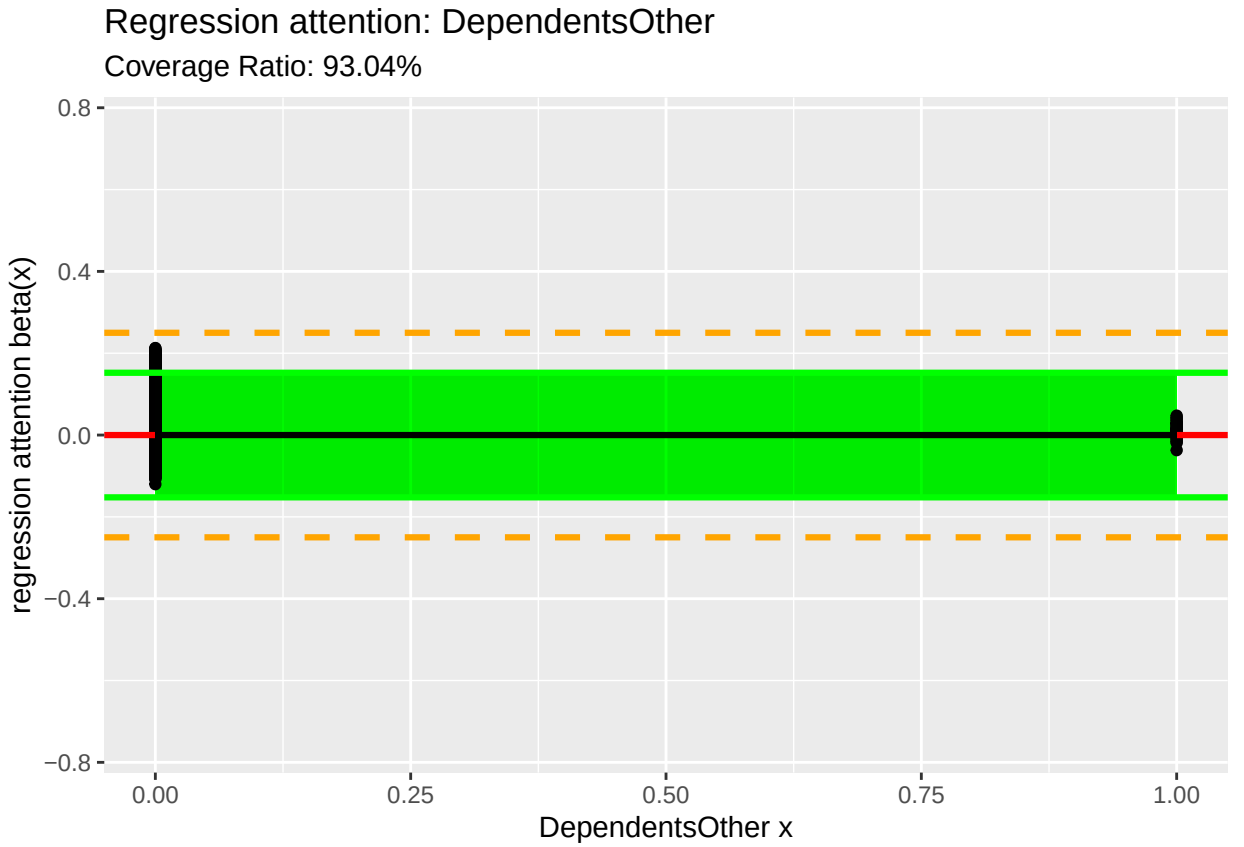
y-scale is identical in all plots (the orange lines are at$+/-0.25$ for orientation purposes), and on the x-scale we have covariate components x_j.

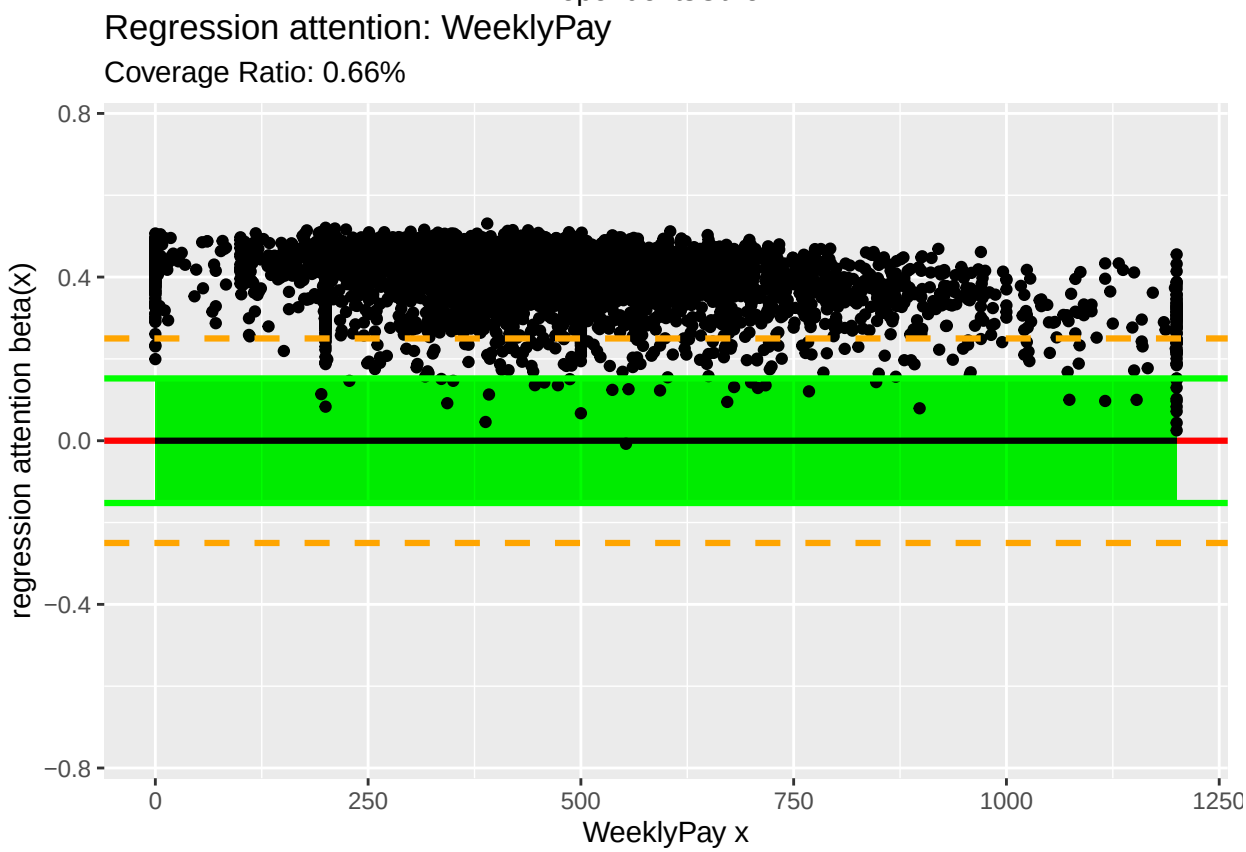
```
# Plotting for all continuous and binary features
for (ll in 1:length(num_names)) {
    kk <- num_names[ll]
    dat_plt <- data.frame(var = test_smp[, col_names[ll]],
                          bx = beta_x_smp[, kk + length(col_features)],
                          col = rep("green", nsample))
    plt <- ggplot(dat_plt, aes(x = var, y = bx)) + geom_point() +
              geom_hline(yintercept = 0, colour = "red", size = line_size) +
              geom_hline(yintercept = c(-quant_rand, quant_rand), colour = "green", size = line_size) +
              geom_hline(yintercept = c(-1,1)/4, colour = "orange", size = line_size, linetype = "dashe
              geom_rect(
                mapping = aes(xmin = min(var), xmax = max(var), ymin = -quant_rand, ymax = quant_rand),
                fill = dat_plt$col, alpha = 0.002
              ) + lims(y = c(-0.75,0.75)) +
              labs(title = paste0("Regression attention: ", col_names[ll]),
                   subtitle = paste0("Coverage Ratio: ", paste0(round(II[, col_names[ll]] * 100, 2)), "
                   x = paste0(col_names[ll], " x"), y = "regression attention beta(x)")
    print(plt)
}
```

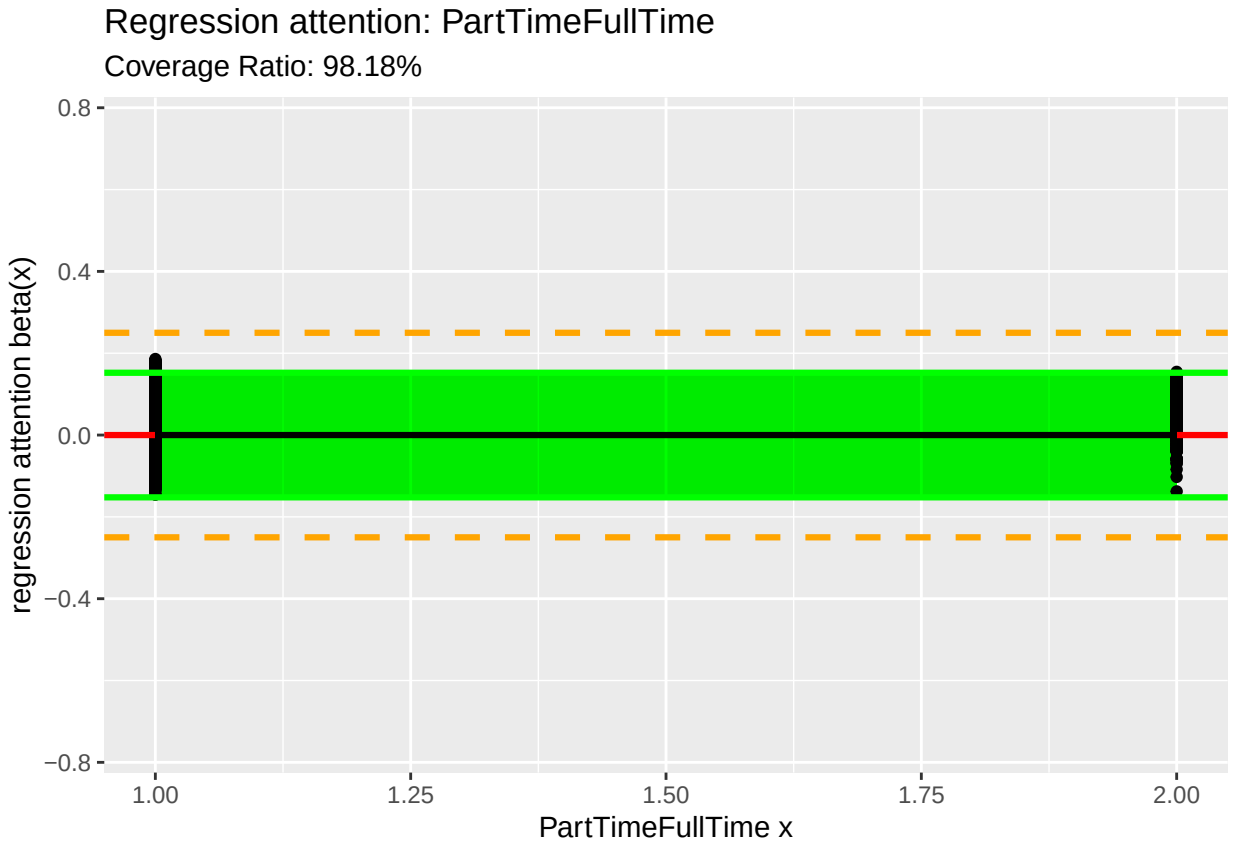


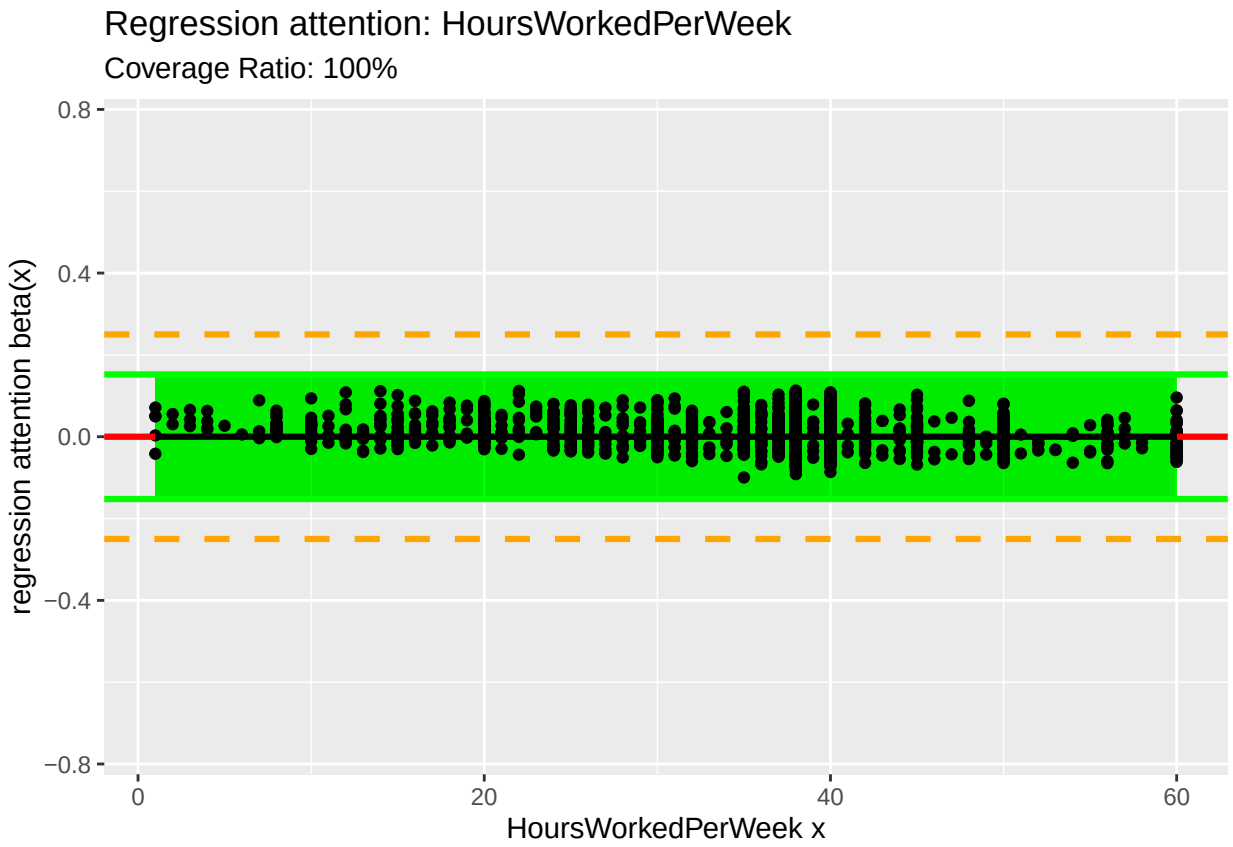

Regression attention: Gender
Coverage Ratio: 26.49%
0.8
0.4
0.0
−0.4
−0.8
regression attention beta(x)
1.00
1.25
1.50
1.75
2.00
Gender x

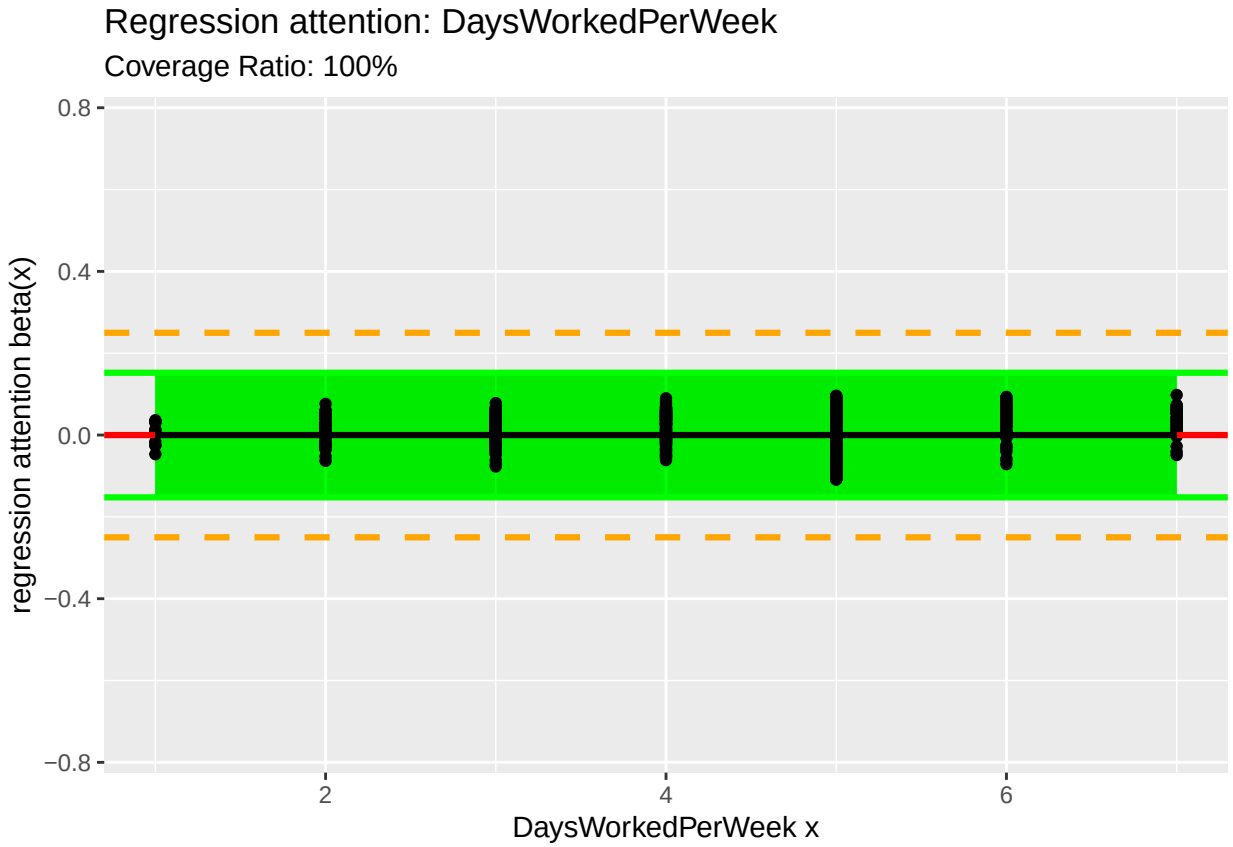

Regression attention: DependentChildren
Coverage Ratio: 96.47%
0.8
0.4
0.0
−0.4
−0.8
regression attention beta(x)
0.00
0.25
0.50
0.75
1.00
DependentChildren x

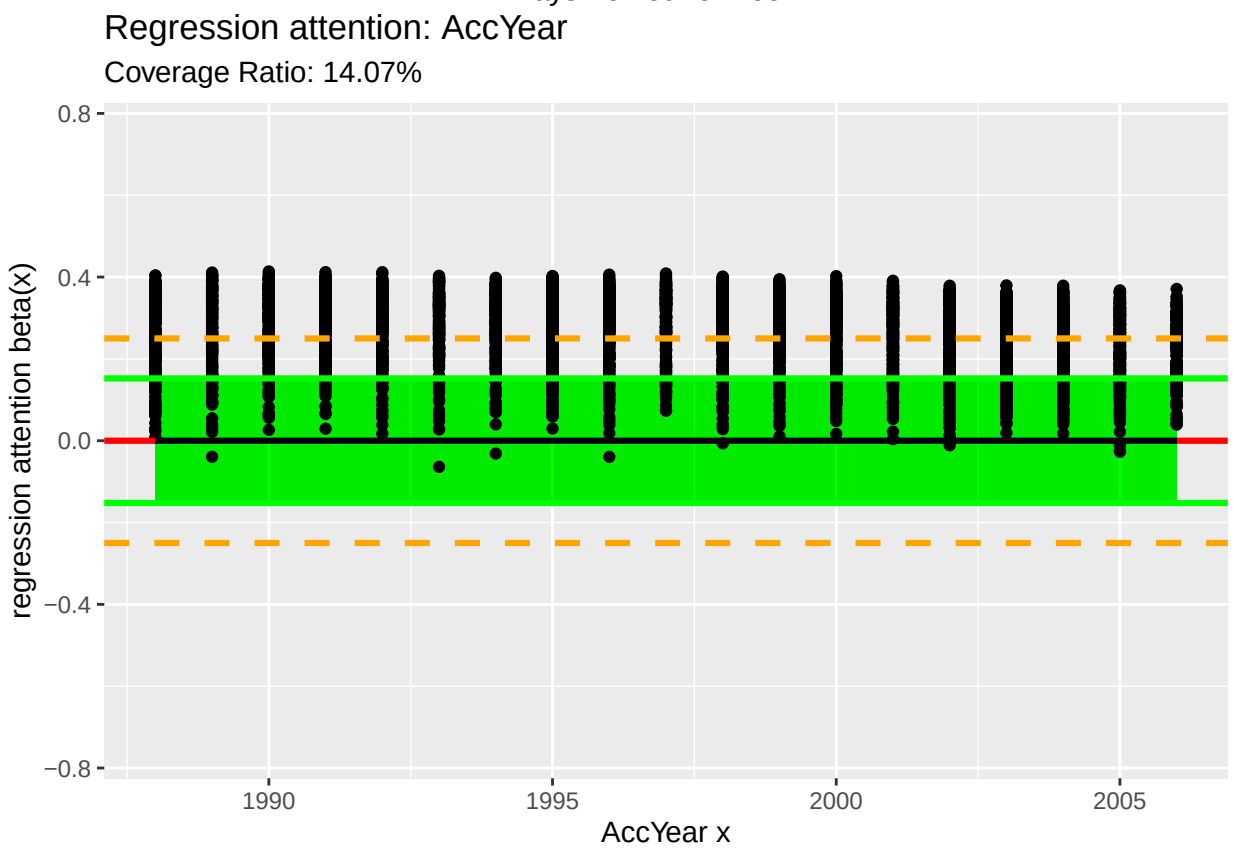

Regression attention: DependentsOther
Coverage Ratio: 93.04%
regression attention beta(x)
0.8
0.4
0.0
−0.4
−0.8
0.00
0.25
0.50
0.75
1.00
DependentsOther x

