Assignment 3

Group 2

2023-10-23

Task 1

In this exercise we want to quantify the expected in-sample error, the expected training error and the difference between the two. Firstly, as we derived from **exercise 3**, we have that

$$E[Err_{in} - \overline{err}] = E[op] = \frac{2}{N}\sum_{i=1}^{N} Cov(\hat{y}_i, y_i)$$

In relation to $E[op]$, in **exercise 4** we have a very similar point to make. For a linear model with the assumptions given in the exercise, we have that the sum over the diagonal elements is the trace of the covariance matrix. Recalling $\hat{y}_i = X(X^TX)^{-1}X^Ty$:

$$\begin{aligned}
&E[op] = \frac{2}{N}\sum_{i=1}^{N} Cov(\hat{y}_i, y_i) = \\
&\frac{2}{N}\, trace(X(X^TX)^{-1}X^T)\, \sigma_\epsilon^2 = \\
&\frac{2}{N}\, trace((X^TX)(X^TX)^{-1})\, \sigma_\epsilon^2 = \\
&\frac{2}{N}\, trace(I_d)\, \sigma_\epsilon^2 = \frac{2d\sigma_\epsilon^2}{N}
\end{aligned}$$

So, this is the actual difference between the Expected in-sample error and the Expected training error. We now turn to deriving the Expected in-sample error. For simplicity in notation we define $H = X(X^TX)^{-1}X^T$ as the hat matrix.

Now we turn to deriving to the task of deriving the expected in-sample error. Note y_0 denote new observations which are independent from the training dataset observations. Then $\hat{y}_0$ is just the estimated value conditional to the training dataset that we have.

$$\begin{aligned}
&E[Err_{in}] = \frac{1}{N}\,(y_0 - \hat{y_0})^T\,(y_0 - \hat{y_0}) = \\
&\frac{1}{N}(\epsilon^T(H-I)^T(H-I)\epsilon)
\end{aligned}$$

Now use that $(H - I)$ is idempotent, since these are the matrices that give the orthogonal projections. Furthermore, since H is symmetric $I - H$ is also symmetric.

$$\begin{aligned}
&E[Err_{in}] = \frac{1}{N}(\epsilon^T(H-I)^T(H-I)\epsilon) = \\
&\frac{1}{N}(\epsilon^T(I-H)\epsilon) = \frac{1}{N}\epsilon^T\epsilon - \frac{1}{N}\epsilon^T H\epsilon
\end{aligned}$$

This is the in-sample error given a training set we have. Now we take the average over all training sets:

$$E_\tau[E[Err_{in}]] = \frac{1}{N}\left(E_\tau[\epsilon^T\epsilon] - E_\tau[\epsilon^T H\epsilon]\right) = \sigma_\epsilon^2 - \frac{\sigma_\epsilon^2\,(d+1))}{N}$$

This is because:

$$E_\tau[\epsilon^T\epsilon] = E_\tau[e_1^2 + e_2^2 + \cdots + e_N^2] = N\sigma_\epsilon^2 \quad (1)$$

$$E_\tau[\epsilon^T H\epsilon] = E_\tau\left[\sum_{i=1}^{N} H_{ii}e_i^2 + \sum_{i\neq j}^{N} H_{ij}\epsilon_i\epsilon_j\right] = trace(H)\;\sigma^2 + 0 \quad (2)$$

For (2) we can say the errors are assumed to be independent. We now derived the expected in-sample error to be:

$$E_\tau[E[Err_{in}]] = \sigma_\epsilon^2\left(1 - \frac{(d+1)}{N}\right)$$

The Expected training error is:

$$E[\overline{err}] = E_\tau[Err_{in}] - E[op] = \sigma_\epsilon^2\left(1 - \frac{(d+1)}{N}\right) - \frac{2d\sigma_\epsilon^2}{N} = \sigma_\epsilon^2\left(1 - \frac{(1-d)}{N}\right)$$

Task 2

We first set up the training data and fix $\hat{f}$.

```
# training data
n = 40
x <- rnorm(n, mean = 0, sd = 1)
x2 <- x^2
noise <- rnorm(n, mean = 0, sd = 1)
y <- x + x2 + noise
df <- data.frame(y = y, x = x, x2 = x2)

# fix f hat
f_hat <- lm(data = df, formula = y ~ .)
```

The test error is given in general as

$$\mathrm{Err}_{\mathcal{T}} = \mathrm{E}[L(Y, \hat{f}(X)) \mid \mathcal{T}]$$

where $mathcalT$ is a particular test data set with data unobserved by the model but drawn from the joint distribution of Y and X and L is a loss function. The squared error loss is given by $(Y - \hat{f}(X))^2$ s.t. we are searching for

$$\mathrm{Err}_{\mathcal{T}} = \mathrm{E}[(Y, \hat{f}(X))^2 \mid \mathcal{T}]$$

To find this quantity we have to integrate over the joint distribution $f(x, y)$ which is given by

$$f(x, y) = f(y \mid x)f(x)$$

which are two quantities we know because $x \sim N(0, \sigma_x^2) \Leftrightarrow f(x) = \frac{1}{\sigma_x\sqrt{2\pi}} e^{-\frac{1}{2\sigma_x^2}x^2}$ and $y = x + x^2 + \epsilon \Rightarrow$ E[y | x] $= x + x^2$, $Var[y|x] = \sigma_\epsilon^2$ as x is predetermined and thus constant. As $y|x$ is further just a function of predetermind values and a normal RV it must be normal too, i.e. we have $y|x \sim N(x + x^2, \sigma_\epsilon^2) \Leftrightarrow f(y|x) = \frac{1}{\sigma_\epsilon\sqrt{2\pi}} e^{-\frac{1}{2\sigma_\epsilon^2}(y-x-x^2)^2}$. Thus, we have

$$f(x,y) = f(y|x)f(x) = \frac{1}{\sigma_\epsilon\sqrt{2\pi}} e^{-\frac{1}{2\sigma_\epsilon^2}(y-x-x^2)^2} \cdot \frac{1}{\sigma_x\sqrt{2\pi}} e^{-\frac{1}{2\sigma_x^2}x^2} = \frac{1}{\sigma_\epsilon\sigma_x 2\pi} e^{-\frac{1}{2\sigma_\epsilon^2}\left(y-x-x^2\right)^2 - \frac{1}{2\sigma_x^2}x^2}$$

Further, the squared loss function for this particular problem is given by

$$L(Y, \hat{f}(X)) = (y - \hat{\beta}_0 - \hat{\beta}_1 x - \hat{\beta}_2 x^2)^2$$

such that we finally arrive at

$$\text{Err}_\mathcal{T} = \text{E}[(Y, \hat{f}(X))^2 \mid \mathcal{T}] = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \frac{1}{\sigma_\epsilon\sigma_x 2\pi} e^{-\frac{1}{2\sigma_\epsilon^2}\left(y-x-x^2\right)^2 - \frac{1}{2\sigma_x^2}x^2} (y - \hat{\beta}_0 - \hat{\beta}_1 x - \hat{\beta}_2 x^2)^2 dxdy$$

Solving this integral by hand does not appear to be very easy so we opt for numerical integration.

```
# library cubature allows multivariate numerical integration
if(!require(cubature)) install.packages("cubature")
```

```
## Lade nötiges Paket: cubature
```

```
## Warning: Paket 'cubature' wurde unter R Version 4.2.3 erstellt
```

```
library(cubature)

sigma_x <- 1
sigma_e <- 1

# setup the integral as a function of two variables
 int <- function(z) {
  # setup the
  x <- z[1]
  y <- z[2]
  # setup the joint joint density
  jdens <- {
    (1 / (2 * pi * sigma_x * sigma_e)) * exp(-x^2 / (2 * sigma_x^2) -
                                               (y - x - x^2)^2 / (2 * sigma_e^2))}

  # calculate swuared error with coefficients from f hat
  sq_loss <- (y - coef(f_hat)[1] - coef(f_hat)[2] * x - coef(f_hat)[3] * x^2)^2
  return(jdens * sq_loss)
 }

# solve the integral
solved <- adaptIntegrate(f = int, lowerLimit = c(-Inf, -Inf), upperLimit = c(Inf, Inf))
```

```
# Simulation
# make test data set of equal size
x_test <- rnorm(40, mean = 0, sd = 1)
```

```r
x2_test <- x^2
noise_test <- rnorm(40, mean = 0, sd = 1)
y_test <- x + x2 + noise
df_test <- data.frame(y = y_test, x = x_test, x2 = x2_test)

# make 10000 test data sets of n = 10
test_data <- lapply(1:10000, function(i) {
  n = 10
  x <- rnorm(n, mean = 0, sd = 1)
  x2 <- x^2
  noise <- rnorm(n, mean = 0, sd = 1)
  y <- x + x2 + noise
  data.frame(y = y, x = x, x2 = x2)
})

# using f hat predict y for the test data and
# calculate the squared error loss
mean_squared_loss <- lapply(test_data, function(tt) {
  pred <- predict(f_hat, tt)
  test_error <- mean((pred - tt$y)^2)
})

# test error of the simulation is the mean over
# the individual test errors
mean(unlist(mean_squared_loss))
```

```
## [1] 1.060133
```

```r
# compare to integral
solved$integral
```

```
## [1] 1.048856
```

We see that the two values are very similar meaning that we can approximate the true test error with simulation methoid in a pretty accurate way.

Task 3

Let $\widehat{f}(x_i) = \widehat{y}_i$. We aim to derive an expression for w:

$$
\begin{aligned}
w &= E_y[op] \\
&= E_y[Err_{in} - \overline{err}] \\
&= E_y[Err_{in}] - E_y[\overline{err}] \\
&= E_y\left[\frac{1}{N}\sum_{i=1}^{N} E_{Y^0}[L(Y_i^0, \widehat{f}(x_i))]\right] - E_y\left[\frac{1}{N}\sum_{i=1}^{N} L(y_i, \widehat{f}(x_i))\right] \\
&= \frac{1}{N}\sum_{i=1}^{N} E_y E_{Y^0}\left[(Y_i^0 - \widehat{y}_i)^2\right] - E_y\left[(y_i - \widehat{y}_i)^2\right].
\end{aligned}
$$

Continuing to simplify:

$$
\begin{aligned}
w &= \frac{1}{N}\sum_{i=1}^{N} E_y\left[{Y_i^0}^2\right] + E_y E_{Y^0}\left[\widehat{y}_i^2\right] - 2E_y E_{Y^0}\left[Y_i^0 \widehat{y}_i\right] - E_y\left[y_i^2\right] - E_y\left[\widehat{y}_i^2\right] + 2E_y\left[y_i \widehat{y}_i\right] \\
&= \frac{1}{N}\sum_{i=1}^{N} E_y\left[y_i^2\right] + E_y\left[\widehat{y}_i^2\right] - 2E_y\left[y_i\right] E_y\left[\widehat{y}_i\right] - E_y\left[y_i^2\right] - E_y\left[\widehat{y}_i^2\right] + 2E_y\left[y_i \widehat{y}_i\right].
\end{aligned}
$$

Further simplifying:

$$
\begin{aligned}
w &= \frac{2}{N}\sum_{i=1}^{N} E_y\left[y_i \widehat{y}_i\right] - E_y\left[y_i\right] E_y\left[\widehat{y}_i\right] \\
&= \frac{2}{N}\sum_{i=1}^{N} E_y\left[y_i \widehat{y}_i - y_i E_y\left[\widehat{y}_i\right] - E_y\left[y_i\right] \widehat{y}_i + E_y\left[y_i\right] E_y\left[\widehat{y}_i\right]\right] \\
&= \frac{2}{N}\sum_{i=1}^{N} E_y\left[(\widehat{y}_i - E_y\left[\widehat{y}_i\right])(y_i - E_y\left[y_i\right])\right].
\end{aligned}
$$

This expression simplifies to:

$$
w = \frac{2}{N}\sum_{i=1}^{N} \mathrm{Cov}(\widehat{y}_i, y_i).
$$

Therefore, we have established that w is equal to the sum of the covariances between the predicted values $\widehat{y}_i$ and the actual values y_i for the given dataset.

Task 4

If $\mathbf{y}$ arises from an additive-error model $Y = f(X) + \epsilon$ with $Var(\epsilon) = \sigma_\epsilon^2$ and where $\hat{y} = Sy$, then one can show that:

$$
\sum_{i=1}^{N} cov(\hat{y}_i, y_i) = trace(S)\sigma_\epsilon^2
$$

Solution:

The term $\sum_{i=1}^{N} cov(\hat{y}_i, y_i)$ represents the sum of the diagonal values of the covariance matrix, which is therefore the trace of the covariance matrix. Furthermore, let's consider $cov(\hat{y}, y)$ for vectors y and $\hat{y}$. We have that:

$$
cov(\hat{y}, y) = cov(Sy, y) = S\ cov(y, y) = S\ \sigma_\epsilon^2
$$

Adding the two observations together we have that:

$$
\sum_{i=1}^{N} cov(\hat{y}_i, y_i) = trace(S)\sigma_\epsilon^2
$$

Task 5

Assume for N observations $y = (y_1, ..., y_N)$ the following model: They are drawn i.i.d. from a Poisson distribution with parameter λ. Further assume that the prior distribution for λ is an improper prior which is proportional to a constant on the positive reals.

a

First we need to determine an approximation of the marginal likelihood based on the Laplace approximation given by

$$p(y|\mathcal{M}) \approx \exp(\ell(y|\hat{\lambda}))\sqrt{\frac{2\pi}{\mathcal{J}(\hat{\lambda})}},$$

where $\ell(y|\lambda)$ is the log-likelihood of the data assuming that the observations are i.i.d. data from a Poisson distribution with parameter λ, $\hat{\lambda}$ is the maximum likelihood estimate and $\mathcal{J}(\lambda)$ is the observed information matrix, i.e., the second derivative of the log-likelihood function evaluated at λ.

The likelihoodfunction of the data is:

$$L(\lambda; y) = \prod_{i=1}^{n} \frac{e^{-\lambda}\lambda^{y_i}}{y_i!}$$

Therefore the log-likelihood function is:

$$\ell(\lambda; y) = \sum_{i=1}^{n} \left(-\lambda + y_i \log(\lambda) - \log(y_i!)\right)$$

To get the MLE of λ we derive the log-likelihood function wrt. λ, then set it to zero and solve for λ:

$$\frac{d\ell}{d\lambda} = \sum_{i=1}^{n} \left(-1 + \frac{y_i}{\lambda}\right) = 0$$

$$\implies \hat{\lambda}_{\text{MLE}} = \frac{1}{n}\sum_{i=1}^{n} y_i$$

For the observed information matrix we derive it again and take the negative of that but since we only have one parameter we have a matrix which exists of one value:

$$-\frac{d^2\ell}{d\lambda^2} = -\left(-\sum_{i=1}^{n} \frac{y_i}{\lambda^2}\right) = \sum_{i=1}^{n} \frac{y_i}{\lambda^2}$$