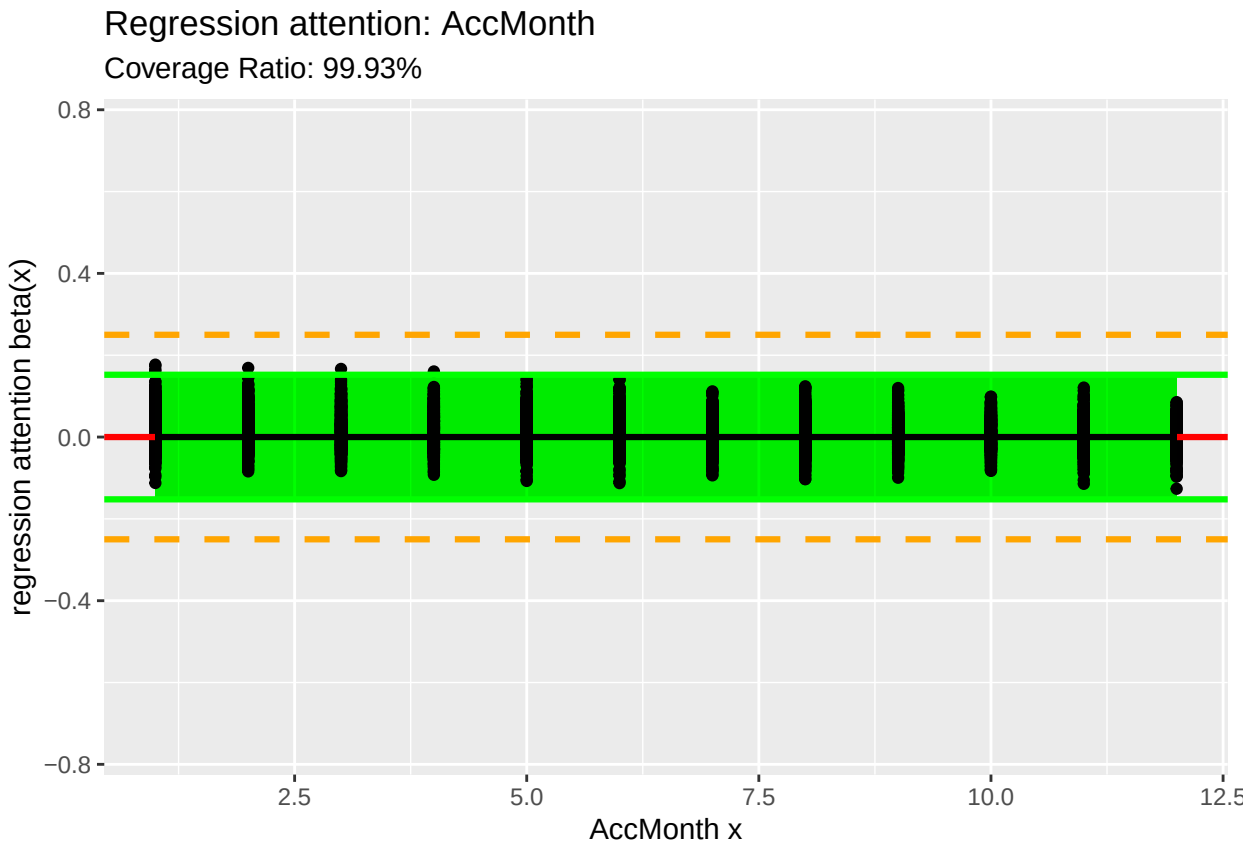

Regression attention: WeeklyPay
Coverage Ratio: 0.66%
regression attention beta(x)
0.8
0.4
0.0
−0.4
−0.8
0
250
500
750
1000
1250
WeeklyPay x

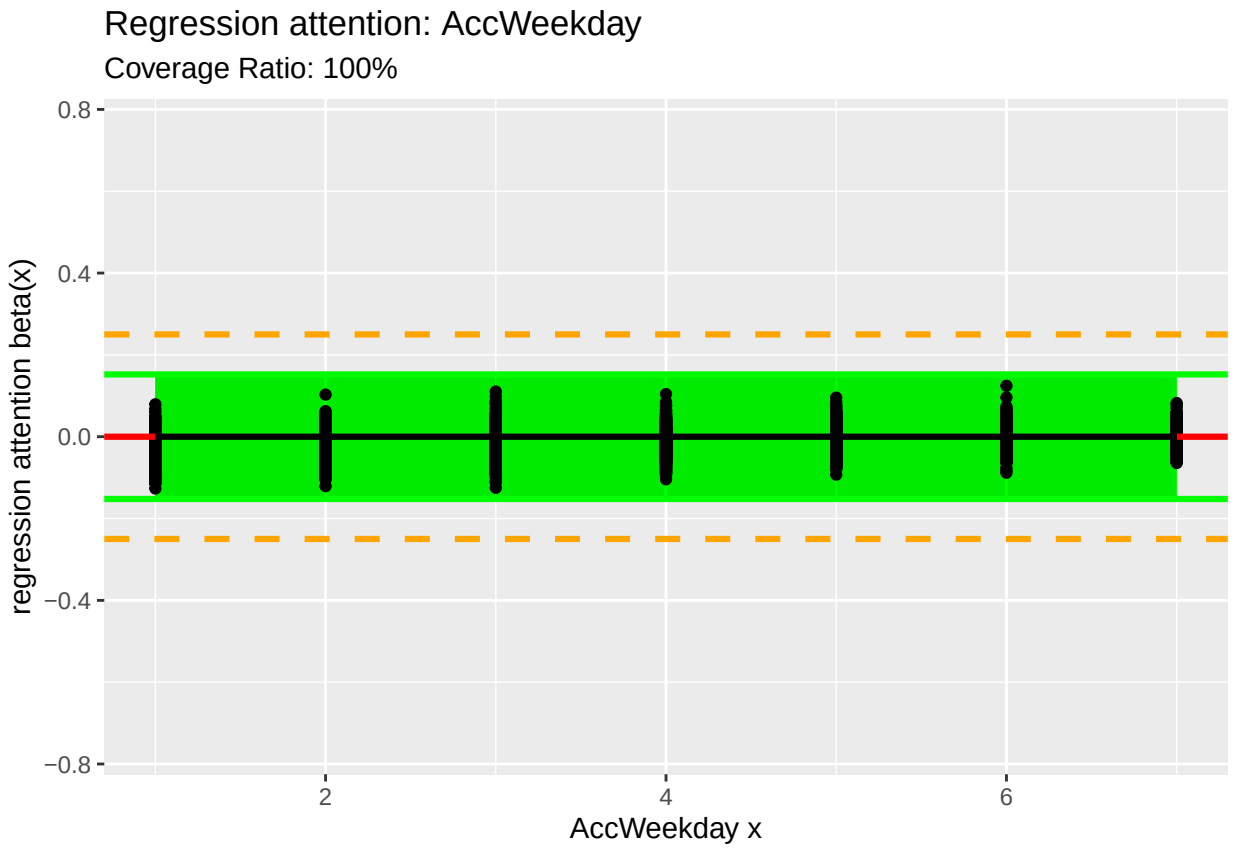

Regression attention: PartTimeFullTime
Coverage Ratio: 98.18%
regression attention beta(x)
0.8
0.4
0.0
−0.4
−0.8
1.00
1.25
1.50
1.75
2.00
PartTimeFullTime x

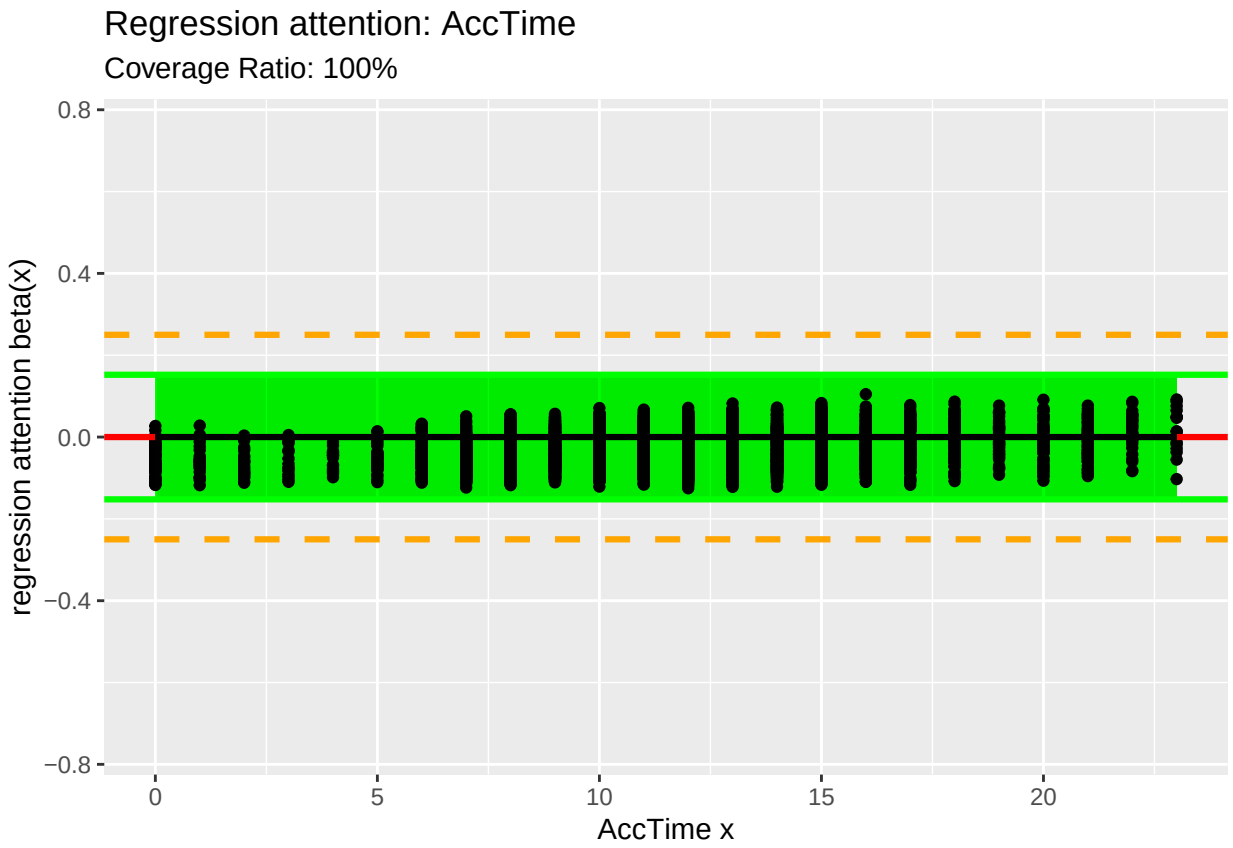

Regression attention: HoursWorkedPerWeek
Coverage Ratio: 100%
regression attention beta(x)
0.8
0.4
0.0
−0.4
−0.8
0
20
40
60
HoursWorkedPerWeek x

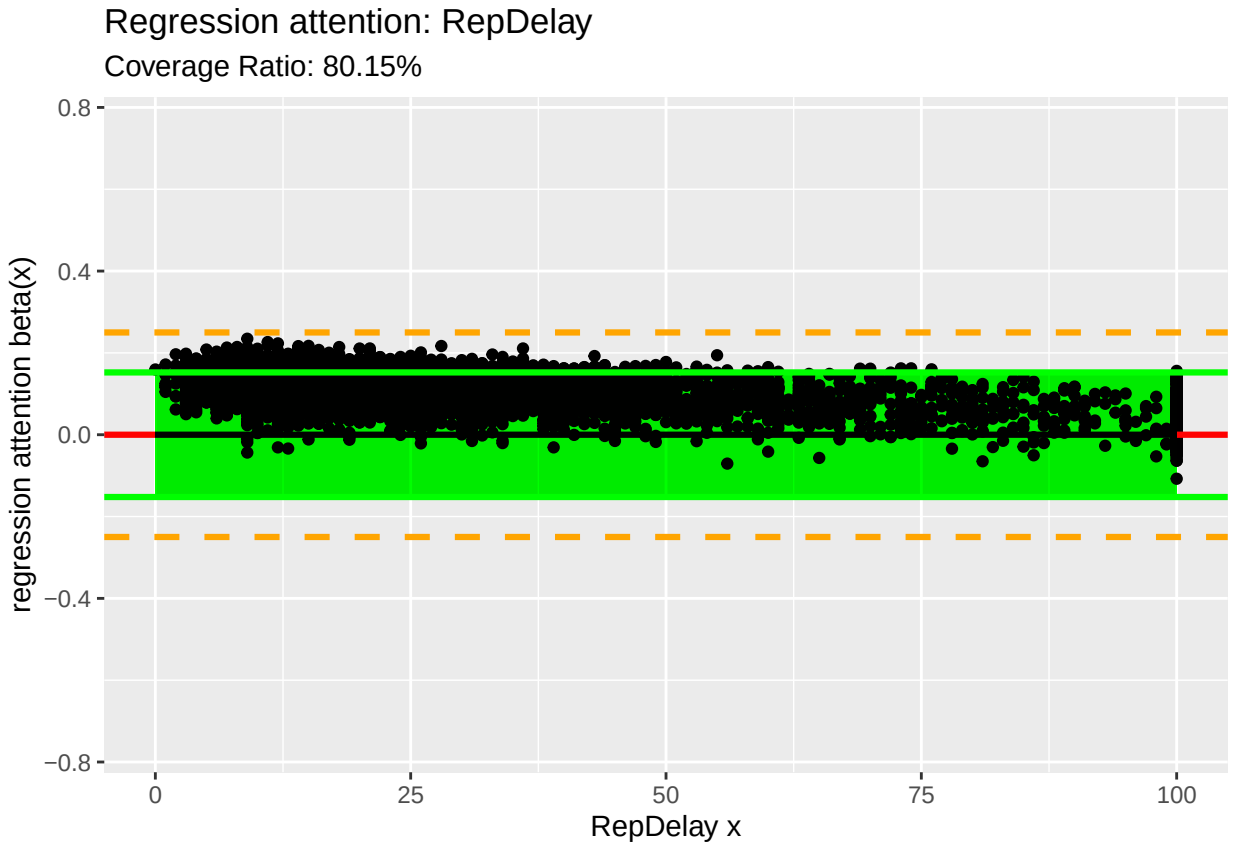

Regression attention: DaysWorkedPerWeek
Coverage Ratio: 100%
0.8
0.4
0.0
−0.4
−0.8
regression attention beta(x)
2
4
6
DaysWorkedPerWeek x

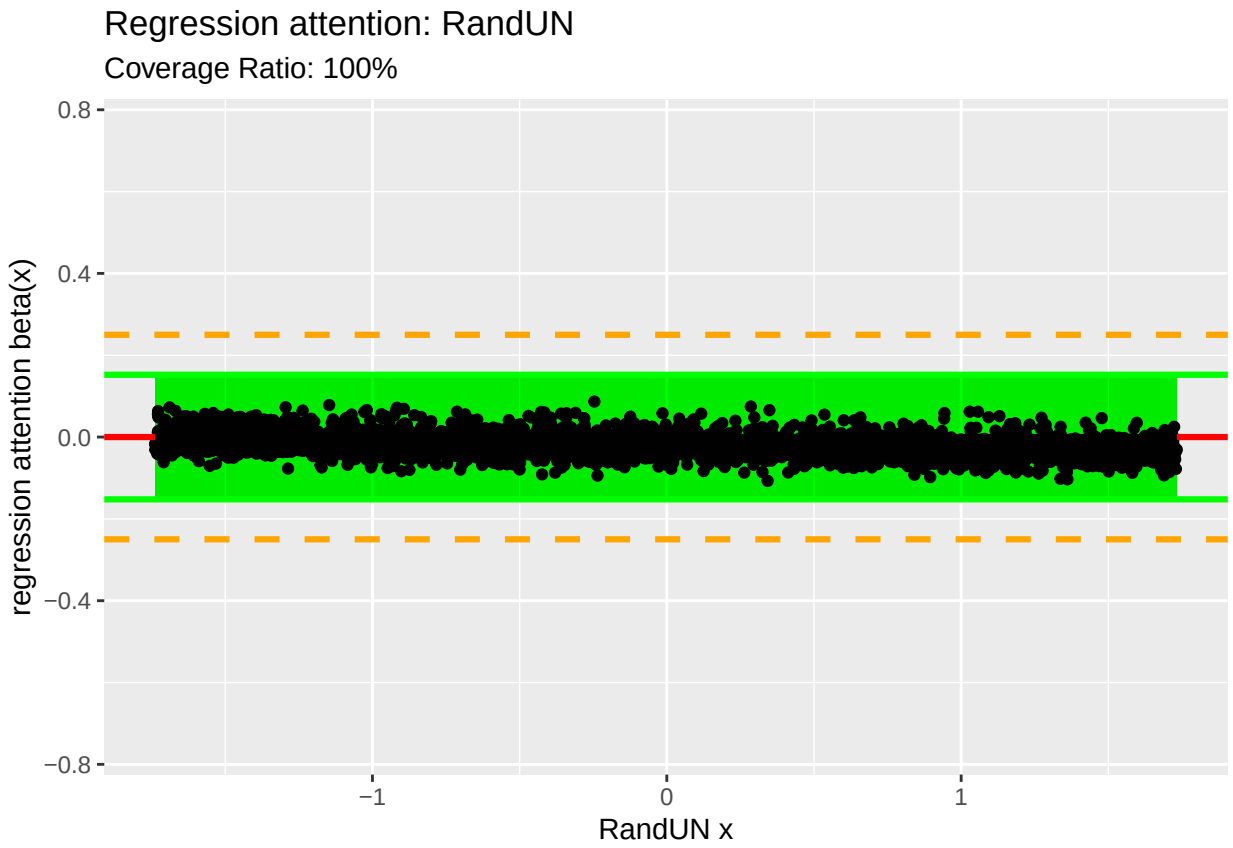

Regression attention: AccYear
Coverage Ratio: 14.07%
0.8
0.4
0.0
−0.4
−0.8
regression attention beta(x)
1990
1995
2000
2005
AccYear x

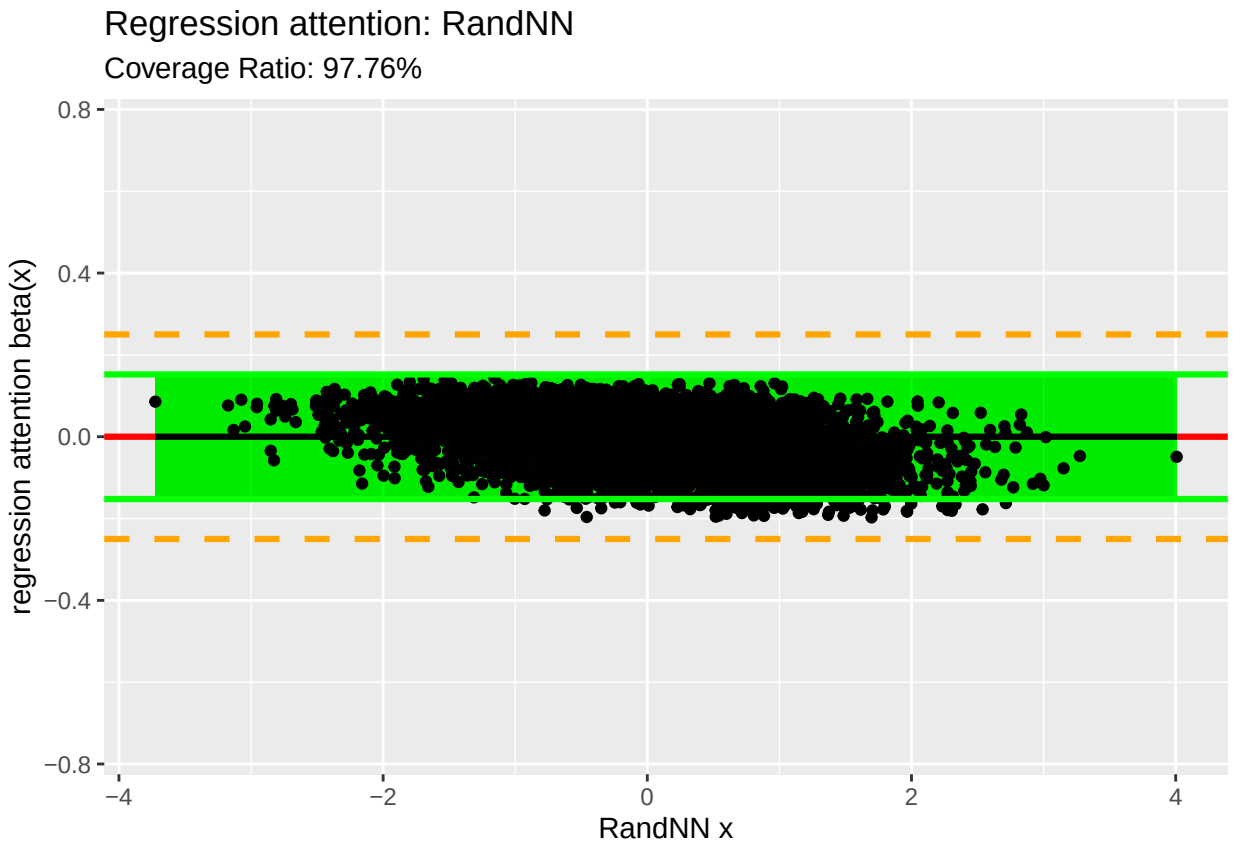

Regression attention: AccMonth
Coverage Ratio: 99.93%
regression attention beta(x)
0.8
0.4
0.0
−0.4
−0.8
2.5
5.0
7.5
10.0
12.5
AccMonth x


Regression attention: AccWeekday
Coverage Ratio: 100%
regression attention beta(x)
0.8
0.4
0.0
−0.4
−0.8
2
4
6
AccWeekday x


Regression attention: AccTime
Coverage Ratio: 100%
regression attention beta(x)
0.8
0.4
0.0
−0.4
−0.8
0
5
10
15
20
AccTime x


Regression attention: RepDelay
Coverage Ratio: 80.15%
regression attention beta(x)
0.8
0.4
0.0
−0.4
−0.8
0
25
50
75
100
RepDelay x


Regression attention: RandUN
Coverage Ratio: 100%
regression attention beta(x)
0.8
0.4
0.0
−0.4
−0.8
−1
0
1
RandUN x


Regression attention: RandNN
Coverage Ratio: 97.76%
regression attention beta(x)
0.8
0.4
0.0
−0.4
−0.8
−4
−2
0
2
4
RandNN x

These coverage ratios are around 99.9% for the covariates HoursWorkedPerWeek, DaysWorkedPerWeek and AccWeekday (besides the two purely random components). This suggests that these three terms may not be necessary in our regression model, and the remaining terms seem significant, i.e., the null hypothesis H_0 of setting these regression attentions to zero is rejected. We mention that HoursWorkedPerWeek and DaysWorkedPerWeek are almost fully concentrated in one single value. These terms seem to not allow to suffciently differentiate the claims.

$\Rightarrow$ We have presented an empirical test for the null hypothesis $H_0 : \beta_j(x) = 0$ for a given $1 \leq j \leq q$. This null hypothesis can be tested using the coverage ratio of the (empirical) interval I99:9%. This test requires some care, as it based on the fitted LocalGLMnet, and if the purely random additional components show to much structure in the fitted model, then, for sure, the model over-fits to these components. If this bias is too large, a different seed should be chosen for SGD fitting. We also mention that this test is only useful for continuous and binary variables because the one-hot encoded categorical variables do not live on the same scale (centered and unit variance).

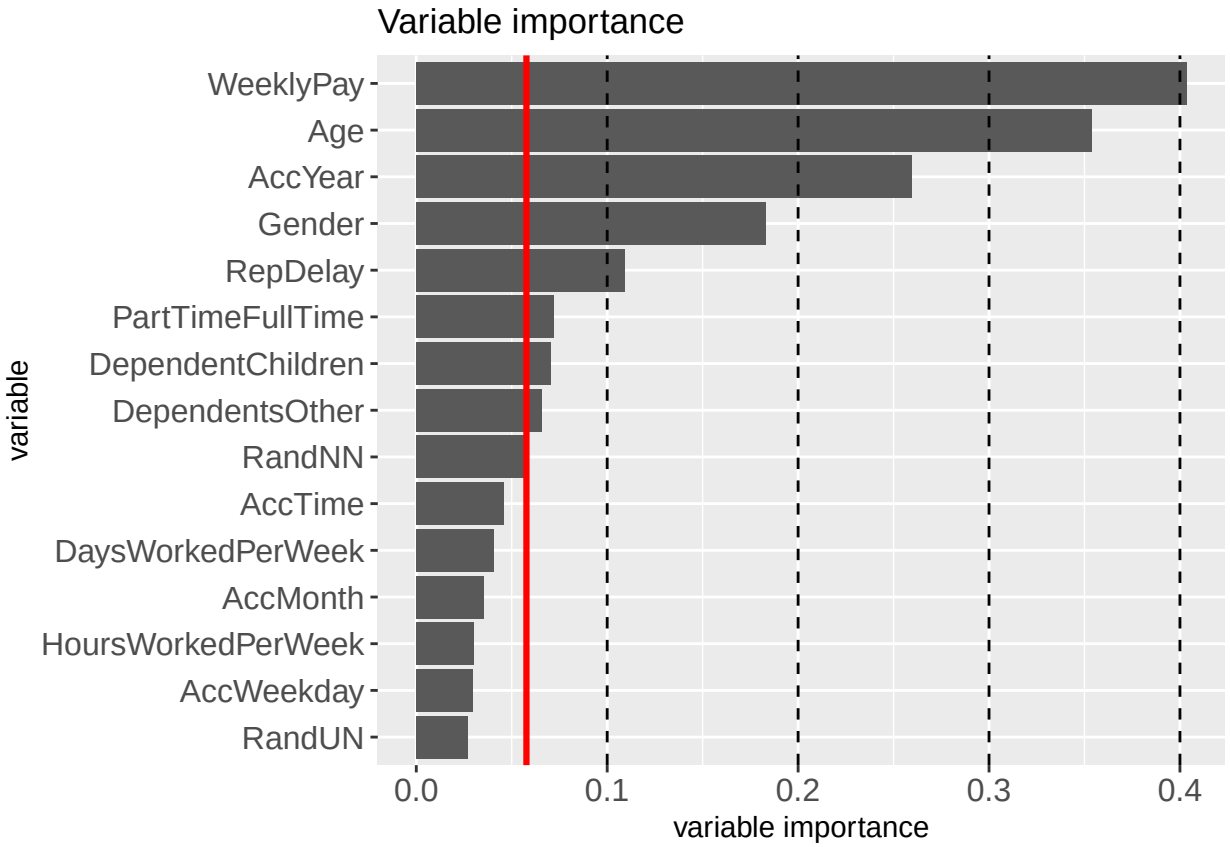
Variable importance plot

An evident empirical measure for variable importance (VI) of continuous and binary covariates is defined by

$$\mathrm{VI}_j = \frac{1}{n} \sum_{i=1}^{n} \left| \widehat{\beta}_j(\boldsymbol{x}_i) \right|, \tag{5.7}$$

```
# calculate variable importance
var_imp <- abs(beta_xL[, num_names])

# store as data.frame for plotting
dat_var_imp <- data.frame(vi = colMeans(var_imp), names = col_names[1:length(num_names)])

# limits from random generated variables
dat_var_imp_limit <- dat_var_imp %>% filter(names %in% c("RandUN", "RandNN"))
limit_rand <- max(dat_var_imp_limit$vi)
```

```
ggplot(dat_var_imp, aes(x = vi)) + geom_col(aes(y = reorder(names, vi))) +
    geom_vline(xintercept = seq(0.1, 0.4, by = 0.1), col = "gray1", linetype = "dashed") +
    geom_vline(xintercept = limit_rand, col = "red", size = line_size) +
    theme(axis.text = element_text(size = 12)) +
    labs(title = "Variable importance", x = "variable importance", y = "variable")
```



Variable importance of the continuous and binary covariates, the red vertical line gives the maximum importance of the two additional randomly generated components.

The most important variables being WeeklyPay, AccYear and Age. At the other end we have the least important variables AccMonth, AccTime, HoursWorkedPerWeek and AccWeekday. Thus, compared to the coverage ratio we additionally question the significance of AccTime from the variable importance plot.

⇒ Based on this chapter, it is appropriate to exclude certain variables from the model. Next, we will fit the reduced model and provide further insights for the covariate distribution and interactions.

Reduced model

Data preparation

```
reduce <- c("AccWeekday", "HoursWorkedPerWeek", "DaysWorkedPerWeek")

col_featuresR <- setdiff(col_features, c(paste0(reduce, "NN"), "RandUN", "RandNN"))
col_namesR <- setdiff(col_names, c(reduce, "RandUN", "RandNN"))
```

```
# Size of input for neural networks
q0 <- length(col_featuresR)
qqq <- c(q0, c(20, 15, 10), 1)

sprintf("Neural network with K=3 hidden layer")
```

```
## [1] "Neural network with K=3 hidden layer"
```

```
sprintf("Input feature dimension: q0 = %s", q0)
```

```
## [1] "Input feature dimension: q0 = 13"
```

```
sprintf("Number of hidden neurons first layer: q1 = %s", qqq[2])
```

```
## [1] "Number of hidden neurons first layer: q1 = 20"
```

```
sprintf("Number of hidden neurons second layer: q2 = %s", qqq[3])
```

```
## [1] "Number of hidden neurons second layer: q2 = 15"
```

```
sprintf("Number of hidden neurons third layer: q3 = %s", qqq[4])
```

```
## [1] "Number of hidden neurons third layer: q3 = 10"
```

```
sprintf("Output dimension: %s", qqq[5])
```

```
## [1] "Output dimension: 1"
```

```
# matrices
# YY remains the same
XX <- as.matrix(learn[, col_featuresR])
TT <- as.matrix(test[, col_featuresR])
```

Definition

```
# neural network structure
Design <- layer_input(shape = c(qqq[1]), dtype = 'float32', name = 'design')

Attention <- Design %>%
    layer_dense(units = qqq[2], activation = 'tanh', name = 'layer1') %>%
    layer_dense(units = qqq[3], activation = 'tanh', name = 'layer2') %>%
    layer_dense(units = qqq[4], activation = 'tanh', name = 'layer3') %>%
    layer_dense(units = qqq[1], activation = 'linear', name = 'attention')

Output <- list(Design, Attention) %>% layer_dot(name = 'LocalGLM', axes = 1) %>%
    layer_dense(units = 1, activation = 'exponential', name = 'output',
                weights = list(array(0, dim = c(1,1)), array(log_size_hom, dim = c(1))))
```

```
model_red <- keras_model(inputs = list(Design), outputs = c(Output))
```

Compilation

```
model_red %>% compile(
    loss = k_gamma_loss,
    optimizer = 'nadam'
)
summary(model_red)
```

```
## Model: "model_8"
## __________________________________________________________________________________
## Layer (type)                Output Shape      Param #  Connected to
## ==================================================================================
## design (InputLayer)         [(None, 13)]      0
## __________________________________________________________________________________
## layer1 (Dense)              (None, 20)        280      design[0][0]
## __________________________________________________________________________________
## layer2 (Dense)              (None, 15)        315      layer1[0][0]
```

```
## ____________________________________________________________________________
## layer3 (Dense)             (None, 10)        160       layer2[0][0]
## ____________________________________________________________________________
## attention (Dense)          (None, 13)        143       layer3[0][0]
## ____________________________________________________________________________
## LocalGLM (Dot)             (None, 1)         0         design[0][0]
##                                                        attention[0][0]
## ____________________________________________________________________________
## output (Dense)             (None, 1)         2         LocalGLM[0][0]
## ============================================================================
## Total params: 900
## Trainable params: 900
## Non-trainable params: 0
## ____________________________________________________________________________
```

Fitting

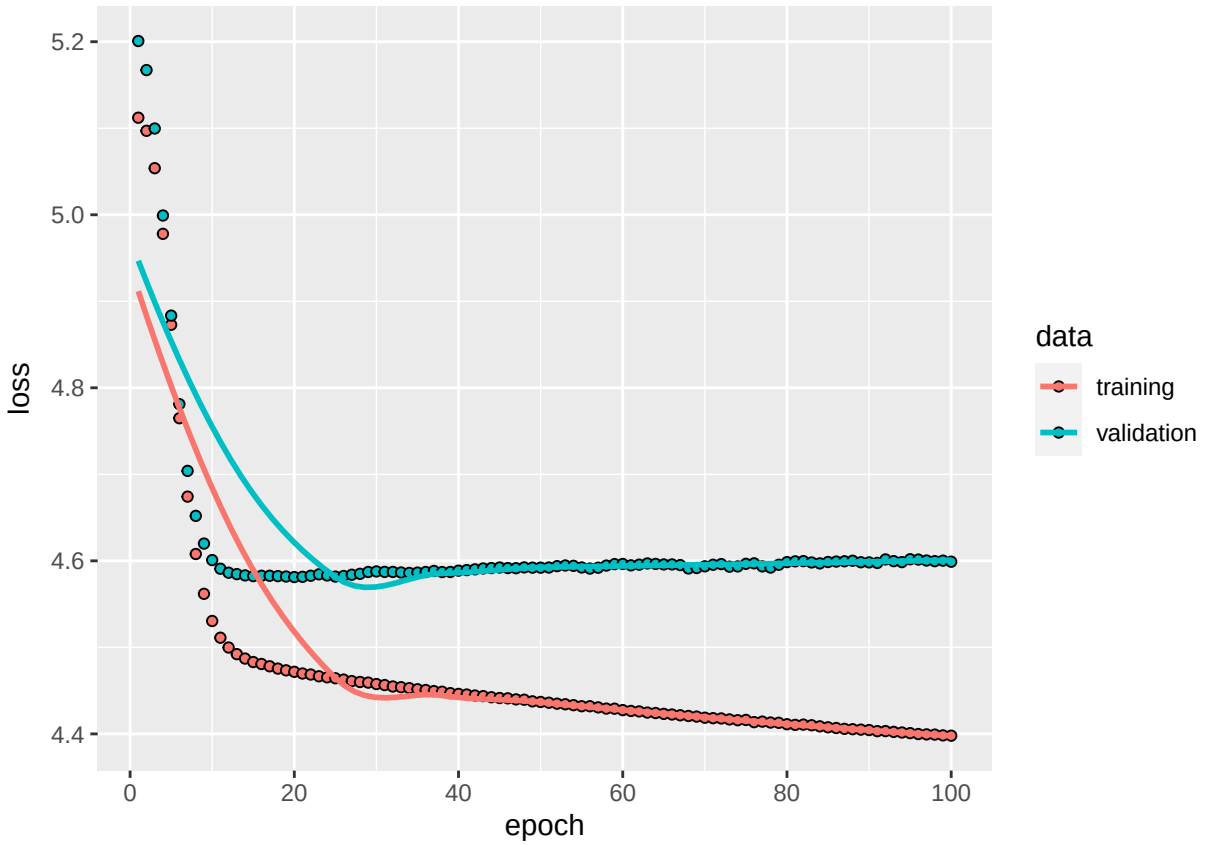
```
# set hyperparameters
epochs <- 100
batch_size <- 5000
validation_split <- 0.2 # set to >0 to see train/validation loss in plot(fit)
verbose <- 1
```

```
# store and use only the best model
cp_path <- paste("./Networks/model_red")

cp_callback <- callback_model_checkpoint(
    filepath = cp_path,
    monitor = "val_loss",
    save_weights_only = TRUE,
    save_best_only = TRUE,
    verbose = 0
)
```

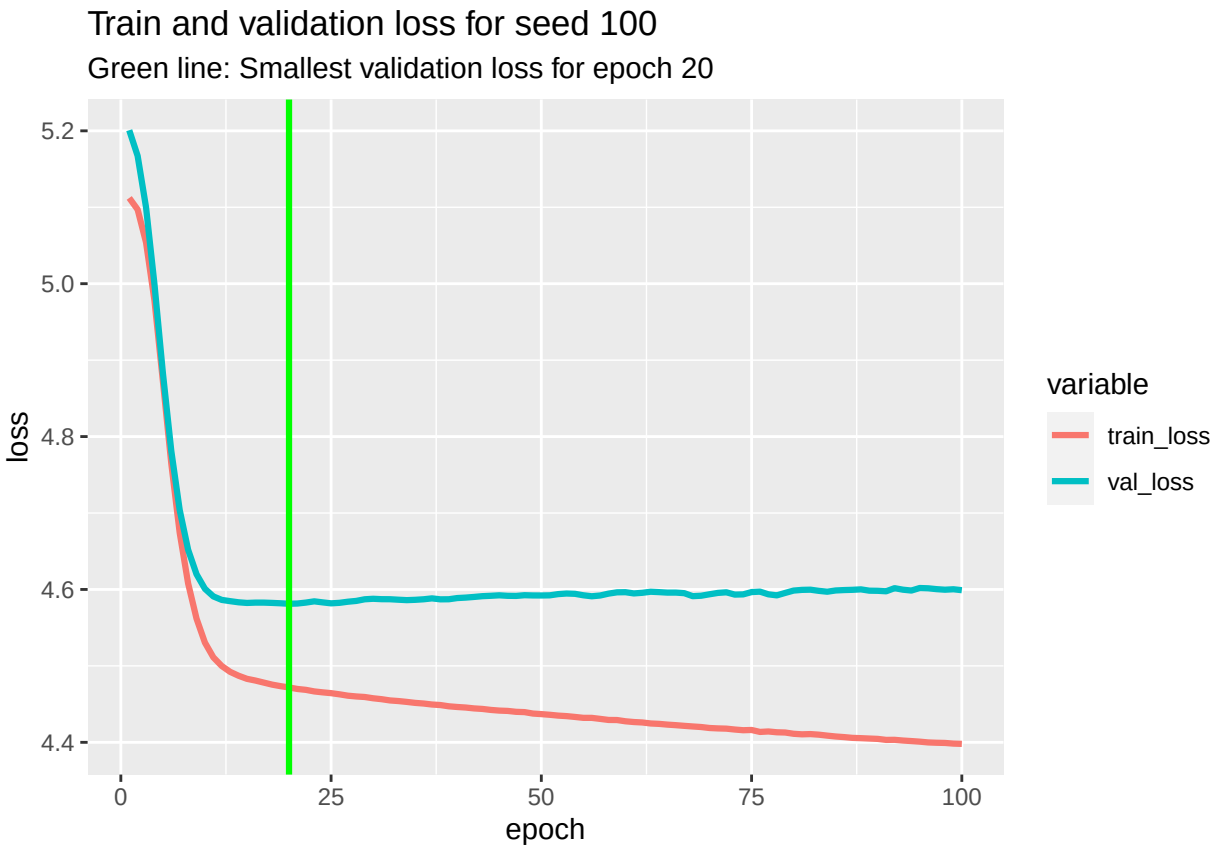
```
fit_red <- model_red %>% fit(
    list(XX), list(YY),
    validation_split = validation_split,
    epochs = epochs,
    batch_size = batch_size,
    callbacks = list(cp_callback),
    verbose = verbose
)
```

```
plot(fit_red)
```

```
## `geom_smooth()` using formula 'y ~ x'
```




```
keras_plot_loss_min(fit_red, seed)
```




```
load_model_weights_hdf5(model_red, cp_path)
```

Validation

```
# calculating the predictions
learn$fitred <- as.vector(model_red %>% predict(list(XX)))
test$fitred <- as.vector(model_red %>% predict(list(TT)))

# average in-sample and out-of-sample losses (in 10^(0))
sprintf("Gamma deviance: %s", round(gamma_loss(learn$Claim, learn$fitred), 4))
```

```
## [1] "Gamma deviance: 4.4911"
```

```
sprintf("Gamma deviance: %s", round(gamma_loss(test$Claim, test$fitred), 4))
```

```
## [1] "Gamma deviance: 4.5896"
```

```
# average claims size
sprintf("Average claim size (test): %s", round(mean(test$fitred), 0))
```

```
## [1] "Average claim size (test): 12837"
```

LocalGLMnet: Covariate contributions $\hat{\beta}_j(x)x_j$

Next we plot covariate contributions $\hat{\beta}_j(x)x_j$, which is the attribution of the predictor to the different additive components, see formula (5.3) below.