Now lets plug everything into the formula for the Laplace approximation:

$$p(y|\mathcal{M}) \approx \exp\left(\sum_{i=1}^{n} \left(-\hat{\lambda} + y_i \log(\hat{\lambda}) - \log(y_i!)\right)\right)\sqrt{\frac{2\pi}{\sum_{i=1}^{n} \frac{y_i}{\hat{\lambda}^2}}}$$

$$= \exp\left(\sum_{i=1}^{n}\left(-\frac{1}{n}\sum_{k=1}^{n} y_k + y_i \log(\frac{1}{n}\sum_{k=1}^{n} y_k) - \log(y_i!)\right)\right)\sqrt{\frac{2\pi}{\frac{1}{\hat{\lambda}^2}\sum_{i=1}^{n} y_i}}$$

$$= \exp\left(\sum_{i=1}^{n}\left(-\frac{1}{n}\sum_{k=1}^{n} y_k + y_i \log(\frac{1}{n}\sum_{k=1}^{n} y_k) - \log(y_i!)\right)\right)\sqrt{\frac{2\pi}{\frac{1}{(\frac{1}{n}\sum_{i=1}^{n} y_i)^2}\sum_{i=1}^{n} y_i}}$$

$$= \exp\left(\sum_{i=1}^{n}\left(-\frac{1}{n}\sum_{k=1}^{n} y_k + y_i \log(\frac{1}{n}\sum_{k=1}^{n} y_k) - \log(y_i!)\right)\right)\sqrt{\frac{2\pi}{\frac{1}{\frac{1}{n^2}(\sum_{i=1}^{n} y_i)^2}\sum_{i=1}^{n} y_i}}$$

$$= \exp\left(\sum_{i=1}^{n}\left(-\frac{1}{n}\sum_{k=1}^{n} y_k + y_i \log(\frac{1}{n}\sum_{k=1}^{n} y_k) - \log(y_i!)\right)\right)\sqrt{\frac{\frac{2\pi}{1}}{\frac{n^2}{\sum_{i=1}^{n} y_i}}}$$

$$= \exp\left(\sum_{i=1}^{n}\left(-\frac{1}{n}\sum_{k=1}^{n} y_k + y_i \log(\frac{1}{n}\sum_{k=1}^{n} y_k) - \log(y_i!)\right)\right)\sqrt{\frac{2\pi\sum_{i=1}^{n} y_i}{n^2}}$$

b

Now we determine -2 times the logarithm of the approximation and compare the result to the Bayesian information criterion for this model:

$$-2 * \log\left(\exp\left(\sum_{i=1}^{n}\left(-\frac{1}{n}\sum_{k=1}^{n} y_k + y_i \log(\frac{1}{n}\sum_{k=1}^{n} y_k) - \log(y_i!)\right)\right)\sqrt{\frac{2\pi\sum_{i=1}^{n} y_i}{n^2}}\right)$$

$$= -2 * \left(\sum_{i=1}^{n}\left(-\frac{1}{n}\sum_{k=1}^{n} y_k + y_i \log(\frac{1}{n}\sum_{k=1}^{n} y_k) - \log(y_i!)\right) + \log\left(\sqrt{\frac{2\pi\sum_{i=1}^{n} y_i}{n^2}}\right)\right)$$

The bayesian information criterion has the form:

$$BIC = -2 * loglik + \log(n) * d$$

(where d is the number of parameters, so in our case 1)

$$= -2 * \sum_{i=1}^{n}\left(-\hat{\lambda} + y_i \log(\hat{\lambda}) - \log(y_i!)\right) + \log(n) * 1$$

$$= -2 * \sum_{i=1}^{n}\left(-\frac{1}{n}\sum_{k=1}^{n} y_k + y_i \log(\frac{1}{n}\sum_{k=1}^{n} y_k) - \log(y_i!)\right) + \log(n)$$

Therefore the left terms are the same for both and we only need to compare

$$-2 * \log\left(\sqrt{\frac{2\pi\sum_{i=1}^{n} y_i}{n^2}}\right) \quad \text{with} \quad \log(n)$$

The left term can be rewritten as:

$$-2 * \log\left(\sqrt{\frac{2\pi\sum_{i=1}^{n} y_i}{n^2}}\right) = -2 * \log\left(\sqrt{\frac{2\pi\sum_{i=1}^{n} y_i}{n}} * \frac{1}{\sqrt{n}}\right)$$

$$= -2 * \log\left(\sqrt{\frac{2\pi\sum_{i=1}^{n} y_i}{n}}\right) + 2 * \log\left(\sqrt{n}\right) = -2 * \log\left(\sqrt{\frac{2\pi\sum_{i=1}^{n} y_i}{n}}\right) + \log\left(n\right)$$

So for the terms to be the same we need to have:

$$-2 * \log\left(\sqrt{\frac{2\pi\sum_{i=1}^{n} y_i}{n}}\right) + \log\left(n\right) = \log(n)$$

$$\Leftrightarrow -2 * \log\left(\sqrt{\frac{2\pi\sum_{i=1}^{n} y_i}{n}}\right) = 0 \quad \Leftrightarrow \log\left(\sqrt{\frac{2\pi\sum_{i=1}^{n} y_i}{n}}\right) = 0$$

$$\Leftrightarrow \sqrt{\frac{2\pi\sum_{i=1}^{n} y_i}{n}} = 1 \quad \Leftrightarrow \frac{2\pi\sum_{i=1}^{n} y_i}{n} = 1$$

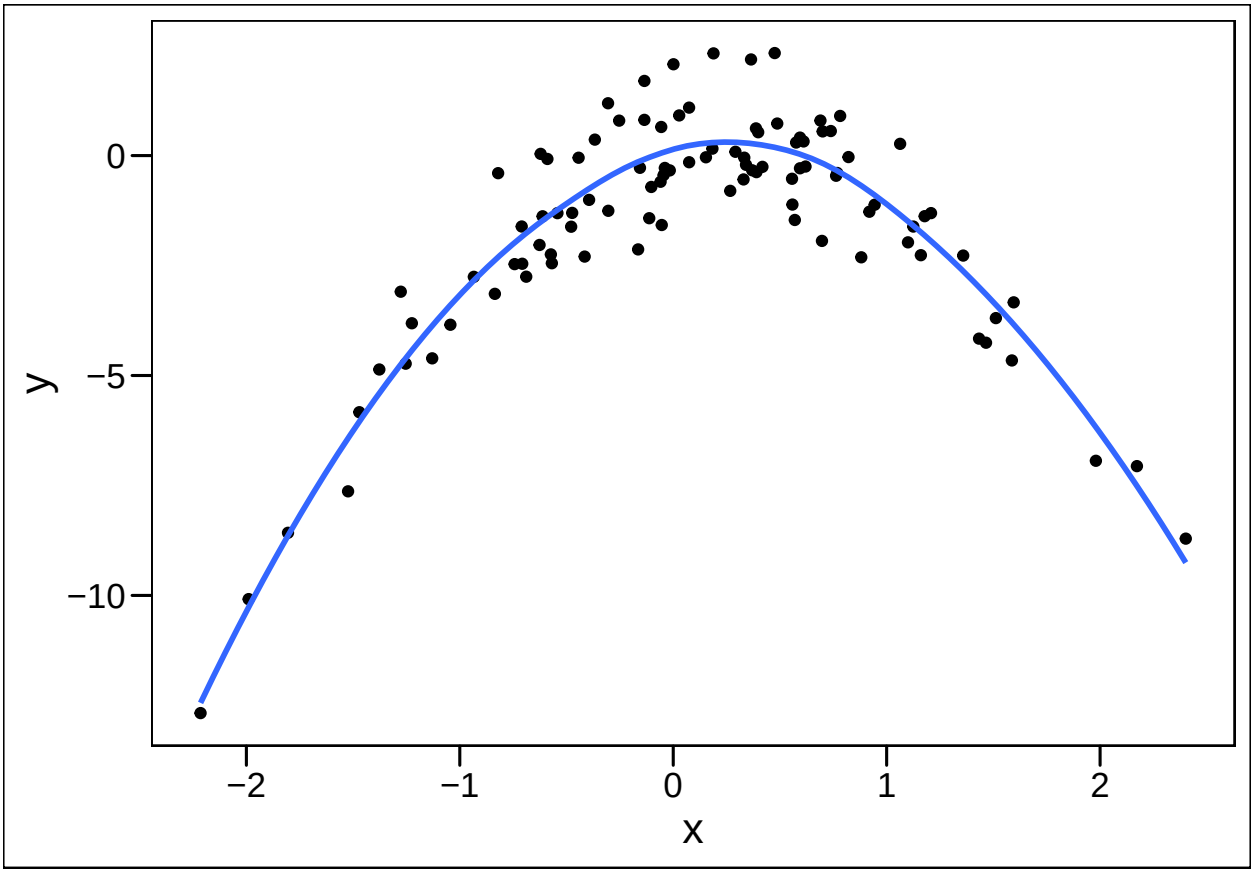
$$\Leftrightarrow 2\pi\sum_{i=1}^{n} y_i = n$$