Assumptions 5.1 (LocalGLMnet) *Choose a FFN network architecture of depth $d \in \mathbb{N}$ with input and output dimensions being equal to $q_0 = q_d = q$ to model the regression attentions*

$$\begin{aligned} \boldsymbol{\beta} : \mathbb{R}^q &\to \mathbb{R}^q \\ \boldsymbol{x} &\mapsto \boldsymbol{\beta}(\boldsymbol{x}) = \boldsymbol{z}^{(d:1)}(\boldsymbol{x}) = \left(\boldsymbol{z}^{(d)} \circ \cdots \circ \boldsymbol{z}^{(1)}\right)(\boldsymbol{x}). \end{aligned} \tag{5.2}$$

The LocalGLMnet is defined by the additive decomposition

$$\boldsymbol{x} \mapsto \mu(\boldsymbol{x}) = g^{-1}\left(\beta_0 + \langle \boldsymbol{\beta}(\boldsymbol{x}), \boldsymbol{x} \rangle\right). \tag{5.3}$$

Extract the regression attentions $\hat{\beta}_j(x_i)$

We only provide below the code, for a better understanding of the code, see the previous chapter.

```
# define predicted layer and corresponding weights
zz <- keras_model(inputs = model_red$input, outputs = get_layer(model_red, 'attention')$output)
ww <- get_weights(model_red)
```

```
## attention weights of test samples
beta_x <- data.frame(zz %>% predict(list(TT)))
names(beta_x) <- paste0("Beta", col_namesR)
beta_x <- beta_x * as.numeric(ww[[9]])
str(beta_x)
```

```
## 'data.frame':    17866 obs. of  13 variables:
##  $ BetaAge              : num  0.314 0.38 0.375 0.314 0.331 ...
##  $ BetaGender           : num  -0.223 -0.222 -0.249 -0.25 -0.21 ...
##  $ BetaDependentChildren: num  0.1414 0.1177 0.1267 0.068 0.0651 ...
##  $ BetaDependentsOther  : num  0.144 0.201 0.203 0.208 0.235 ...
##  $ BetaWeeklyPay        : num  0.361 0.397 0.435 0.408 0.437 ...
##  $ BetaPartTimeFullTime : num  0.149 0.114 0.157 0.141 0.157 ...
##  $ BetaAccYear          : num  0.273 0.221 0.223 0.258 0.149 ...
##  $ BetaAccMonth         : num  -0.02863 0.01085 0.02659 -0.00955 0.03989 ...
##  $ BetaAccTime          : num  -0.09925 -0.111 -0.0563 -0.03637 -0.00309 ...
##  $ BetaRepDelay         : num  0.049 0.0911 0.0318 0.026 -0.0158 ...
##  $ BetaMarital1         : num  -0.203 -0.199 -0.261 -0.154 -0.212 ...
##  $ BetaMarital2         : num  -0.373 -0.37 -0.421 -0.307 -0.341 ...
##  $ BetaMarital3         : num  0.123 0.155 0.156 0.184 0.169 ...
```

Plot the covariate contributions $\hat{\beta}_j(x)x_j$

```
## merge covariates x with beta(x)
beta_x <- cbind(TT, beta_x)
```

We show the estimated regression attentions for all continuous and binary components j for 5000 randomly selected claims (we do not display all claims to not overload the plots).

```
## select at random 5000 claims to not overload plots
nsample <- 5000
set.seed(seed)
idx <- sample(x = 1:nrow(test), size = nsample)
```

```
# continuous, binary and categorical variables in col_namesR
var_cont <- c(1, 5, 7, 8, 9, 10)
var_bin <- c(2, 3, 4, 6)
```

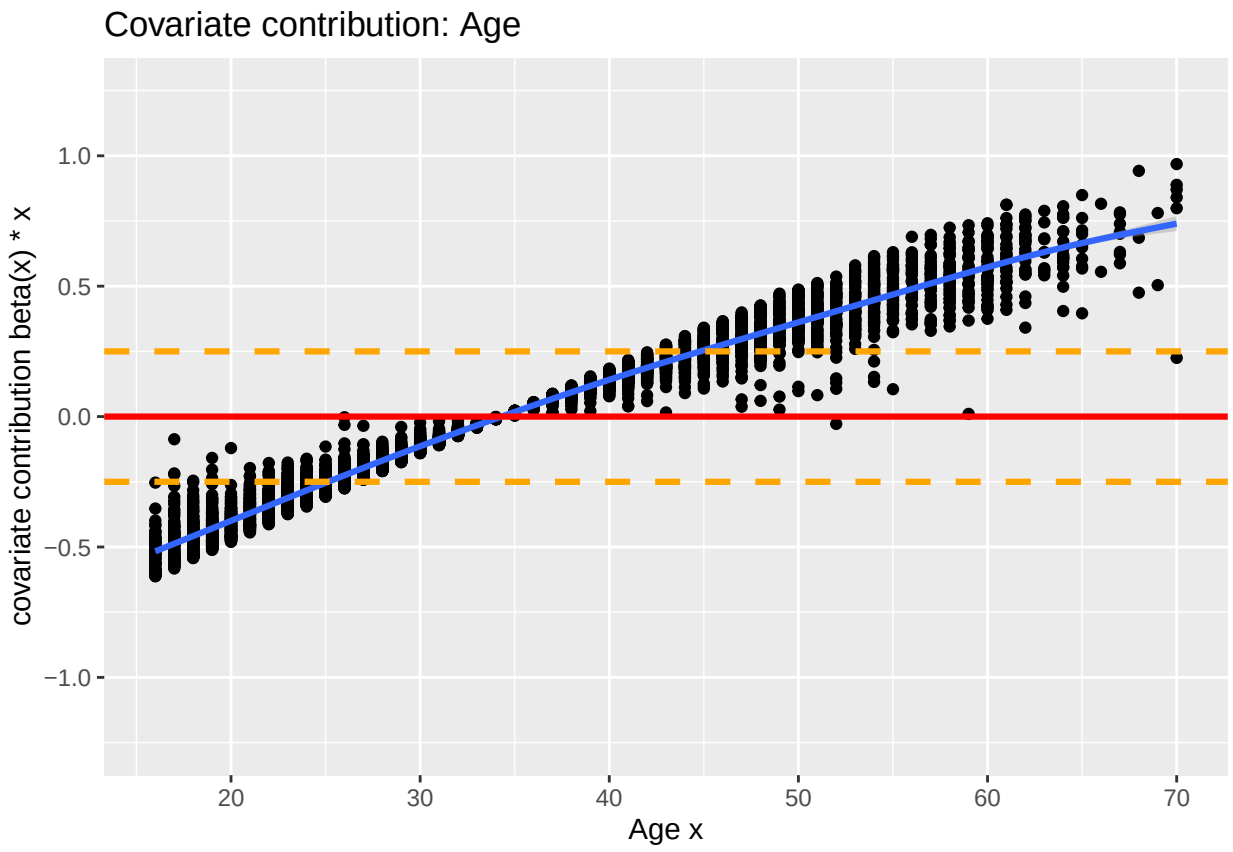
```
var_cat <- 11:13
```

```
# data for plotting
beta_x_smp <- beta_x[idx, ]
test_smp <- test[idx, ]
```

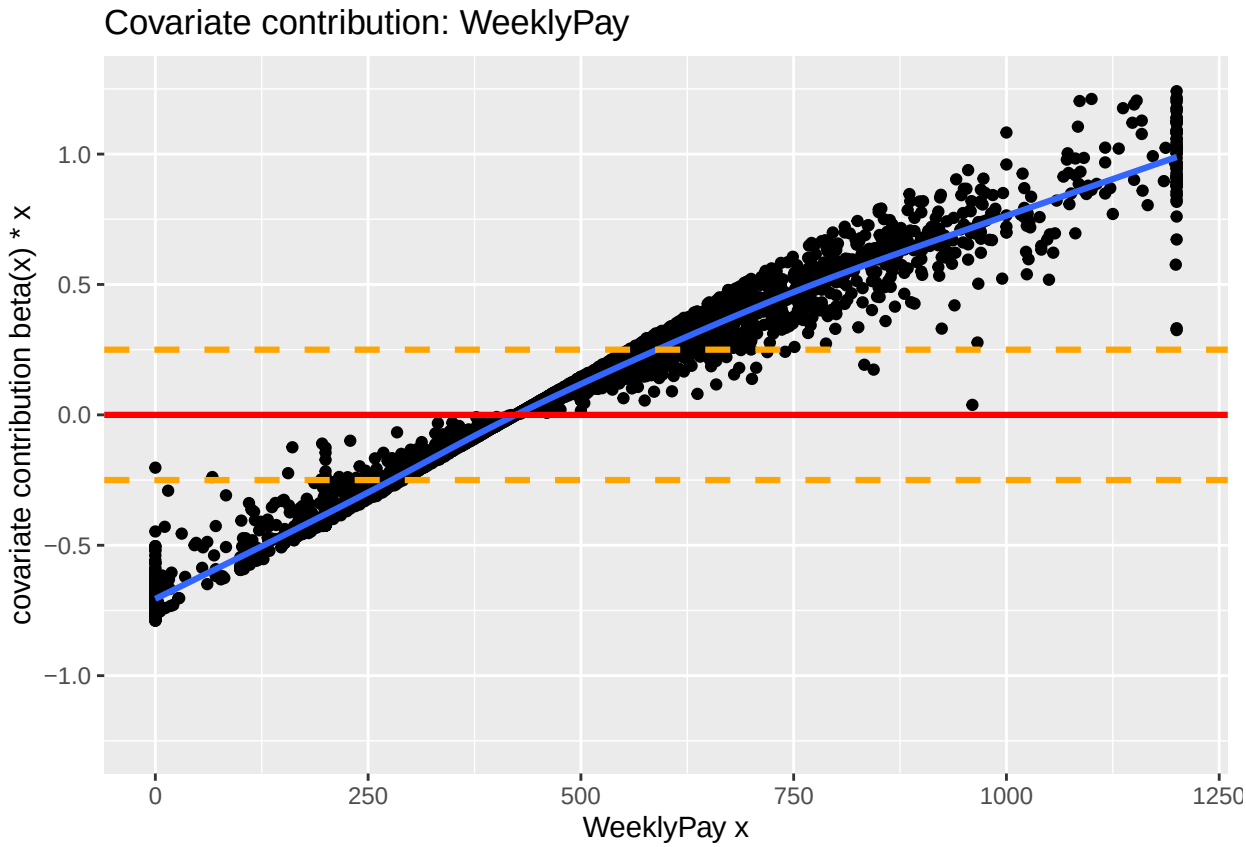
We are going to plot on the x-axis the covariates $x_{i,j}$ and on the y-axis the covariate contributions $\hat{\beta}_j(x_i)x_{i,j}$. The y-axis is the same for all covariates, and we add a spline fit. The orange line are at ± 0.25 for orientation purposes

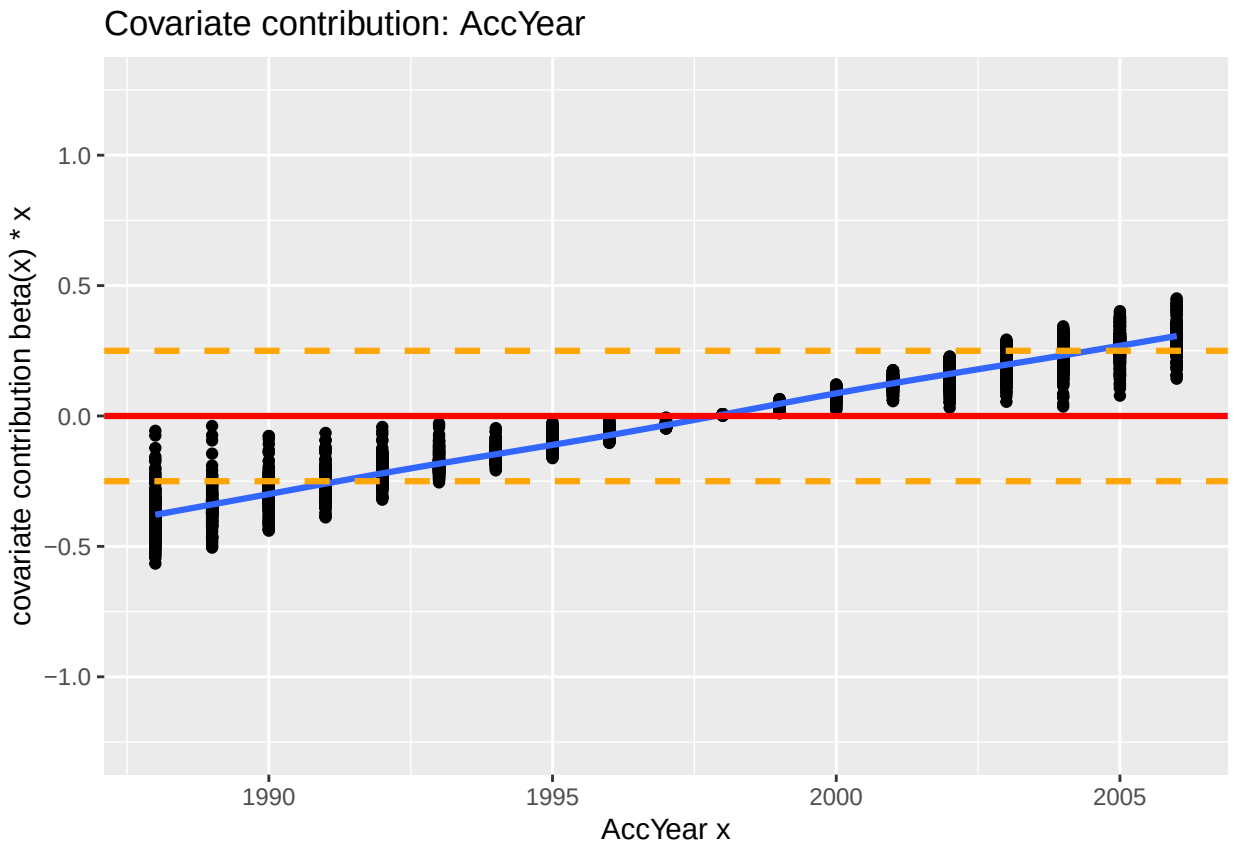
Continuous variables

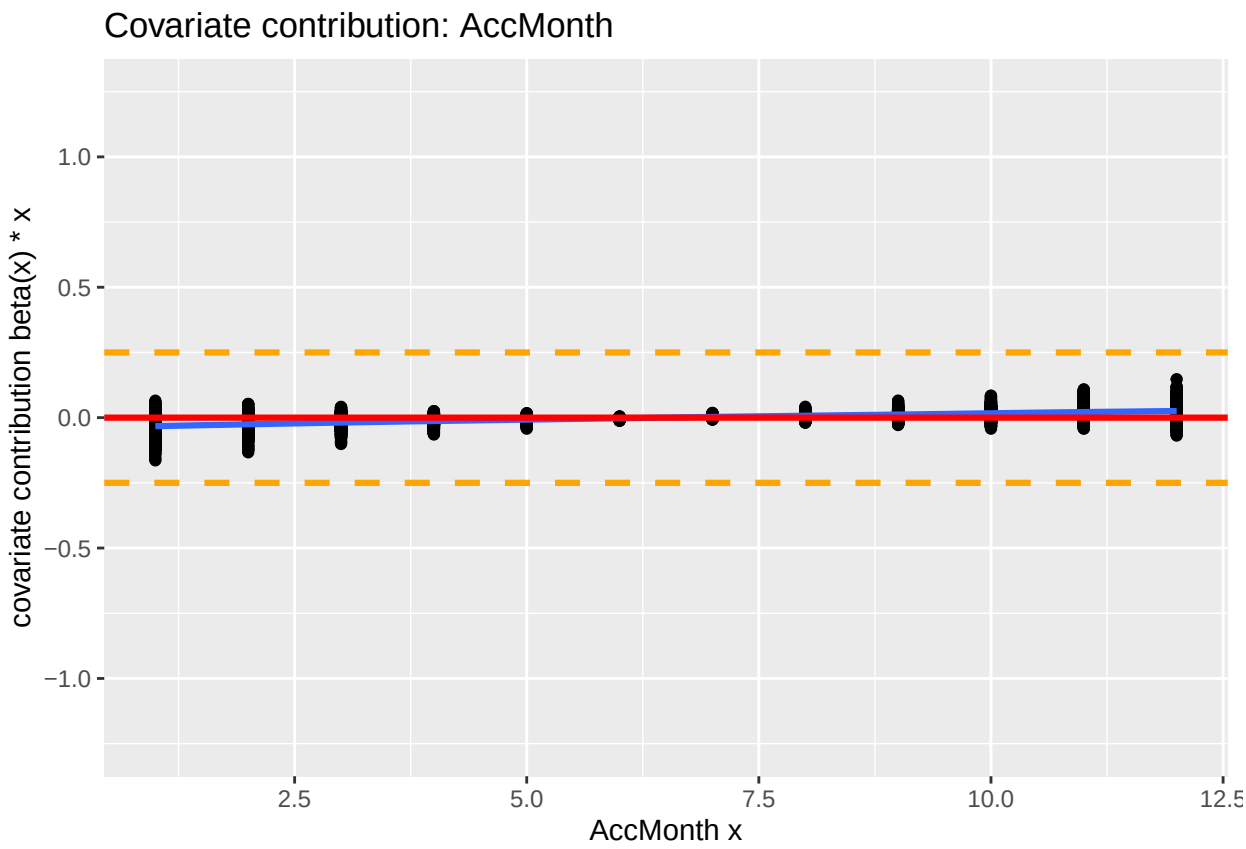
```
# Plotting for all continuous features
for (ll in 1:length(var_cont)) {
    kk <- var_cont[ll]
    dat_plt <- data.frame(var = test_smp[, col_namesR[kk]],
                          bx = beta_x_smp[, kk + length(col_featuresR)] * beta_x_smp[, kk],
                          col = rep("green", nsample))
    plt <- ggplot(dat_plt, aes(x = var, y = bx)) + geom_point() +
        geom_smooth(size = line_size) +
        geom_hline(yintercept = 0, colour = "red", size = line_size) +
        geom_hline(yintercept = c(-1, 1) / 4, colour = "orange", size = line_size, linetype = "dashed") +
        lims(y = c(-1.25, 1.25)) +
        labs(title = paste0("Covariate contribution: ", col_namesR[kk]),
             x = paste0(col_namesR[kk], " x"),
             y = "covariate contribution beta(x) * x")
    suppressMessages(print(plt))
}
```

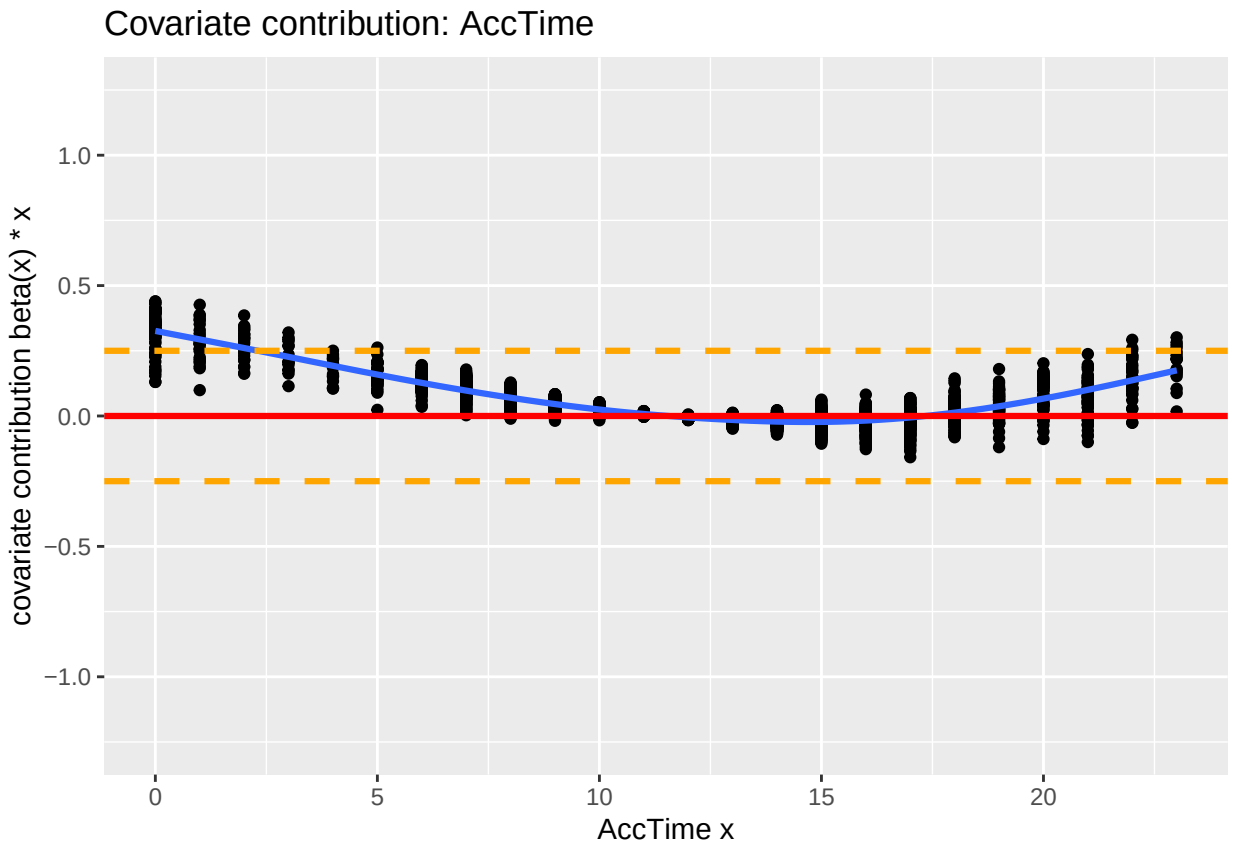


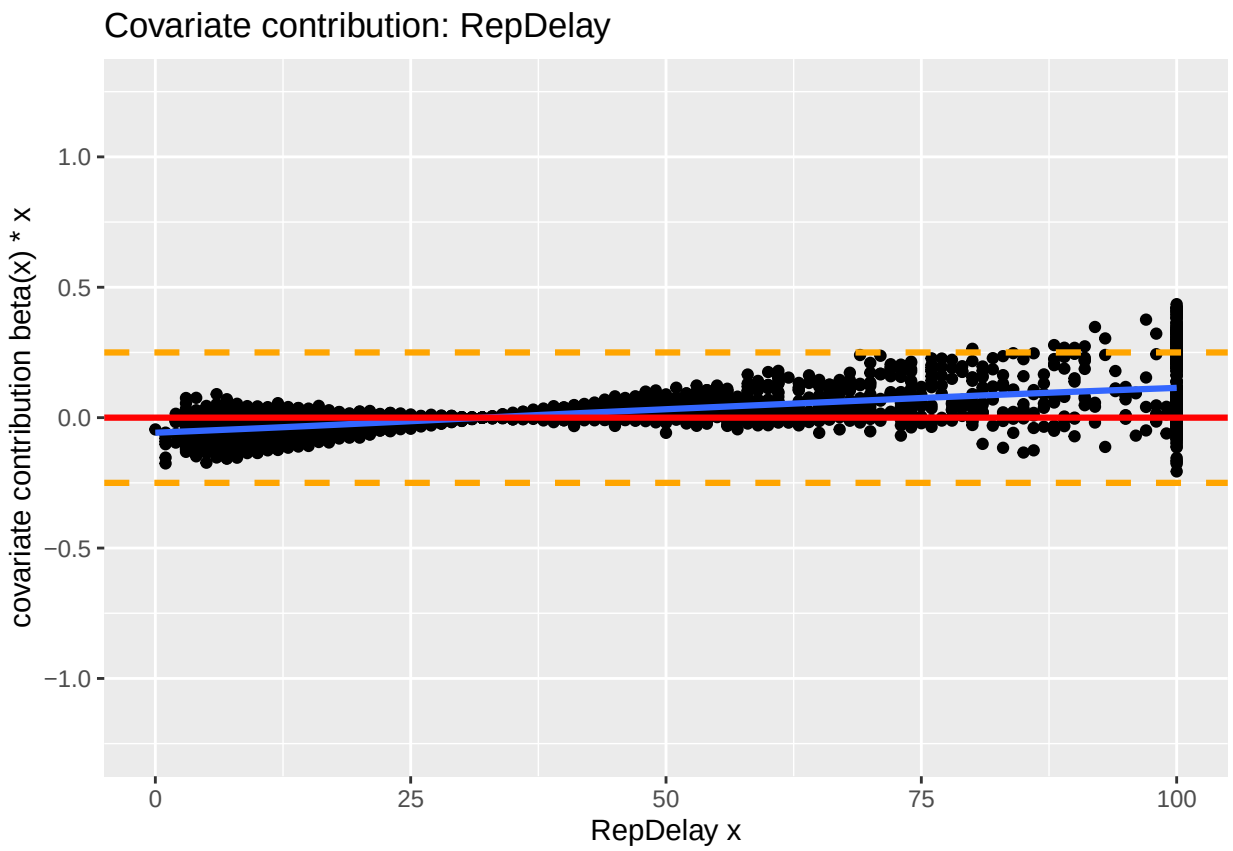
```
## Warning: Removed 4 rows containing non-finite values (stat_smooth).
## Warning: Removed 4 rows containing missing values (geom_point).
```




Covariate contribution: AccYear
covariate contribution beta(x) * x
1.0
0.5
0.0
−0.5
−1.0
1990
1995
2000
2005
AccYear x


Covariate contribution: AccMonth
covariate contribution beta(x) * x
1.0
0.5
0.0
−0.5
−1.0
2.5
5.0
7.5
10.0
12.5
AccMonth x


Covariate contribution: AccTime
covariate contribution beta(x) * x
1.0
0.5
0.0
−0.5
−1.0
0
5
10
15
20
AccTime x


Covariate contribution: RepDelay
covariate contribution beta(x) * x
1.0
0.5
0.0
−0.5
−1.0
0
25
50
75
100
RepDelay x

$\Rightarrow$ The figures above show the covariate contributions $\hat{\beta}_j(x)x_{i,j}$ of the continuous variables Age, WeeklyPay, AccYear, AccMonth, AccTime, RepDelay. We see that claim amounts are increasing in most of these variables, and AccTime shows that claims are more expensive during nights. The AccMonth contribution is not so clear, so further investigations are needed.

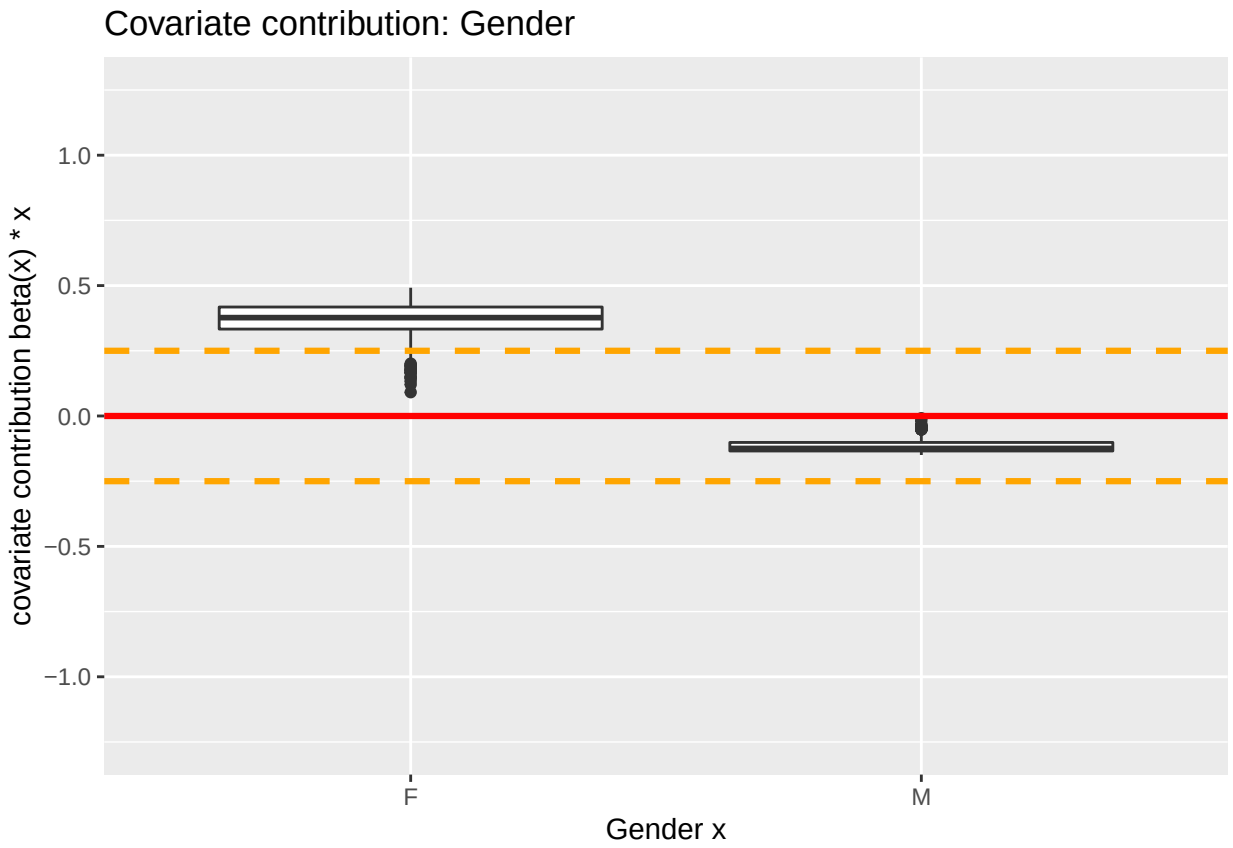
The blue curve shows a spline fit, and the more volatility is observed around this spline the more strong the interactions are in $\hat{\beta}_j(x_i)x_{i,j}$.

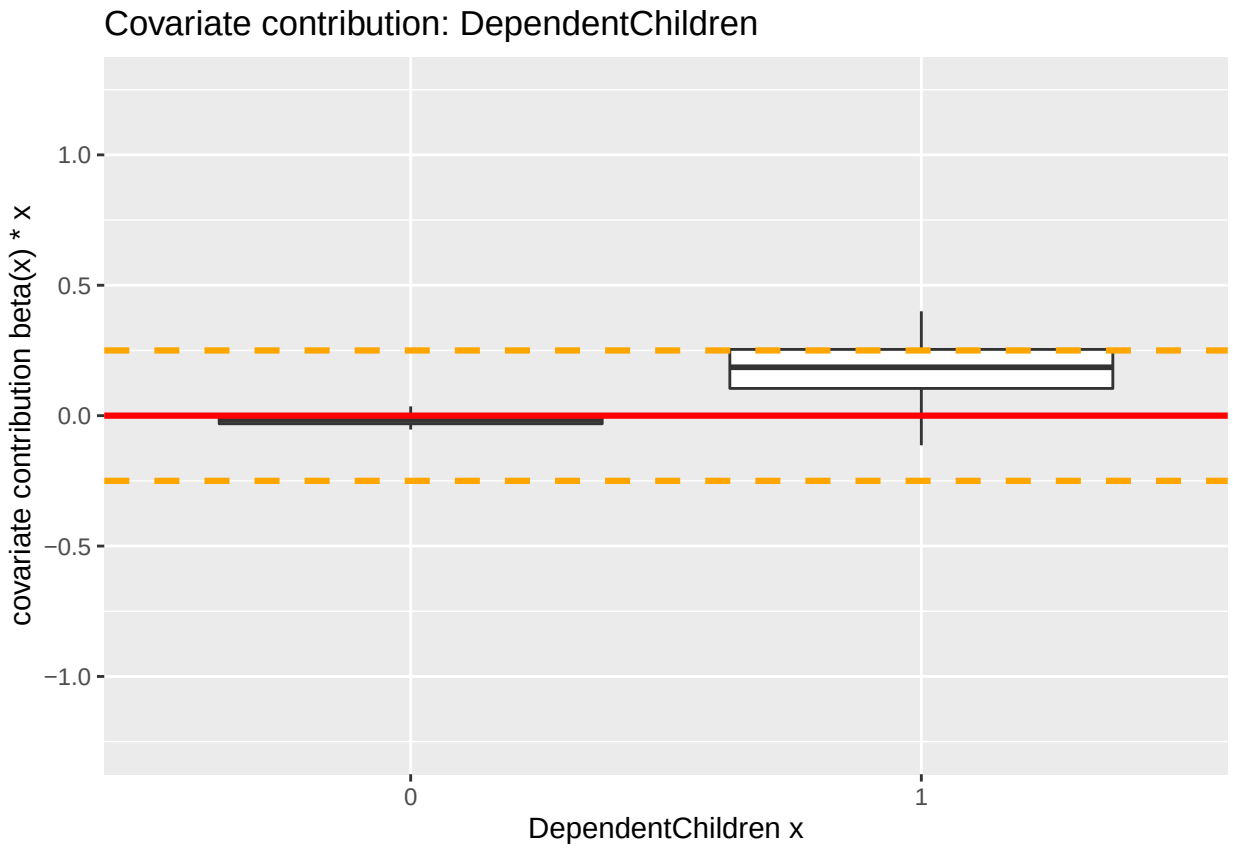
Exercise: Do change the numbers of points shown in chart, do see some more interesting patterns?

Exercise: The conclusion for the AccMonth variable is more involved. Have a careful look at it and try out what could be done to improve it.

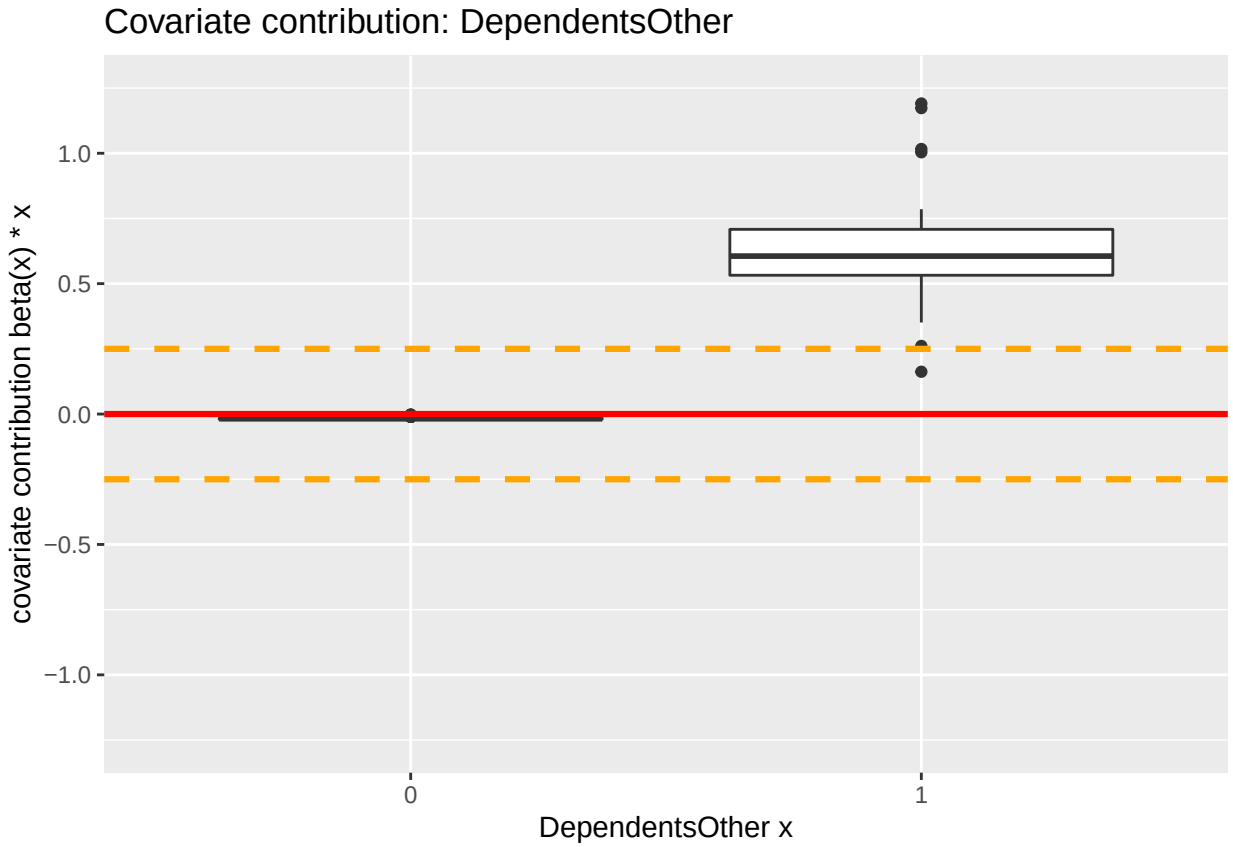
Binary variables

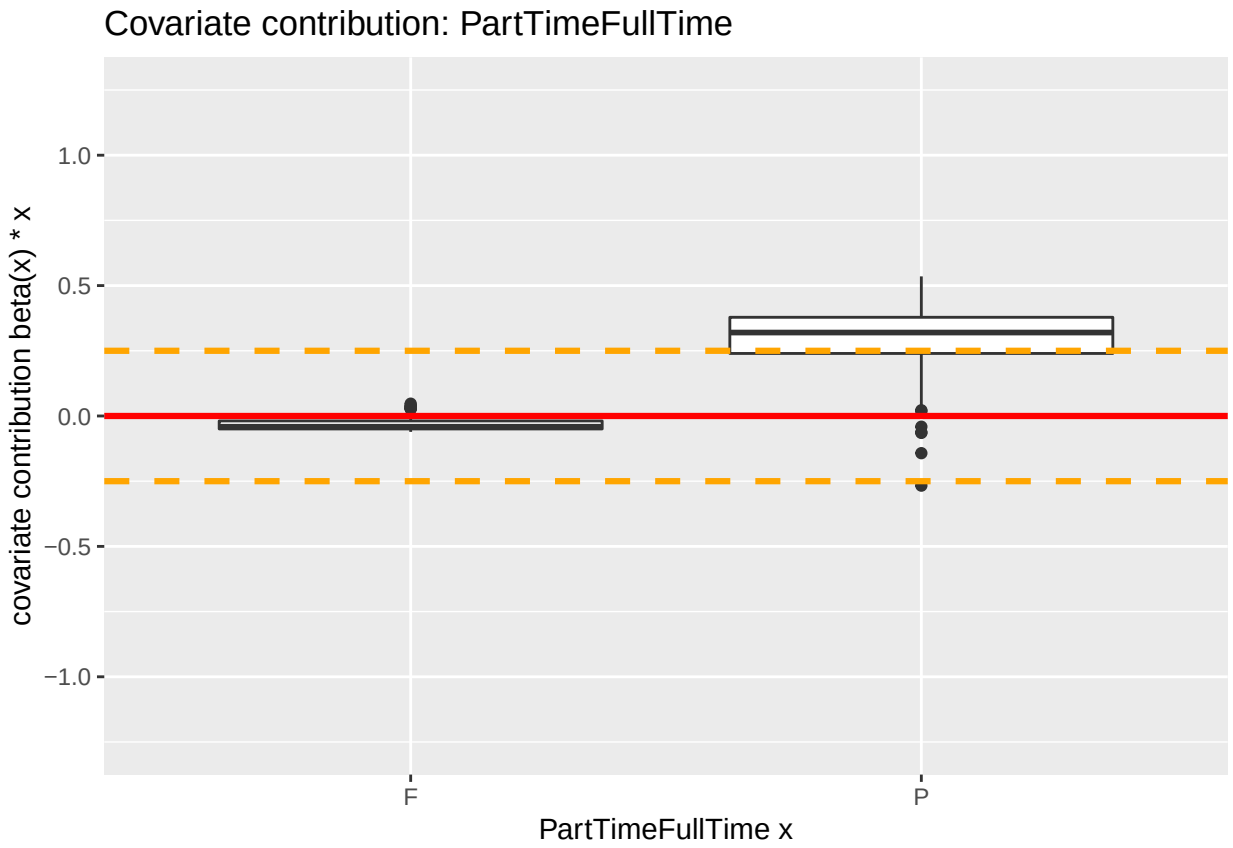
```
# Plotting for all binary features
for (ll in 1:length(var_bin)) {
    kk <- var_bin[ll]
    dat_plt <- data.frame(var = factor(test_smp[, col_namesR[kk]]),
                          bx = beta_x_smp[, kk + length(col_featuresR)] * beta_x_smp[, kk],
                          col = rep("green", nsample))
    plt <- ggplot(dat_plt, aes(x = var, y = bx)) + geom_boxplot() +
        geom_hline(yintercept = 0, colour = "red", size = line_size) +
        geom_hline(yintercept = c(-1,1)/4, colour = "orange", size = line_size, linetype = "dashed") +
        lims(y = c(-1.25, 1.25)) +
        labs(title = paste0("Covariate contribution: ", col_namesR[kk]),
             x = paste0(col_namesR[kk], " x"),
             y = "covariate contribution beta(x) * x")
    suppressMessages(print(plt))
}
```


Covariate contribution: Gender
covariate contribution beta(x) * x
1.0
0.5
0.0
−0.5
−1.0
F
M
Gender x


Covariate contribution: DependentChildren
covariate contribution beta(x) * x
1.0
0.5
0.0
−0.5
−1.0
0
1
DependentChildren x

```
## Warning: Removed 1 rows containing non-finite values (stat_boxplot).
```





⇒ The figures above show the covariate contributions $\hat{\beta}_j(x_i)x_{i,j}$ of the binary variables Gender, DependentChildren, DependentsOther, PartTimeFullTime. The difference in the binary variables are clearly smaller than for the continuous ones, which is also reflected by the variable importance plot. DependentChildren and DependentsOther only has a small volume in level 1, and these levels seem mainly driven by interactions (because of the large boxes in the boxplot).