Task 6

```
# a)
# data
set.seed(1)
x <- rnorm(100)
y <- x - 2*x^2 + rnorm(100)
```

```
# assuming you dont mean orthogonal polynomials we set up a
#data.frame with all the polynomials inside
dat <- data.frame(y = y, x = x, x2 = x^2, x3 = x^3, x4 = x^4)

# b)
# we then create a scatterplot and we dare to use
# ggplots in built loess implementation to draw a curve
# that fits the data as good as possible

dat %>%
  ggplot(aes(x = x, y = y)) +
  geom_point() +
  geom_smooth(method = "loess", se = F) +
  theme_base()
```

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



Looking at the scatterplot we observe that the data follows an inverted U-shaped pattern that could be

expected if you choose $f(x) = x - 2x^2$. Another notable fact is that most of the data points are located around the turning point of the inverted U. However, this behavior of y is also expectable as $x, \epsilon \sim N(0, 1)$ both have most of their mass around 0 and hence the most common value should be around 0 looking on the data generating process.

```
# fit the models
fits <- lapply(2:5, function(i) {
  lm(data = dat[, 1:i], y ~ .)
})

# calculate AIC stepwise
# A get number of coefficients + 1 (we estimate the variance in addition)
K <- unlist(lapply(fits, function(f) length(f$coef))) + 1

# get the LogLikelihood of the models
LL <- unlist(lapply(fits, logLik))

# AIC
AIC <- 2*K - 2*LL
AIC
```

```
## [1] 478.8804 280.1670 282.0886 282.2963
```

```
# print model summary with for lowest AIC
summary(fits[[which.min(AIC)]])
```

```
##
## Call:
## lm(formula = y ~ ., data = dat[, 1:i])
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.9650 -0.6254 -0.1288  0.5803  2.2700
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.05672    0.11766   0.482    0.631
## x            1.01716    0.10798   9.420  2.4e-15 ***
## x2          -2.11892    0.08477 -24.997  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.958 on 97 degrees of freedom
## Multiple R-squared:  0.873,  Adjusted R-squared:  0.8704
## F-statistic: 333.3 on 2 and 97 DF,  p-value: < 2.2e-16
```

```
# d)

# get fitted values from models in c
pred_c <- lapply(fits, function(f) f$fitted)

# get residual standard deviations
sigma_c <- lapply(fits, function(f) summary(f)$sigma)
```

```r
# create 100 test data sets
new_y <- lapply(1:100, function(i) {
  y <- x - 2*x^2 + rnorm(100)
  y
})

# now the assumption is that errors in linear regression are gaussian hence
# the likelihood must be that of a gaussian distribution

# for each of the model specifications calculate the in sample error as the
# -2 log(LL) where log(LL) for each test data set is determined by inserting
# the new y values into the normal density with a mean equal to the fitted value
# of the particular observation and standard deviation equal to the residual
# standard deviation of the training fits. This gives us then the individual
# log likelihood for every newly generated y to come from the assumed data
# generating process. Summing over these then gives the log-likelihood of the
# whole test data set. Then we average over the log likelihoods of the 100 test
# data sets and multiply with -2.

Err <- rep(NA, 4)
for(i in 1:length(pred_c)) {
LL <- lapply(new_y, function(ny) {
  sum(log(dnorm(x = ny, mean = pred_c[[i]], sd = sigma_c[[i]])))
})
Err[i] <- -2 * mean(unlist(LL))
}

# print results
Err
```

```
## [1] 465.4097 288.7366 288.7320 290.5469
```

Looking at the in sample errors for the 4 models we calculated using this particular loss function we see that they are quite similar to the AIC values from task b). This is not very surprising as also the AIC builds on the $-2 * LL$ if you remember the formula to calculate AIC.

Task 7

a)

```
## Warning: Paket 'caret' wurde unter R Version 4.2.3 erstellt
```

```
## Warning: Paket 'knitr' wurde unter R Version 4.2.3 erstellt
```

```r
# Generate a simulated data set
set.seed(1)
x <- rnorm(100)
y <- x - 2*x^2 + rnorm(100)
```

b)

```
# Data prep
set.seed(100)
df_1 = data.frame(y=y,x=x)
df_2 = data.frame(y=y,x=x,x2=x^2)
df_3 = data.frame(y=y,x=x,x2=x^2,x3=x^3)
df_4 = data.frame(y=y,x=x,x2=x^2,x3=x^3,x4=x^4)
model=list("i", "ii", "iii", "iv")
LOOCV_MSE=list()
kCV_MSE=list()
mse = function(sm)
    mean(sm$results$RMSE^2)

# LOOCV cross-validation method
ctrl_loocv <- trainControl(method = "LOOCV")
ctrl_kcv <- trainControl(method = "cv", number = 10)

# First model

## LOOCV
model_LOOCV_1 <- train(y ~ ., data=df_1, method="lm", trControl=ctrl_loocv)
print(model_LOOCV_1)
```

```
## Linear Regression
##
## 100 samples
##   1 predictor
##
## No pre-processing
## Resampling: Leave-One-Out Cross-Validation
## Summary of sample sizes: 99, 99, 99, 99, 99, 99, ...
## Resampling results:
##
##   RMSE     Rsquared    MAE
##   2.69966  0.00130345  1.921879
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

```
## 10 fold CV
model_kCV_1 <- train(y ~ ., data=df_1, method="lm", trControl=ctrl_kcv)
print(model_kCV_1)
```

```
## Linear Regression
##
## 100 samples
##   1 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 91, 89, 91, 88, 89, 92, ...
## Resampling results:
```

```
##
##   RMSE      Rsquared  MAE
##   2.638627  0.40973   1.925501
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

```
## Save the results
LOOCV_MSE = append(LOOCV_MSE, mse(model_LOOCV_1))
kCV_MSE = append(kCV_MSE, mse(model_kCV_1))

# Second model

## LOOCV
model_LOOCV_2 <- train(y ~ ., data=df_2, method="lm", trControl=ctrl_loocv)
print(model_LOOCV_2)
```

```
## Linear Regression
##
## 100 samples
##   2 predictor
##
## No pre-processing
## Resampling: Leave-One-Out Cross-Validation
## Summary of sample sizes: 99, 99, 99, 99, 99, 99, ...
## Resampling results:
##
##   RMSE       Rsquared   MAE
##   0.9682064  0.8663108  0.7829438
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

```
## 10 fold CV
model_kCV_2 <- train(y ~ ., data=df_2, method="lm", trControl=ctrl_kcv)
print(model_kCV_2)
```

```
## Linear Regression
##
## 100 samples
##   2 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 90, 92, 91, 89, 91, 91, ...
## Resampling results:
##
##   RMSE       Rsquared   MAE
##   0.9459685  0.8724754  0.7751198
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

```
## Save the results
LOOCV_MSE = append(LOOCV_MSE, mse(model_LOOCV_2))
```

```
kCV_MSE = append(kCV_MSE, mse(model_kCV_2))


# Third model

## LOOCV
model_LOOCV_3 <- train(y ~ ., data=df_3, method="lm", trControl=ctrl_loocv)
print(model_LOOCV_3)
```

```
## Linear Regression
##
## 100 samples
##   3 predictor
##
## No pre-processing
## Resampling: Leave-One-Out Cross-Validation
## Summary of sample sizes: 99, 99, 99, 99, 99, 99, ...
## Resampling results:
##
##   RMSE       Rsquared   MAE
##   0.9780705  0.8637902  0.7924894
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