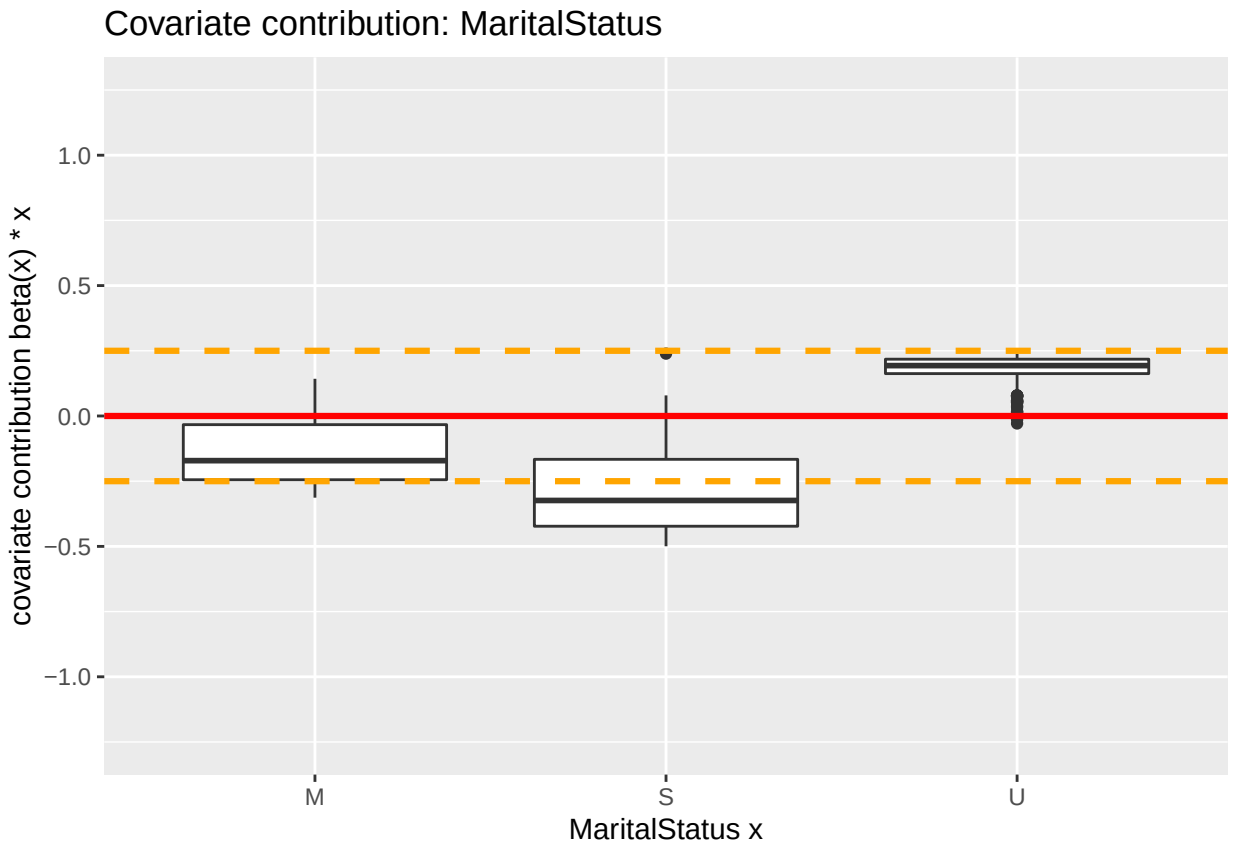
Categorical variable

```
# data preparation for plotting
beta_xx <- cbind(test$MaritalStatus, beta_x[, 11:13] * beta_x[, 24:26])
beta_xx$BetaMarital <- rowSums(beta_xx[, -1])
beta_xx <- beta_xx[, c(1, ncol(beta_xx))]
names(beta_xx) <- c("MaritalStatus", "BetaMarital")
str(beta_xx)
```

```
## 'data.frame':    17866 obs. of  2 variables:
##  $ MaritalStatus: Factor w/ 3 levels "M","S","U": 2 1 1 2 2 2 3 1 2 2 ...
##  $ BetaMarital  : num  -0.373 -0.199 -0.261 -0.307 -0.341 ...
```

```
namesCat <- "MaritalStatus"
dat_plt <- beta_xx[idx, ]
```

```
ggplot(dat_plt, aes(x = MaritalStatus, y = BetaMarital)) + geom_boxplot() +
     geom_hline(yintercept = 0, colour = "red", size = line_size) +
     geom_hline(yintercept = c(-1, 1) / 4, colour = "orange", size = line_size, linetype = "dashed") +
     lims(y = c(-1.25, 1.25)) +
     labs(title = paste0("Covariate contribution: ", namesCat),
          x = paste0(namesCat, " x"),
          y = "covariate contribution beta(x) * x")
```



$\Rightarrow$ Finally, the figure above gives the covariate contributions $\hat{\beta}_j(x_i)$ of the categorical variable MaritalStatus. Note that we do not multiply with $x_{i,j}$ in this case, because these covariate components just correspond to the 0-1's from one-hot encoding. This plot now also illustrates why we do not consider dummy coding, namely, if we would use dummy coding, e.g., choosing level single (S) as reference level, then the corresponding $x_j = 0$, and we could not model interactions for this level because it would imply that $\hat{\beta}_j(x_i) = 0$, and all single persons would share the same value (being the bias estimate $\hat{\beta}_0$).

LocalGLMnet: Interactions

We are going to examine point (3) of the interpretation for LocalGLMnet.

(3) Property $\beta_j(\boldsymbol{x}) = \beta_j(x_j)$ says that we have a term $\beta_j(x_j)x_j$ that does not interact with other terms. Sensitivities of $\beta_j(\boldsymbol{x})$ in the components of $\boldsymbol{x}$ can be obtained by the gradient

$$\nabla\beta_j(\boldsymbol{x}) = \left(\partial_{x_1}\beta_j(\boldsymbol{x}), \ldots, \partial_{x_q}\beta_j(\boldsymbol{x})\right)^\top \in \mathbb{R}^q. \tag{5.5}$$

The j-th component $\partial_{x_j}\beta_j(\boldsymbol{x})$ of this gradient $\nabla\beta_j(\boldsymbol{x})$ explores whether we have a linear term in x_j, and the components different from j quantify the interactions.

We analyze the interaction terms by studying the sensitivities $\partial_{x_k}\beta_j(x)$ given in (5.5). A zero term $\partial_{x_k}\beta_j(x) \approx 0$ for $k = j$ supports linearity, and $\partial_{x_k}\beta_j(x) \neq 0$ for $k \neq j$ means that x_k and x_j interact.

The code shown below to plot the interactions is not intuitive and straightforward. You can skip the code details and focus on the understanding of the charts and the correspondings conclusions to be drawn.

Set parameters

```
# limits for plotting (y-axis)
ax_limit <- c(-.6, .6)

# number of support points
n_points <- 100
```

Model

```
zz <- keras_model(inputs = model_red$input, outputs = get_layer(model_red, 'attention')$output)
ww <- get_weights(zz)
```

```
Input <- layer_input(shape = c(qqq[1]), dtype = 'float32', name = 'Design2')

Attention <- Input %>%
          layer_dense(units = qqq[2], activation = 'tanh', name = 'FNLayer1') %>%
          layer_dense(units = qqq[3], activation = 'tanh', name = 'FNLayer2') %>%
          layer_dense(units = qqq[4], activation = 'tanh', name = 'FNLayer3') %>%
          layer_dense(units = qqq[1], activation = 'linear', name = 'attention')

model_int <- keras_model(inputs = c(Input), outputs = c(Attention))
```

```
set_weights(model_int, ww)
```

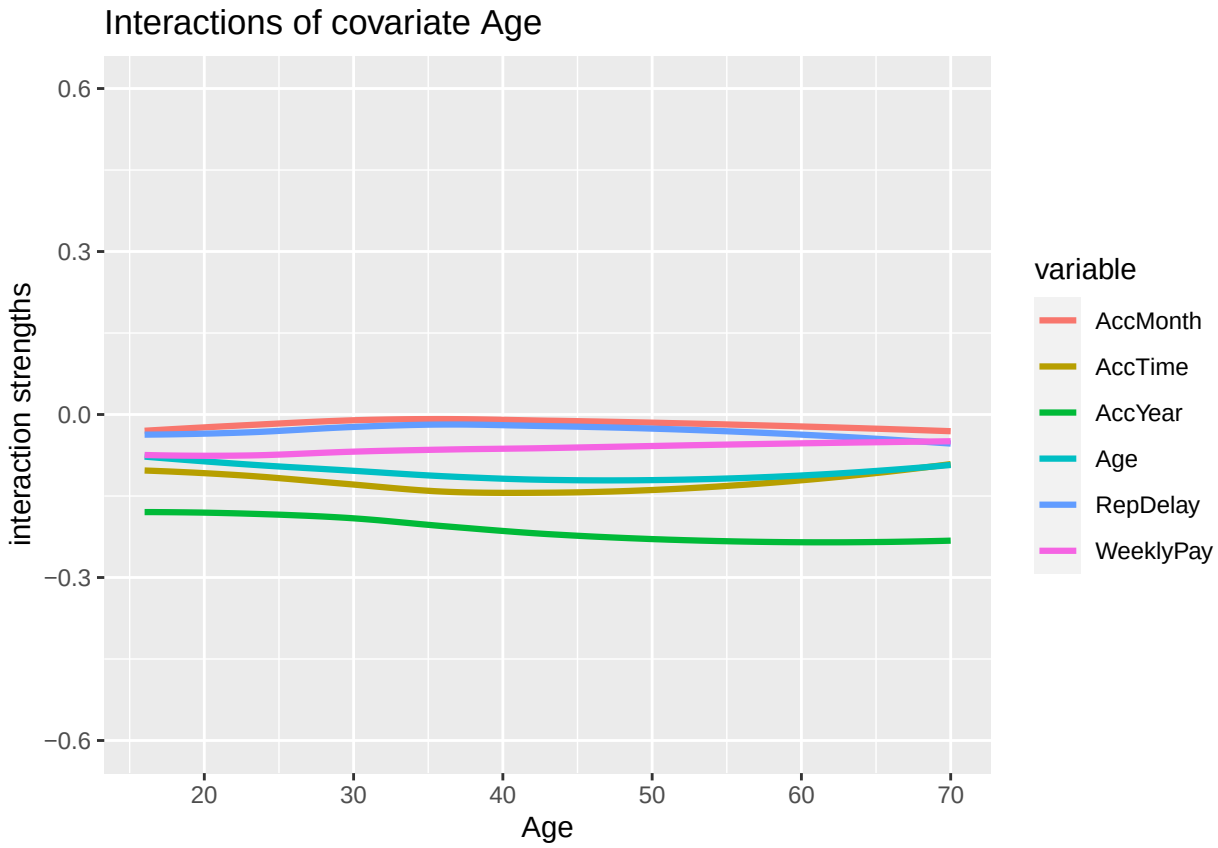
The figures below show a spline fit to these partial derivatives to all claims $1 \leq i \leq n$ for the continuous covariate components.

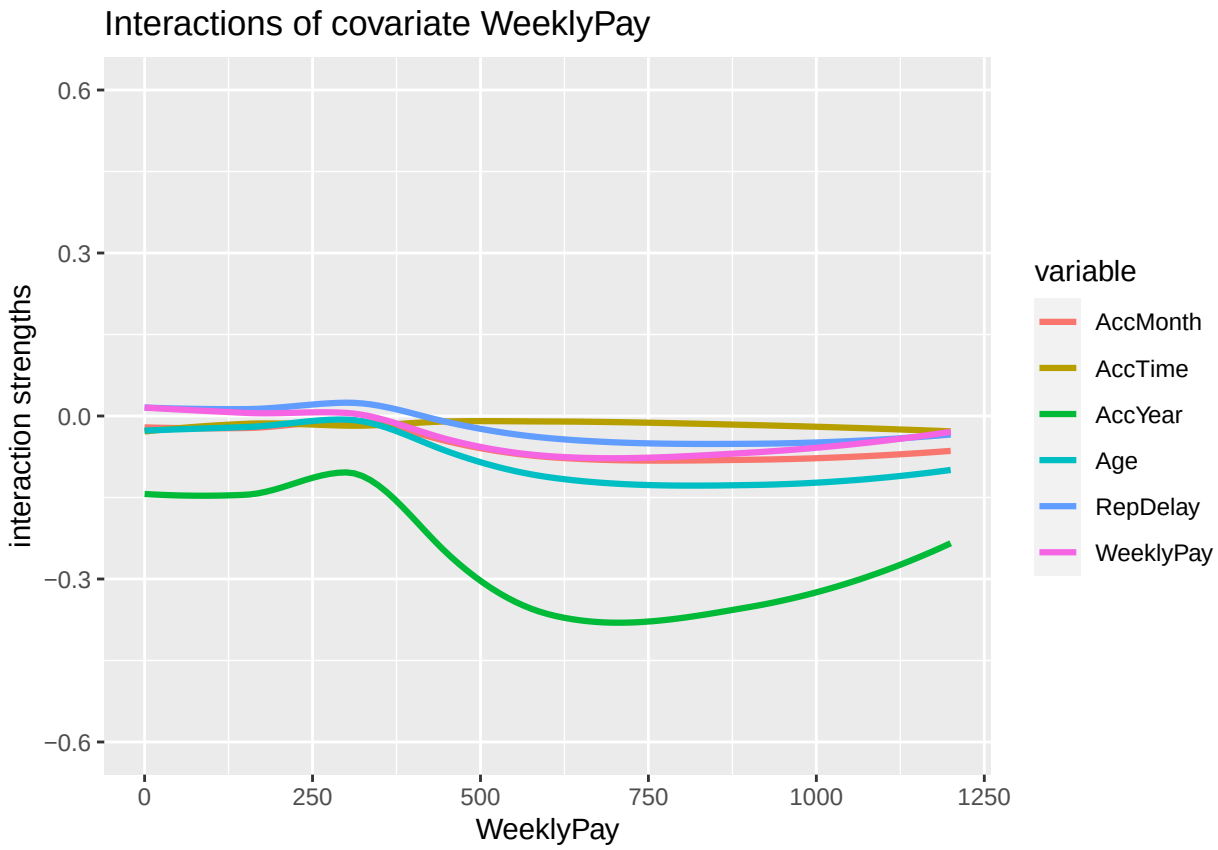
Continuous-Continuous

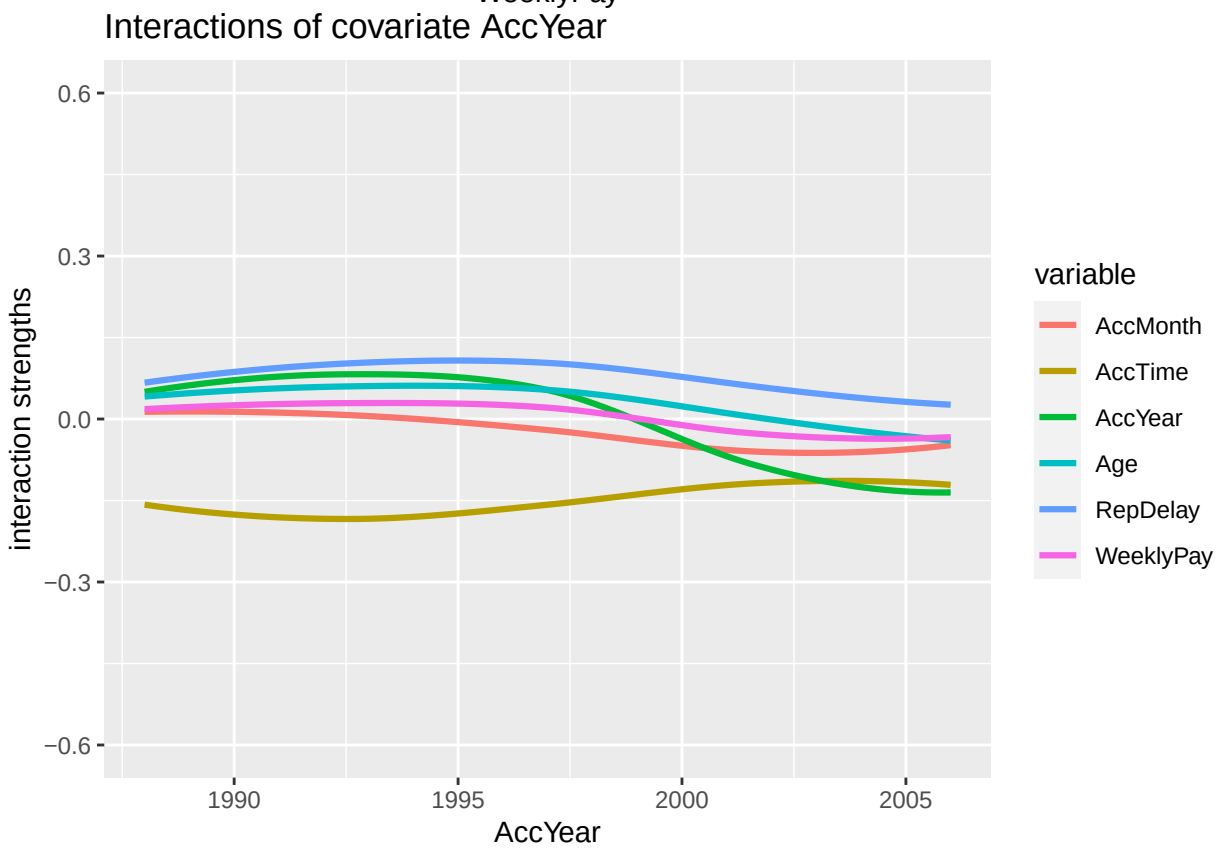
```
col_names_cont <- col_namesR[var_cont]
n_col_names <- length(col_names_cont)
```