```
## 10 fold CV
model_kCV_3 <- train(y ~ ., data=df_3, method="lm", trControl=ctrl_kcv)
print(model_kCV_3)
```

```
## Linear Regression
##
## 100 samples
##   3 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 92, 91, 90, 89, 90, 89, ...
## Resampling results:
##
##   RMSE       Rsquared   MAE
##   0.9520476  0.8334319  0.7787047
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

```
## Save the results
LOOCV_MSE = append(LOOCV_MSE, mse(model_LOOCV_3))
kCV_MSE = append(kCV_MSE, mse(model_kCV_3))


# Fourth model

## LOOCV
model_LOOCV_4 <- train(y ~ ., data=df_4, method="lm", trControl=ctrl_loocv)
print(model_LOOCV_4)
```

```
## Linear Regression
##
## 100 samples
##   4 predictor
##
## No pre-processing
## Resampling: Leave-One-Out Cross-Validation
## Summary of sample sizes: 99, 99, 99, 99, 99, 99, ...
## Resampling results:
##
##   RMSE       Rsquared   MAE
##   0.9766805  0.8639008  0.8019279
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

```
## 10 fold CV
model_kCV_4 <- train(y ~ ., data=df_4, method="lm", trControl=ctrl_kcv)
print(model_kCV_4)
```

```
## Linear Regression
##
## 100 samples
##   4 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 89, 91, 91, 91, 90, 88, ...
## Resampling results:
##
##   RMSE       Rsquared   MAE
##   0.9638881  0.7964419  0.8081947
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

```
## Save the results
LOOCV_MSE = append(LOOCV_MSE, mse(model_LOOCV_4))
kCV_MSE = append(kCV_MSE, mse(model_kCV_4))

# Results
result = data.frame(Model = unlist(model), LOOCV_100 = unlist(LOOCV_MSE),
                    kCV_100 = unlist(kCV_MSE))
kable(result, caption = "Summary")
```

Table 1: Summary

Model	LOOCV_100	kCV_100
i	7.2881616	6.9623524
ii	0.9374236	0.8948565
iii	0.9566218	0.9063947
iv	0.9539049	0.9290803

c)

```
# Set seed
set.seed(99)
LOOCV_MSE_2=list()
kCV_MSE_2=list()
# First model

## LOOCV
model_LOOCV_1_2 <- train(y ~ ., data=df_1, method="lm", trControl=ctrl_loocv)
print(model_LOOCV_1_2)
```

```
## Linear Regression
##
## 100 samples
##   1 predictor
##
## No pre-processing
## Resampling: Leave-One-Out Cross-Validation
## Summary of sample sizes: 99, 99, 99, 99, 99, 99, ...
## Resampling results:
##
##   RMSE     Rsquared    MAE
##   2.69966  0.00130345  1.921879
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

```
## 10 fold CV
model_kCV_1_2 <- train(y ~ ., data=df_1, method="lm", trControl=ctrl_kcv)
print(model_kCV_1_2)
```

```
## Linear Regression
##
## 100 samples
##   1 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 91, 91, 89, 90, 89, 90, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   2.605796  0.2937923  1.912726
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

```
## Save the results
LOOCV_MSE_2 = append(LOOCV_MSE_2, mse(model_LOOCV_1_2))
kCV_MSE_2 = append(kCV_MSE_2, mse(model_kCV_1_2))

# Second model
```

```r
## LOOCV
model_LOOCV_2_2 <- train(y ~ ., data=df_2, method="lm", trControl=ctrl_loocv)
print(model_LOOCV_2_2)
```

```
## Linear Regression
##
## 100 samples
##   2 predictor
##
## No pre-processing
## Resampling: Leave-One-Out Cross-Validation
## Summary of sample sizes: 99, 99, 99, 99, 99, 99, ...
## Resampling results:
##
##   RMSE       Rsquared   MAE
##   0.9682064  0.8663108  0.7829438
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

```r
## 10 fold CV
model_kCV_2_2 <- train(y ~ ., data=df_2, method="lm", trControl=ctrl_kcv)
print(model_kCV_2_2)
```

```
## Linear Regression
##
## 100 samples
##   2 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 91, 90, 88, 91, 91, 91, ...
## Resampling results:
##
##   RMSE       Rsquared   MAE
##   0.9462944  0.8508985  0.7817394
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

```r
## Save the results
LOOCV_MSE_2 = append(LOOCV_MSE_2, mse(model_LOOCV_2_2))
kCV_MSE_2 = append(kCV_MSE_2, mse(model_kCV_2_2))


# Third model

## LOOCV
model_LOOCV_3_2 <- train(y ~ ., data=df_3, method="lm", trControl=ctrl_loocv)
print(model_LOOCV_3_2)
```

```
## Linear Regression
##
## 100 samples
```

```
##   3 predictor
##
## No pre-processing
## Resampling: Leave-One-Out Cross-Validation
## Summary of sample sizes: 99, 99, 99, 99, 99, 99, ...
## Resampling results:
##
##   RMSE       Rsquared   MAE
##   0.9780705  0.8637902  0.7924894
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

```
## 10 fold CV
model_kCV_3_2 <- train(y ~ ., data=df_3, method="lm", trControl=ctrl_kcv)
print(model_kCV_3_2)
```

```
## Linear Regression
##
## 100 samples
##   3 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 89, 90, 90, 91, 90, 88, ...
## Resampling results:
##
##   RMSE       Rsquared   MAE
##   0.9697049  0.8468917  0.7957661
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

```
## Save the results
LOOCV_MSE_2 = append(LOOCV_MSE_2, mse(model_LOOCV_3_2))
kCV_MSE_2 = append(kCV_MSE_2, mse(model_kCV_3_2))


# Fourth model

## LOOCV
model_LOOCV_4_2 <- train(y ~ ., data=df_4, method="lm", trControl=ctrl_loocv)
print(model_LOOCV_4_2)
```

```
## Linear Regression
##
## 100 samples
##   4 predictor
##
## No pre-processing
## Resampling: Leave-One-Out Cross-Validation
## Summary of sample sizes: 99, 99, 99, 99, 99, 99, ...
## Resampling results:
##
##   RMSE       Rsquared   MAE
```

```
##   0.9766805  0.8639008  0.8019279
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

```
## 10 fold CV
model_kCV_4_2 <- train(y ~ ., data=df_4, method="lm", trControl=ctrl_kcv)
print(model_kCV_4_2)
```

```
## Linear Regression
##
## 100 samples
##   4 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 90, 90, 90, 90, 89, 89, ...
## Resampling results:
##
##   RMSE       Rsquared   MAE
##   0.9581952  0.8539694  0.8136199
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

```
## Save the results
LOOCV_MSE_2 = append(LOOCV_MSE_2, mse(model_LOOCV_4_2))
kCV_MSE_2 = append(kCV_MSE_2, mse(model_kCV_4_2))

# Results
result_2 = data.frame(Model = unlist(model), LOOCV_100 = unlist(LOOCV_MSE),
                      kCV_100 = unlist(kCV_MSE),LOOCV_99=unlist(LOOCV_MSE_2),
                      kCV_99 = unlist(kCV_MSE_2))
kable(result_2, caption = "Comparison of different seeds")
```

Table 2: Comparison of different seeds

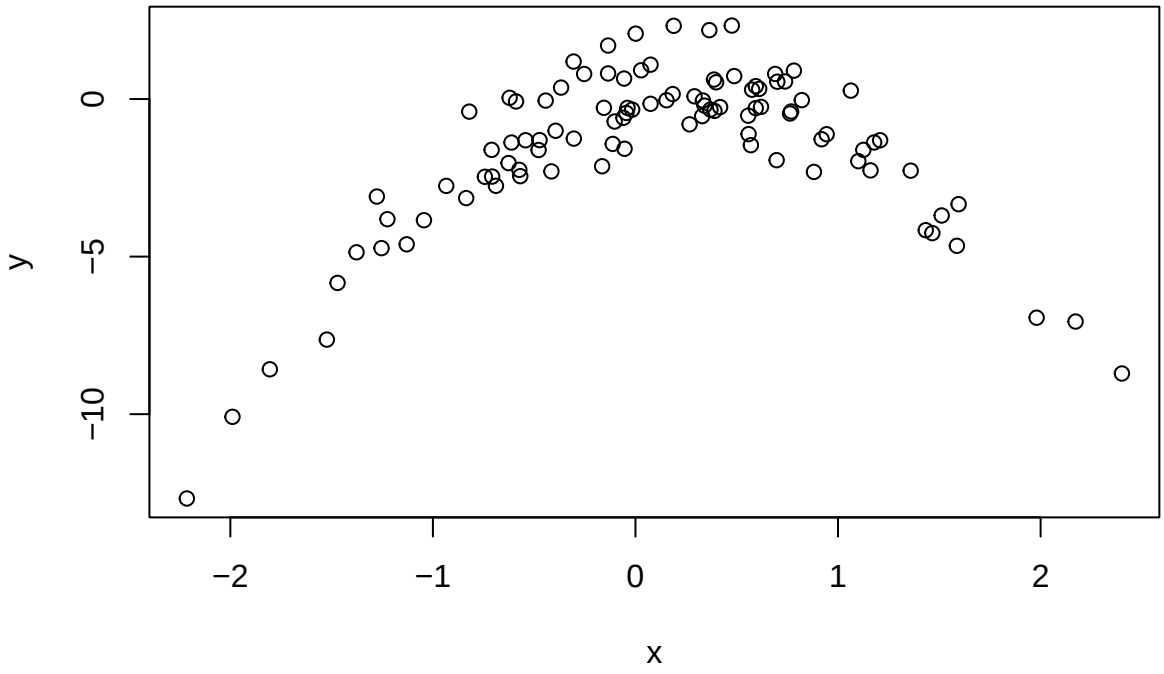
Model	LOOCV_100	kCV_100	LOOCV_99	kCV_99
i	7.2881616	6.9623524	7.2881616	6.7901735
ii	0.9374236	0.8948565	0.9374236	0.8954731
iii	0.9566218	0.9063947	0.9566218	0.9403276
iv	0.9539049	0.9290803	0.9539049	0.9181379

As can be seen from Table 2, the results for Leave-One-Out CV is similar for different seeds since in LOOCV we average the result of n models, which differ in only one observation, therefore we can say that there is more overlap. However, for 10-fold cross validation we observe an (insignificant) difference. In this case, we use 10 fitted models and average the results. Here the difference can be explained by the lower correlation between the training sets resulting from a smaller overlap.

d)

$Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \epsilon$ has the smallest LOOCV and 10-fold cross validation error in both seeds. This can be explained with the scatterplot below. One can see the curvature on the graph, which can be explained by a second degree polynomial, consistent with our results.

```
plot(x,y)
```



Task 8

We perform a simulation study to assess how good the performance of the LASSO is for variable selection:

The following data generating process is used:
- Draw a 100-dimensional vector from a standard multivariate normal distribution.
- Determine the dependent variable with $\epsilon \sim N(0, 0.1)$ by:

$$y = \sum_{i=1}^{10} x_i + \epsilon.$$

a

Draw 100 data sets of size 1000. Split each data set into a training data set containing the first 100 observations and a test data set containing the remaining 900 observations. For each of the 100 repetitions use glmnet from the glmnet package to fit the LASSO model for different values of λ to the training data set and select the λ value where predictive performance is best on the test data set.

b

Determine the proportion of correctly included coefficients from all relevant ones (true positive rate) and the proportion of wrongly included coefficients from all irrelevant ones (false positive rate) for each of the 100

data sets and visualize the distribution of the two rates.

```
library(glmnet)
```

```
## Warning: Paket 'glmnet' wurde unter R Version 4.2.3 erstellt
```

```
## Lade nötiges Paket: Matrix
```

```
## Warning: Paket 'Matrix' wurde unter R Version 4.2.3 erstellt
```

```
##
## Attache Paket: 'Matrix'
```

```
## Die folgenden Objekte sind maskiert von 'package:tidyr':
##
##     expand, pack, unpack
```

```
## Loaded glmnet 4.1-8
```

```
library(MASS)
```

```
##
## Attache Paket: 'MASS'
```

```
## Das folgende Objekt ist maskiert 'package:dplyr':
##
##     select
```

```
library(mvtnorm)
library(Metrics)
```

```
## Warning: Paket 'Metrics' wurde unter R Version 4.2.3 erstellt
```

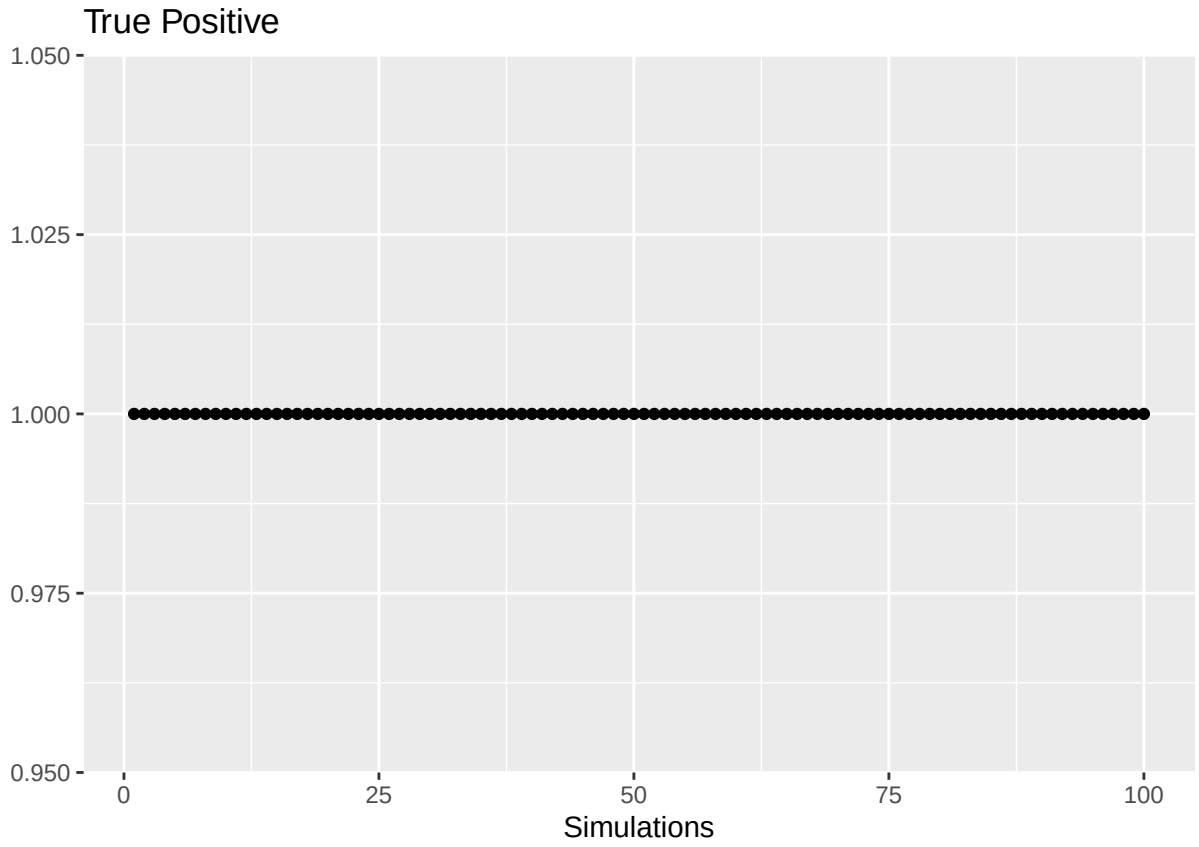
```
##
## Attache Paket: 'Metrics'
```

```
## Das folgende Objekt ist maskiert durch '.GlobalEnv':
##
##     mse
```

```
## Die folgenden Objekte sind maskiert von 'package:caret':
##
##     precision, recall
```