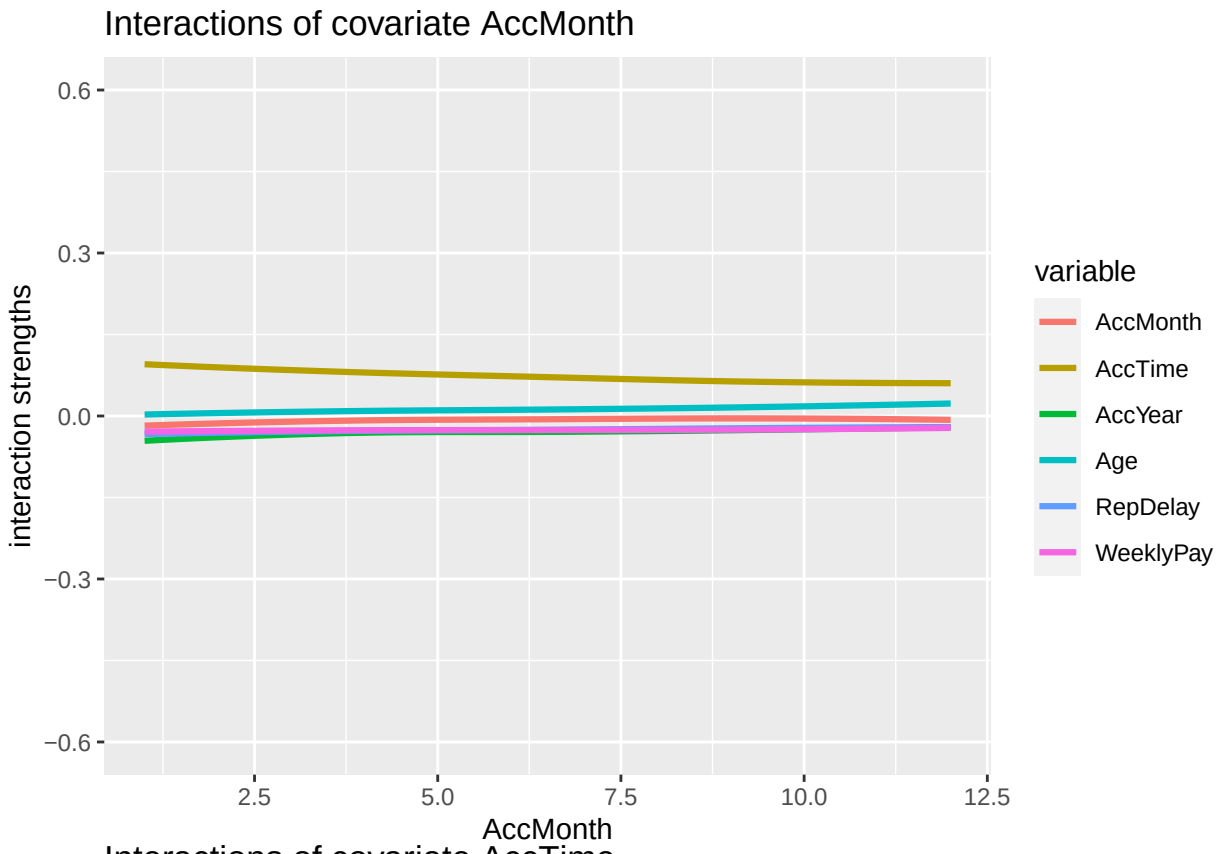
```
for (jj in 1:length(col_names_cont)) {
    beta_j <- Attention %>% layer_lambda(function(x) x[, var_cont[jj]])
    model_grad1 <- keras_model(inputs = c(Input), outputs = c(beta_j))
    grad <- beta_j %>% layer_lambda(function(x) k_gradients(model_grad1$outputs, model_grad1$inputs))

    model_grad2 <- keras_model(inputs = c(Input), outputs = c(grad))
    grad_beta <- data.frame(model_grad2 %>% predict(as.matrix(TT)))
    grad_beta <- grad_beta[, var_cont]
    names(grad_beta) <- paste0("Grad", col_names_cont)

    beta_x <- cbind(test[, col_names_cont[jj]], grad_beta)
    names(beta_x)[1] <- col_names_cont[jj]
    beta_x <- beta_x[order(beta_x[, 1]), ]

    rr <- range(beta_x[, 1])
    xx <- rr[1] + (rr[2] - rr[1]) * 0:n_points / n_points
    yy <- array(NA, c(n_points + 1, n_col_names))
    for (kk in 1:length(var_cont)) {
        yy[, kk] <- predict(locfit(beta_x[, kk + 1]~ beta_x[, 1], alpha = 0.7, deg = 2), newdata = xx)
    }
```

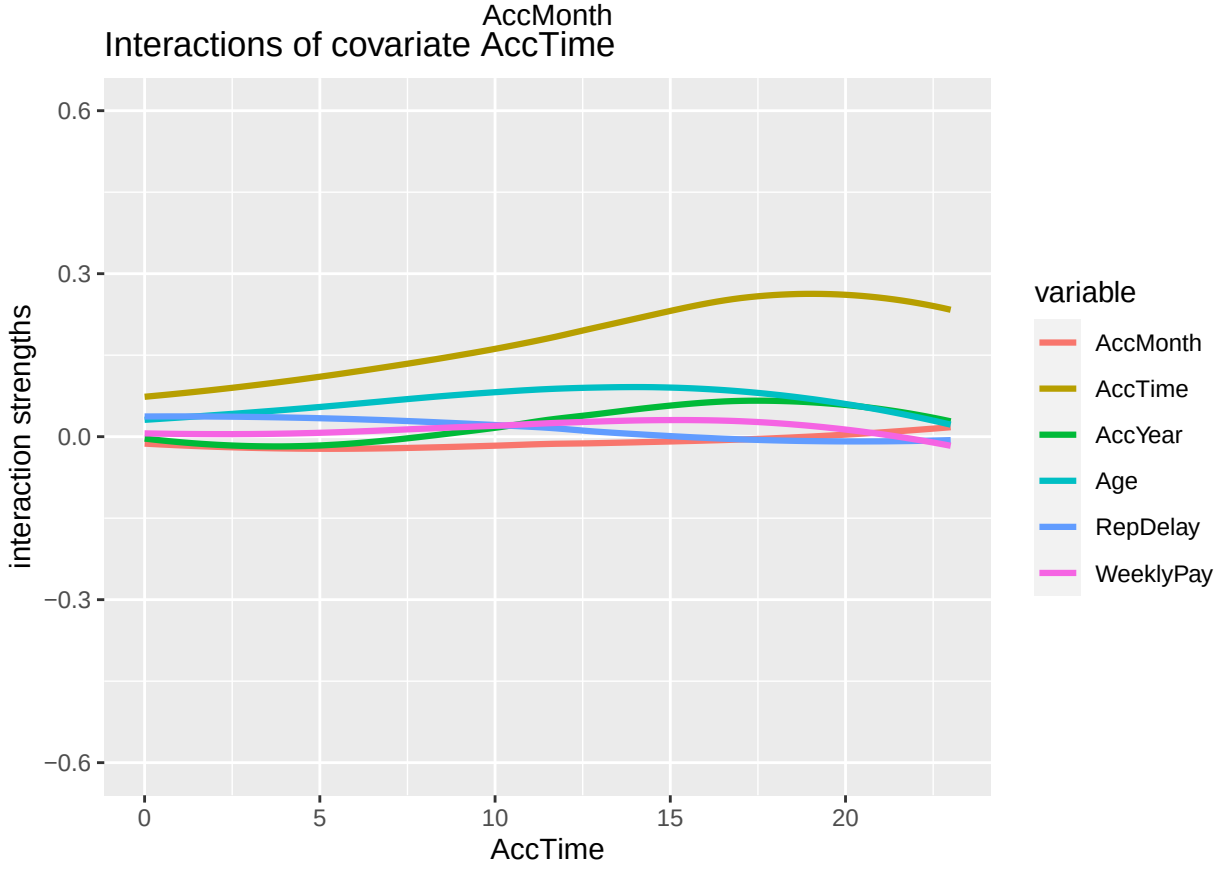
```
    dat_plt <- data.frame(xx, yy)
    colnames(dat_plt) <- c("x", col_names_cont)
    dat_plt <- dat_plt %>% gather(key = "variable", value = "value", -x)

    plt <- ggplot(dat_plt, aes(x = x, y = value)) +
        geom_line(aes(color = variable), size = line_size) +
        ylim(ax_limit) +
        labs(title = paste0("Interactions of covariate ", col_names_cont[jj]),
             x = col_names_cont[jj],
             y = "interaction strengths")
    print(plt)
}
```

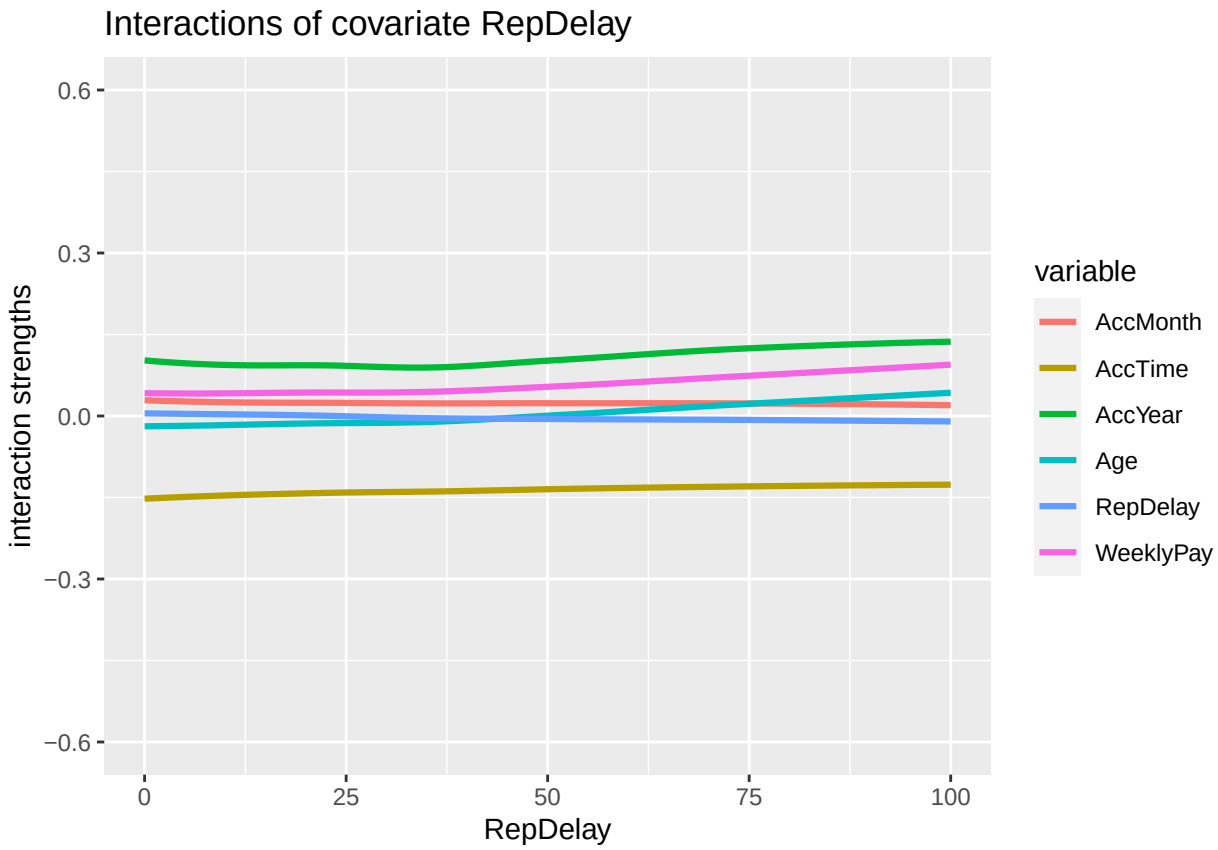



Interactions of covariate WeeklyPay
interaction strengths
0.6
0.3
0.0
−0.3
−0.6
0
250
500
750
1000
1250
WeeklyPay
variable
AccMonth
AccTime
AccYear
Age
RepDelay
WeeklyPay


Interactions of covariate AccYear
interaction strengths
0.6
0.3
0.0
−0.3
−0.6
1990
1995
2000
2005
AccYear
variable
AccMonth
AccTime
AccYear
Age
RepDelay
WeeklyPay


Interactions of covariate AccMonth
interaction strengths
0.6
0.3
0.0
−0.3
−0.6
2.5
5.0
7.5
10.0
12.5
AccMonth
variable
AccMonth
AccTime
AccYear
Age
RepDelay
WeeklyPay


Interactions of covariate AccTime
interaction strengths
0.6
0.3
0.0
−0.3
−0.6
0
5
10
15
20
AccTime
variable
AccMonth
AccTime
AccYear
Age
RepDelay
WeeklyPay



$\Rightarrow$ We do not observe any zero terms $\partial_{x_k}\beta_j(x) \approx 0$, saying, that non of these components can be modeled by a linear term. Moreover, we observe that most continuous variables interact, e.g., Age, WeeklyPay and AccYear interact, but also RepDelay, AccTime and AccYear interact, as well as Age with AccTime. Only, AccMonth does not seem to have major interactions with other variables.

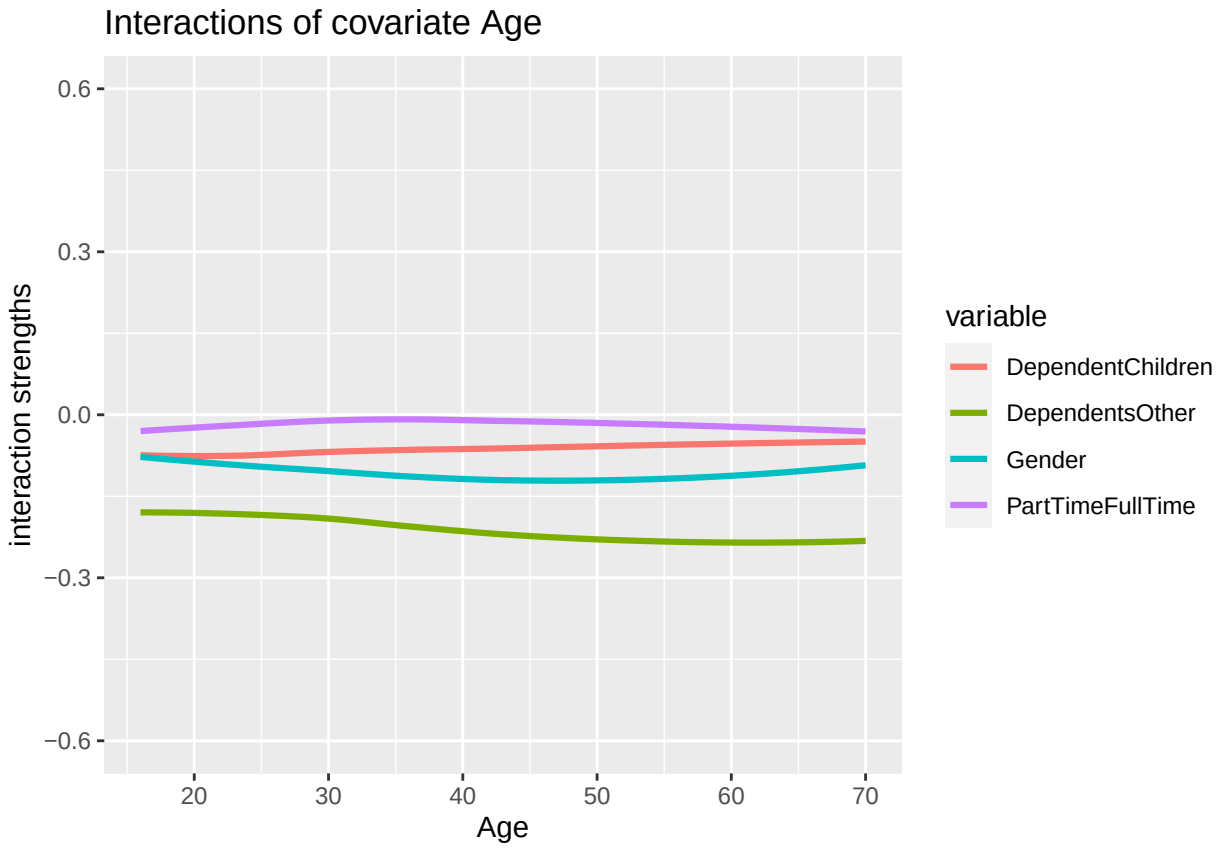
Continuous-Binary

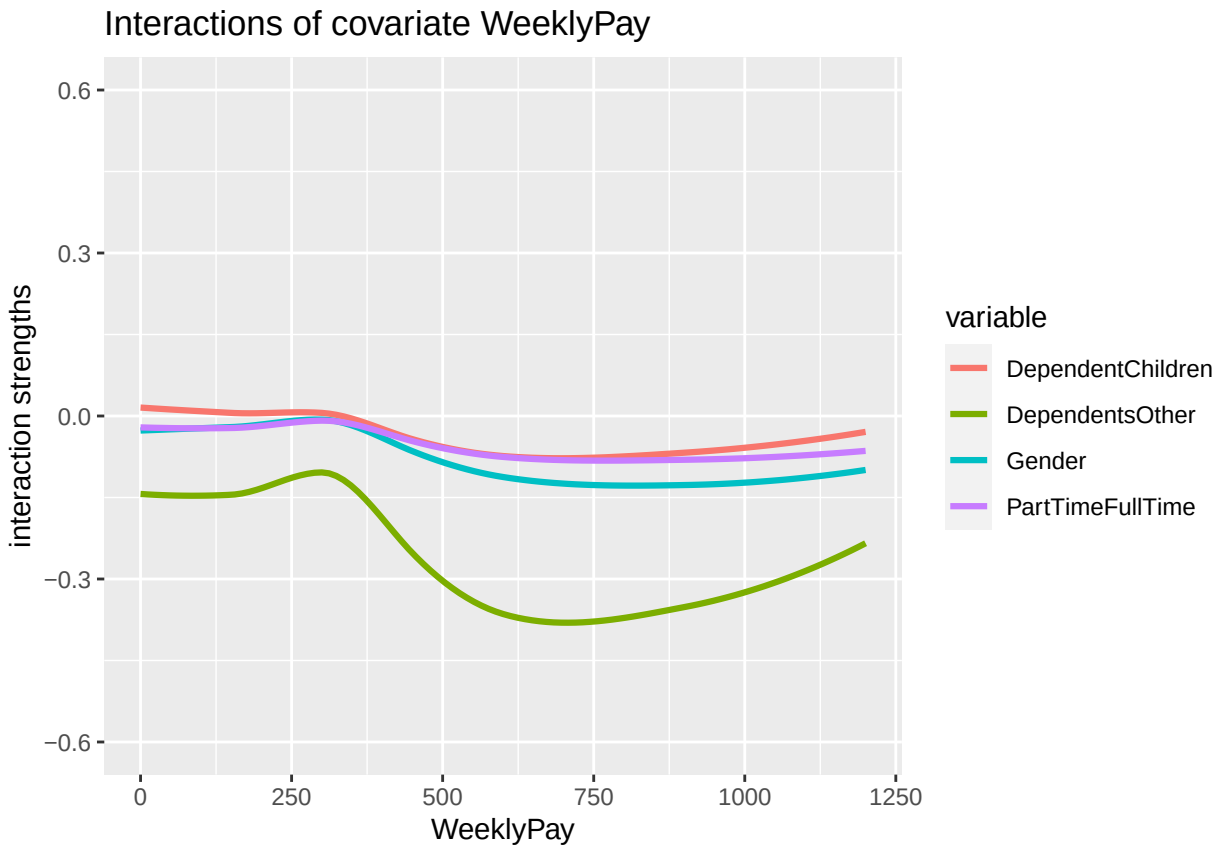
The figures below show the interactions of the continuous variables with the binary variables.