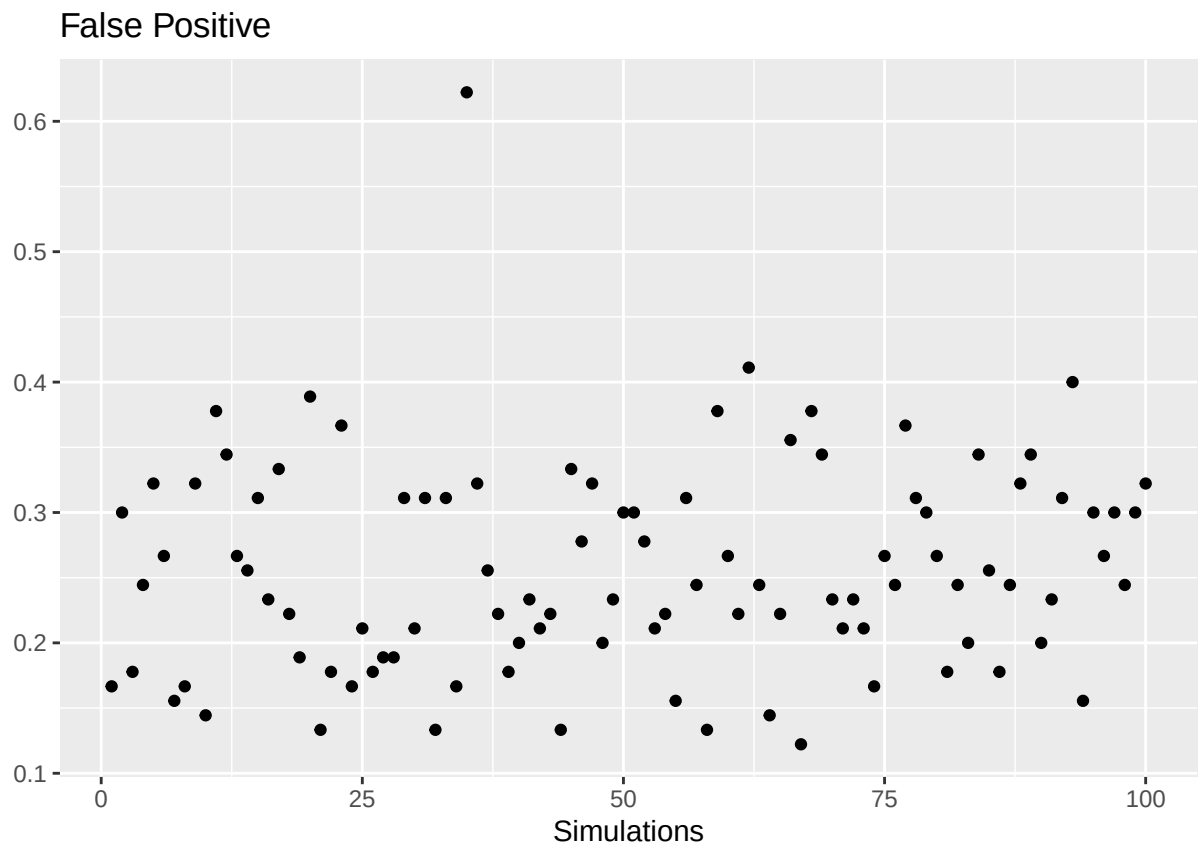
```
library(ggplot2)


ex8 <- function(x){
  # a)

  set.seed(x)
  # Matrix with 100 columns and 1000 rows
  X <- rmvnorm(n = 1000, mean = rep(0, 100), sigma = diag(100))
  # First 100 rows of each column
  train_X <- X[1:100,]
  # Other rows
  test_X <- X[101:1000,]
  # 1000 N(0,0.1) distributed values
  epsilon <- rnorm(n = 1000, mean = 0, sd = sqrt(0.1))
  # Calculate Y by formula
  Y <- rowSums(X[,1:10])+epsilon
  # First 100 elements of calculated Y
  train_Y <- Y[1:100]
  # Rest of elements
  test_Y <- Y[101:1000]
  # Fit lasso model on training dataset
  lasso <- glmnet(train_X, train_Y, alpha=1)
  # Get different lambdas
  lambda <- lasso$lambda
  # Create vector to store the predictions
  predictions <- c()
  # Loop through each lambda to predict on test data set
  # and calculate the root mean squared error
  for (i in 1:length(lambda)) {
    prediction <- predict(lasso, newx=test_X, s=lambda[i])
    predictions[i] <- rmse(test_Y, prediction)
  }
  # Find the best predictive lambda (where the RMSE is the smallest)
  lambda_best <- lambda[which(predictions==min(predictions))]

  #b)

  # Get coefficients for best lambda and create a matrix without the intercept
  coefs <- as.matrix(coef(lasso, s=lambda_best)[-1,])
  # Converting them to true or false if they are unequal to 0 or not
  coefs <- coefs[,1]!=0
  # Correct if we have unequal to 0 for the first 10 (relevant ones)
  correctly <- sum(coefs[1:10])
  # Wrong if have unequal to 0 for the others
  wrongly <- sum(coefs[11:100])
  # Proportion of correct ones
  true_positive <- correctly/10
  # Proportion of wrong ones
  false_positive <- wrongly/90
  # Giving back a vector with the true positive and false positive values
  return(c(true_positive, false_positive))
}
```

```
# Draw the 100 datasets and rowbind the results and put it in a dataframe
sets <- as.data.frame(do.call(rbind, lapply(1:100, ex8)))

# Visualize True positive values of the 100 datasets
ggplot(data=sets, mapping=aes(x=1:100, y=V1)) +
  geom_point() +
  labs(y="",
       x="Simulations",
       title="True Positive")
```