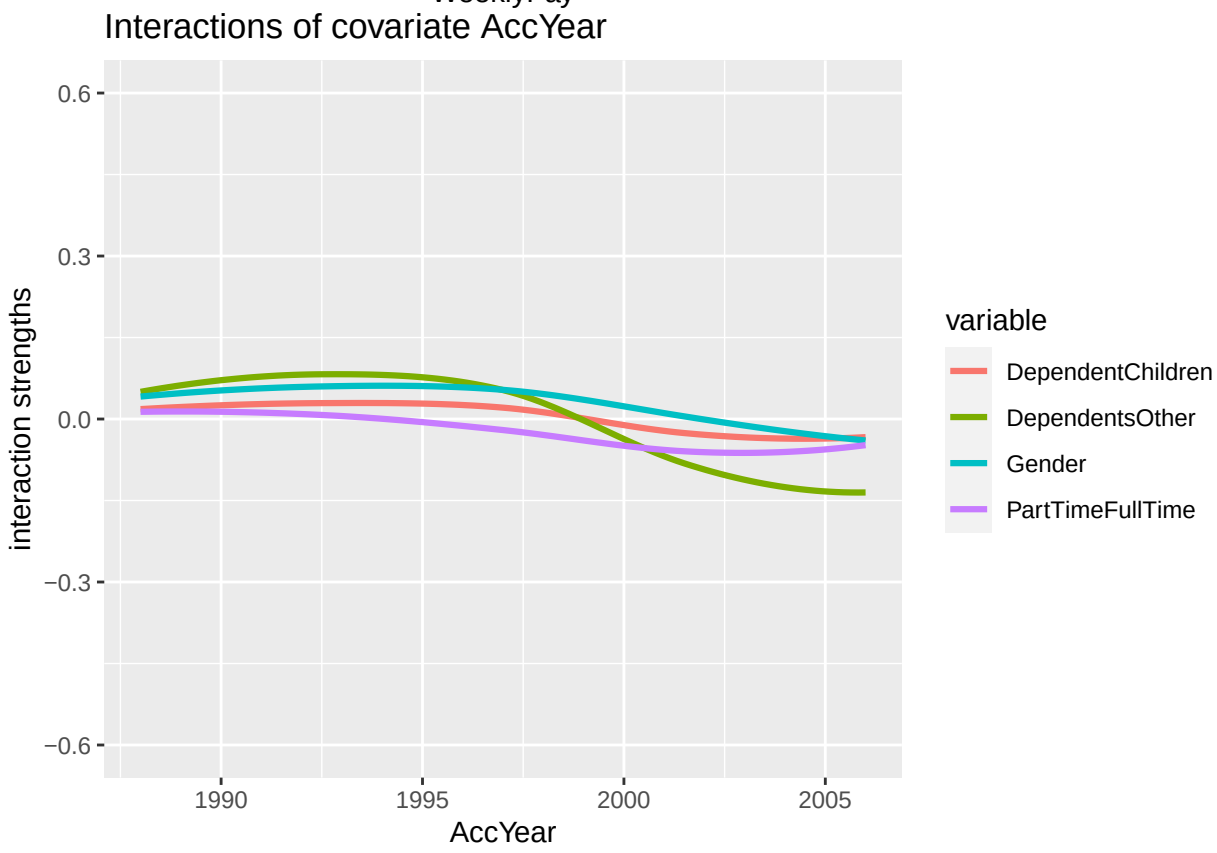
```
col_names_cont <- col_namesR[var_cont]
col_names_bin <- col_namesR[var_bin]
n_col_names <- length(col_names_bin)
```

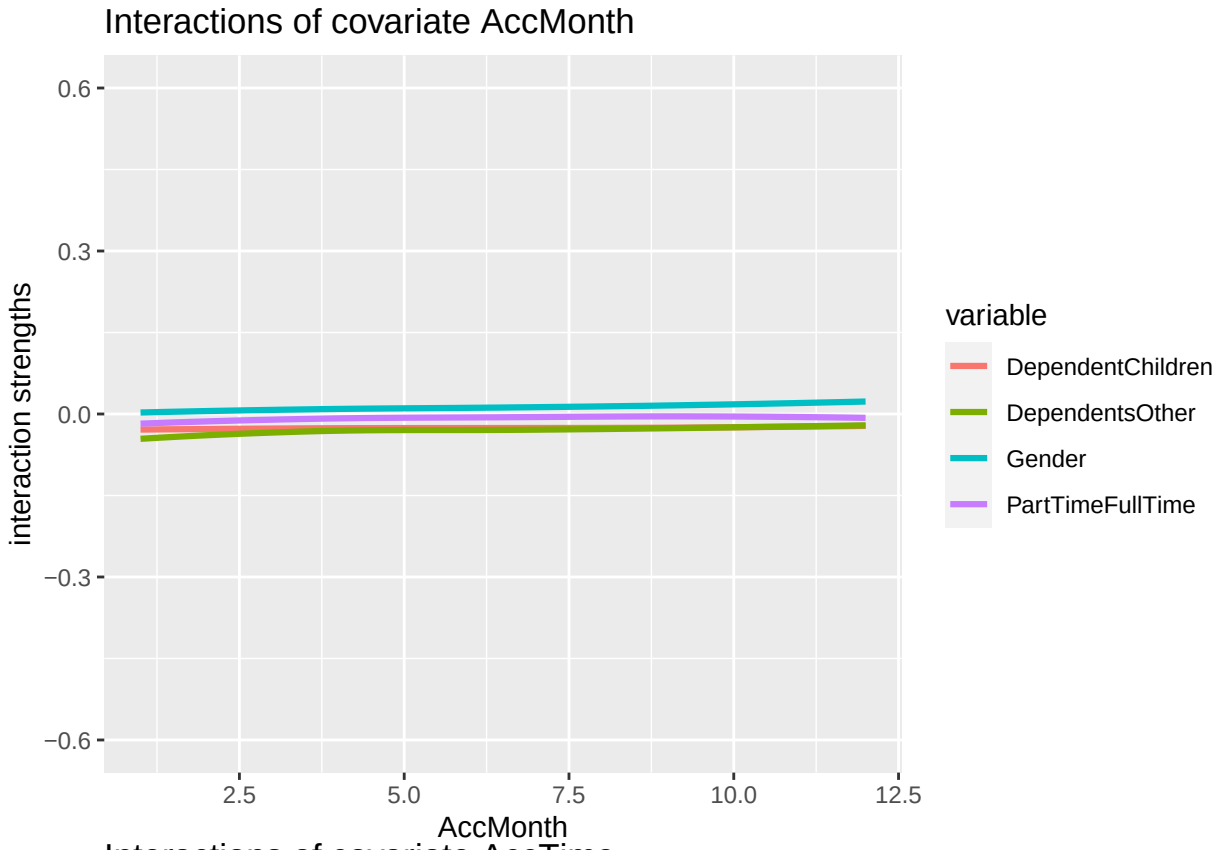
```
for (jj in 1:length(col_names_cont)) {
    beta_j <- Attention %>% layer_lambda(function(x) x[, var_cont[jj]])
    model_grad1 <- keras_model(inputs = c(Input), outputs = c(beta_j))
    grad <- beta_j %>% layer_lambda(function(x) k_gradients(model_grad1$outputs, model_grad1$inputs))

    model_grad2 <- keras_model(inputs = c(Input), outputs = c(grad))
    grad_beta <- data.frame(model_grad2 %>% predict(as.matrix(TT)))
    grad_beta <- grad_beta[, var_cont]
    names(grad_beta) <- paste0("Grad", col_names_cont)

    beta_x <- cbind(test[, col_names_cont[jj]], grad_beta)
    names(beta_x)[1] <- col_names_cont[jj]
    beta_x <- beta_x[order(beta_x[, 1]), ]

    rr <- range(beta_x[, 1])
```

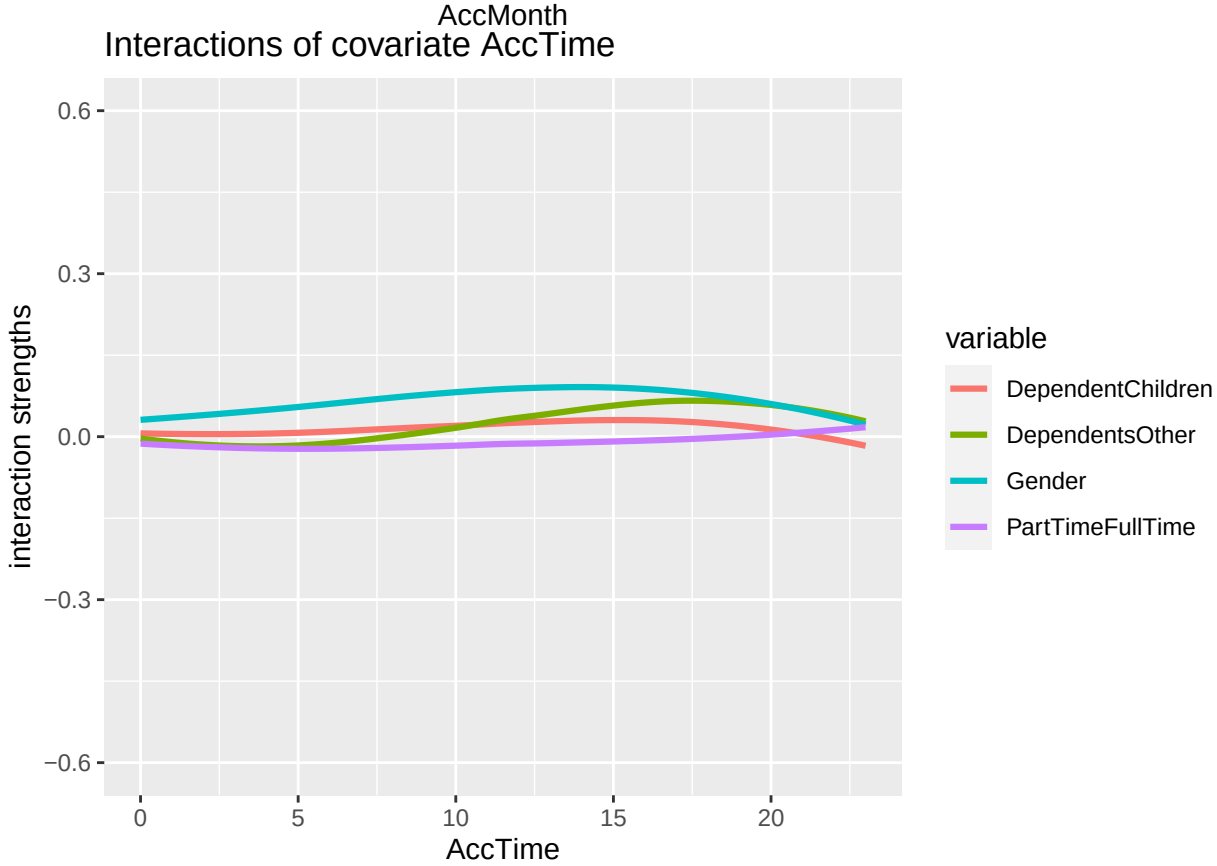
```
    xx <- rr[1] + (rr[2] - rr[1]) * 0:n_points/n_points
    yy <- array(NA, c(n_points + 1, n_col_names))
    for (kk in 1:length(var_bin)) {
        yy[, kk] <- predict(locfit(beta_x[, kk + 1]~ beta_x[, 1] , alpha = 0.7, deg = 2), newdata = xx)
    }

    dat_plt <- data.frame(xx, yy)
    colnames(dat_plt) <- c("x", col_names_bin)
    dat_plt <- dat_plt %>% gather(key = "variable", value = "value", -x)

    plt <- ggplot(dat_plt, aes(x = x, y = value)) +
        geom_line(aes(color = variable), size = line_size) +
        ylim(ax_limit) +
        labs(title = paste0("Interactions of covariate ", col_names_cont[jj]),
             x = col_names_cont[jj],
             y = "interaction strengths")
    print(plt)
}
```

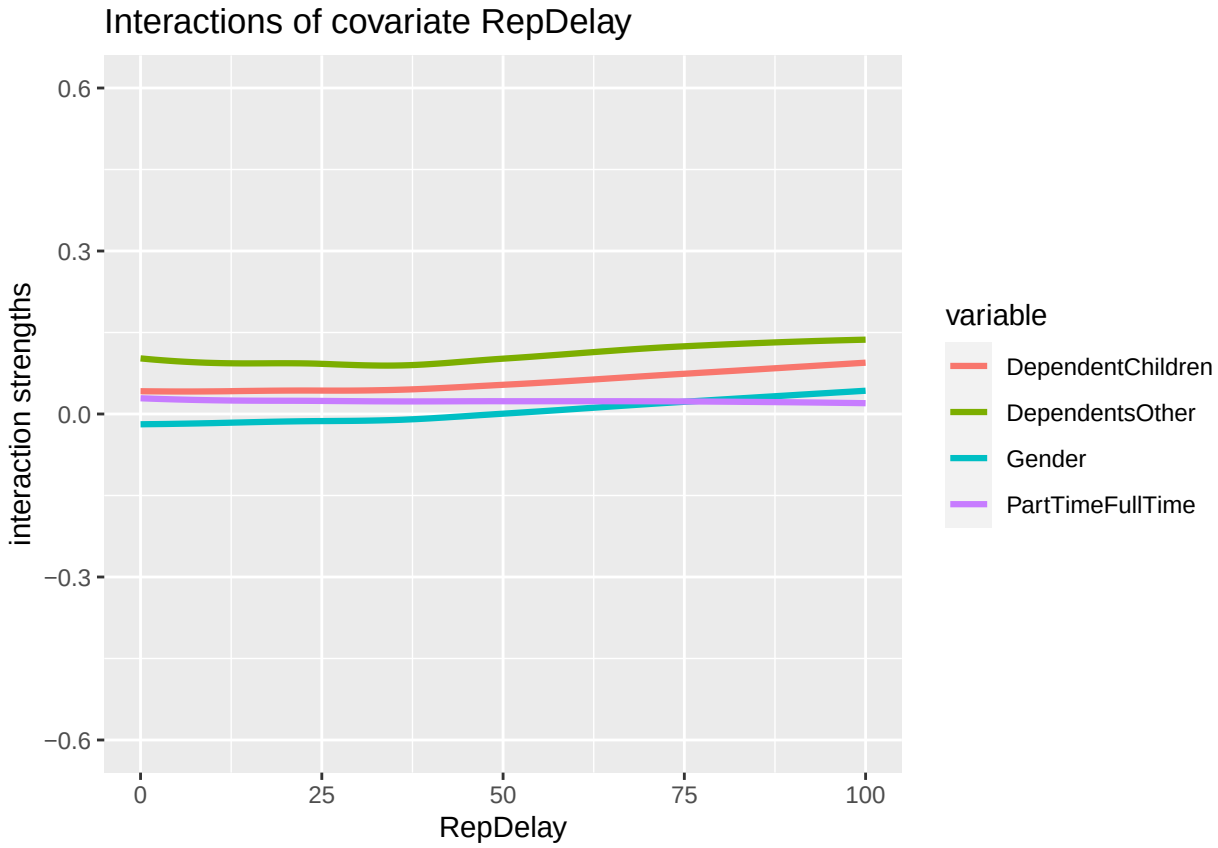



Interactions of covariate WeeklyPay
interaction strengths
0.6
0.3
0.0
−0.3
−0.6
0
250
500
750
1000
1250
WeeklyPay
variable
DependentChildren
DependentsOther
Gender
PartTimeFullTime


Interactions of covariate AccYear
interaction strengths
0.6
0.3
0.0
−0.3
−0.6
1990
1995
2000
2005
AccYear
variable
DependentChildren
DependentsOther
Gender
PartTimeFullTime


Interactions of covariate AccMonth
interaction strengths
0.6
0.3
0.0
−0.3
−0.6
2.5
5.0
7.5
10.0
12.5
AccMonth
variable
DependentChildren
DependentsOther
Gender
PartTimeFullTime


Interactions of covariate AccTime
interaction strengths
0.6
0.3
0.0
−0.3
−0.6
0
5
10
15
20
AccTime
variable
DependentChildren
DependentsOther
Gender
PartTimeFullTime



⇒ Also here we observe major interactions, e.g., between Age, PartTimeFullTime and Gender, or WeeklyPay and PartTimeFullTime, AccMonth and Gender or AccTime and DependentChildren.

Exercise: Based on the results above, how would the plots look like when there are no interactions? You can try to look at the randomly generated features and validate your guess

Conclusions

This tutorial presents the LocalGLMnet which gives us an interpretable network architecture. * We have seen that this architecture allows for variable selection. * It allows to quantify variable importance. * It allows for the study of interactions.

We see the major advantage of LocalGLMnet by allowing to quickly understand the data, in a very simple and straightforward way. LocalGLMnet brings faster a much better understanding than well-known techniques from explainable Machine Learning.

We have exemplified this on a synthetic accident insurance data set, and we have identified variables that can be dropped from the model.

As a side product we have seen that working with claim sizes can be challenging because the commonly used distributional models for regression modeling are less heavy-tailed than typical insurance claim size data. If this is the case, the regression models often slightly over-fit to the largest claims. This can be partially mitigated by a more robust estimation approach.

Exercises

Exercise: Use the function `square_loss` provided and fit models using the square loss function and compare the results. What do you conclude?

Exercise: Change the value of p and compare the results.

Exercise: Change the `validation_split` argument for training the models and compare the results. Does this has a relevant impact on the results?

Exercise: Change the `batch_size` argument for training the models and compare the results. Does this has a relevant impact on the results?

Exercise: Change the number of neurons, compare the number of parameters and the fitted results.

Exercise: Read the documentation to the optimizers and run the subsequent analysis with different optimizers and compare the results.

Exercise: Change the random seeds (at the beginning of the tutorial) and compare the results.

Exercise: Run the same analysis and see if you can fully reproduce the results. What do you conclude from that?

Exercise: Change the activation function (only where it is appropriate) and compare the results.

Exercise: Change the validation_split and verbose argument and see the difference in the fitting of the model.

Exercise: The derivation of variable importance, the regression attentions and the interactions is not easy to follow. Try to understand the code in more details.

Exercise: Write a function which calculates the regression attentions and the covariance contributions.

Session Info

The html is generated with the follow packages (which might be slightly newer than the ones used in the published tutorial).

```
sessionInfo()
```

```
## R version 4.0.5 (2021-03-31)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 20.04.2 LTS
##
## Matrix products: default
## BLAS/LAPACK: /usr/lib/x86_64-linux-gnu/openblas-pthread/libopenblasp-r0.3.8.so
##
## Random number generation:
##  RNG:     Mersenne-Twister
##  Normal:  Inversion
##  Sample:  Rounding
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
##  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=C
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
```

```
##  [1] corrplot_0.90  tidyr_1.1.3    gridExtra_2.3  ggplot2_3.3.5  purrr_0.3.4
##  [6] tibble_3.1.4   dplyr_1.0.7    magrittr_2.0.1 locfit_1.5-9.4 keras_2.6.0
##
## loaded via a namespace (and not attached):
##  [1] reticulate_1.14   tidyselect_1.1.1  xfun_0.23         splines_4.0.5
##  [5] lattice_0.20-41   colorspace_2.0-1  vctrs_0.3.8       generics_0.1.0
##  [9] htmltools_0.5.1.1 yaml_2.2.1        mgcv_1.8-34       base64enc_0.1-3
## [13] utf8_1.2.1        rlang_0.4.11      pillar_1.6.2      glue_1.4.2
## [17] withr_2.4.2       DBI_1.1.1         rappdirs_0.3.3    lifecycle_1.0.0
## [21] tensorflow_2.6.0  stringr_1.4.0     munsell_0.5.0     gtable_0.3.0
## [25] evaluate_0.14     labeling_0.4.2    knitr_1.34        tfruns_1.5.0
## [29] fansi_0.4.2       highr_0.9         Rcpp_1.0.7        scales_1.1.1
## [33] jsonlite_1.7.2    farver_2.1.0      digest_0.6.27     stringi_1.6.1
## [37] grid_4.0.5        cli_2.5.0         tools_4.0.5       crayon_1.4.1
## [41] whisker_0.4       pkgconfig_2.0.3   zeallot_0.1.0     Matrix_1.3-2
## [45] ellipsis_0.3.2    assertthat_0.2.1  rmarkdown_2.11    rstudioapi_0.13
## [49] R6_2.5.0          nlme_3.1-152      compiler_4.0.5
```

```
reticulate::py_config()
```

```
## python:         /home/rstudio/.virtualenvs/r-reticulate/bin/python
## libpython:      /opt/conda/lib/libpython3.9.so
## pythonhome:     /opt/conda:/opt/conda
## version:        3.9.5 | packaged by conda-forge | (default, Jun 19 2021, 00:32:32)  [GCC 9.3.0]
## numpy:          /home/rstudio/.virtualenvs/r-reticulate/lib/python3.9/site-packages/numpy
## numpy_version:  1.19.5
## tensorflow:     /home/rstudio/.virtualenvs/r-reticulate/lib/python3.9/site-packages/tensorflow
##
## python versions found:
##  /home/rstudio/.virtualenvs/r-reticulate/bin/python
##  /opt/conda/bin/python3
##  /usr/bin/python3
```

```
tensorflow::tf_version()
```

```
## [1] '2.6'
```

References

- https://tensorflow.rstudio.com/guide/
- https://github.com/rstudio/cheatsheets/raw/master/keras.pdf
- https://cran.r-project.org/web/packages/keras/vignettes/guide_keras.html
- https://keras.rstudio.com/articles/about_keras_models.html
- https://keras.rstudio.com/articles/functional_api.html
- https://cran.rstudio.com/web/packages/keras/vignettes/sequential_model.html
- https://www.rdocumentation.org/packages/keras/versions/2.3.0.0/topics/layer_dense
- https://www.rdocumentation.org/packages/keras/versions/2.1.6/topics/compile