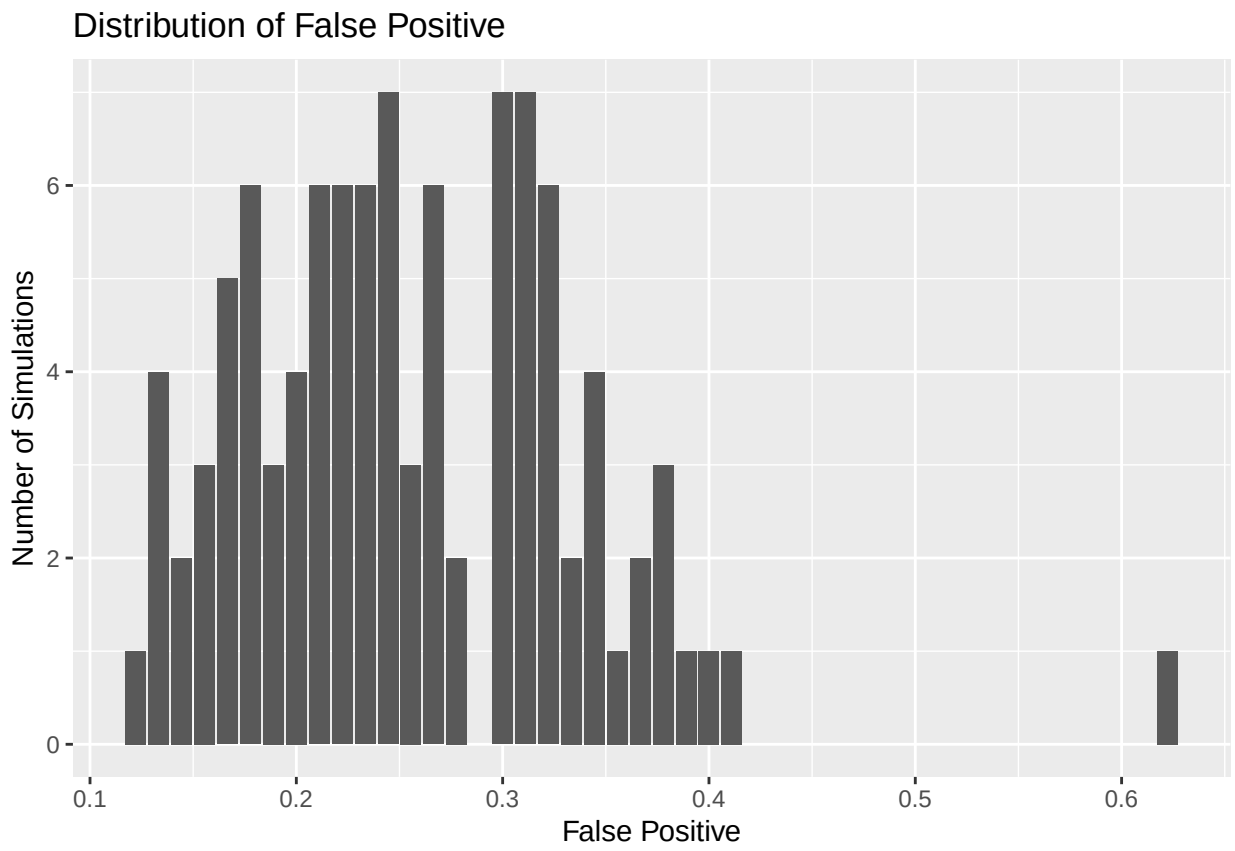


We can see that the proportion of correctly included coefficients from all relevant ones (true positive rate) is 1 for all 100 data sets in the simulation, which means that the lasso model works well for that.

```
# Visualize False positive values of the 100 datasets
ggplot(data=sets, mapping=aes(x=1:100, y=V2)) +
  geom_point() +
    labs(y="",
         x="Simulations",
         title="False Positive")
```




```
# Visualize distribution
ggplot(data=sets, mapping=aes(x=V2)) +
  geom_bar() +
    labs(y="Number of Simulations",
         x="False Positive",
         title="Distribution of False Positive")
```




```
# >Mean value of false positive
mean(sets[,2])
```

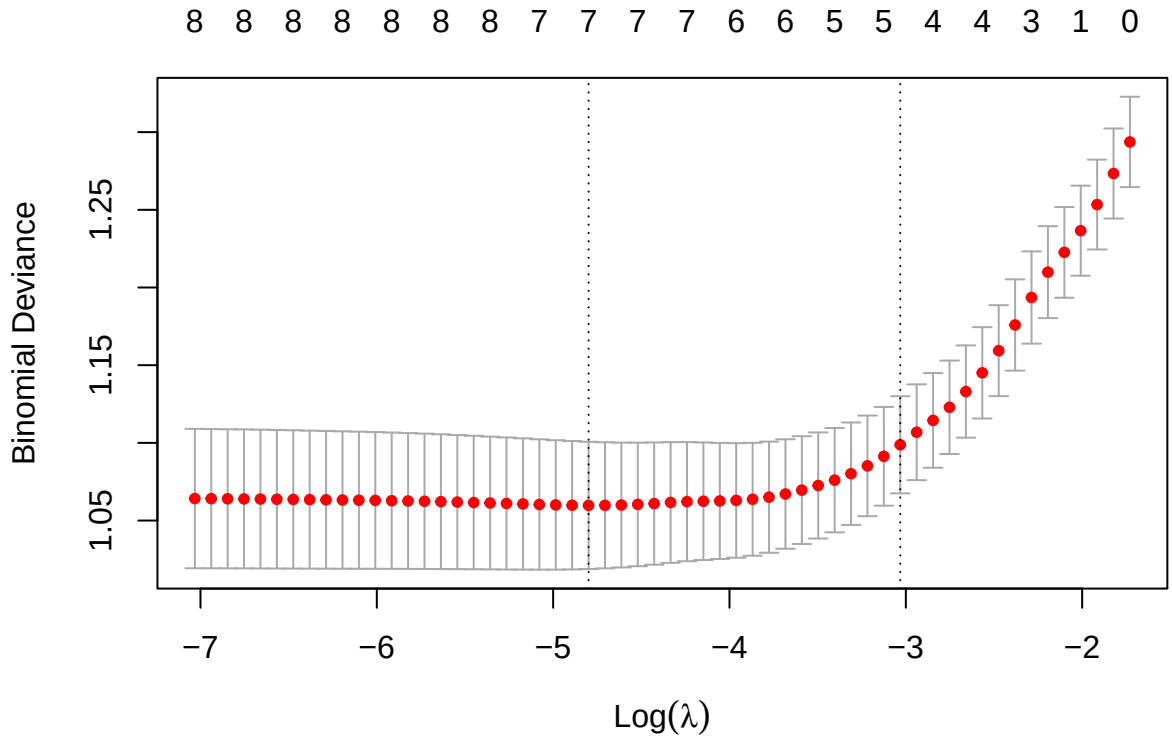
```
## [1] 0.2561111
```

We can see that the proportion of wrongly included coefficients from all irrelevant ones (false positive rate) are not constant like before, but fluctuate a lot across the simulations. We get that the average false positive rate is 0.256 and all of the rates stay below 0.42 except for one which is 0.62. Therefore the lasso model doesn't do the best job at not including irrelevant coefficients.

Task 9

```
# Import dataset
data("SAheart", package="ElemStatLearn")
df = SAheart
X<-cbind(df$sbp,df$tobacco,df$ldl,df$adiposity,factor(df$famhist),
         df$typea,df$obesity,df$alcohol,df$age)
colnames(X) <- c('sbp','tobacco','ldl','adiposity','famhist',
                 'typea','obesity','alcohol','age')

# Fit a logistic regression model with Lasso penalty using only
#linear effects for the covariates.
model<-glmnet(y = df$chd,x=X,family = "binomial",alpha = 1, nfolds=20)
```

```
model_cv<-cv.glmnet(y = df$chd,x=X,family = "binomial",alpha = 1, nfolds=20)

# Visualize the results
plot(model_cv)
```




```
# Penalty selection
cat("Penalty value which minimizes the cross-validation loss: ",model_cv$lambda.min)
```

```
## Penalty value which minimizes the cross-validation loss:  0.008236941
```

```
cat("Penalty value according to 1 - SE rule: ",model_cv$lambda.1se)
```

```
## Penalty value according to 1 - SE rule:  0.04824393
```

```
# Model selection
kable(as.matrix(cbind(coef(model_cv$glmnet.fit,s = model_cv$lambda.1se),
                      coef(model_cv$glmnet.fit,s = model_cv$lambda.min))),
      caption="Complexity Assessment", col.names = c("Lambda - 1SE", "Lambda - Min"))
```

Table 3: Complexity Assessment

	Lambda - 1SE	Lambda - Min
(Intercept)	-3.5109766	-6.6730770
sbp	0.0000000	0.0045921
tobacco	0.0424312	0.0720369
ldl	0.0778817	0.1533532
adiposity	0.0000000	0.0000000
famhist	0.4847274	0.8289014
typea	0.0044875	0.0311625
obesity	0.0000000	-0.0204137
alcohol	0.0000000	0.0000000
age	0.0314041	0.0447483

In terms of model complexity, we can see that the coefficients with the 1-SE lambda show that we eliminate 4 features whereas in the min lambda coefficients the algortihm eliminates only 2 features. That makes the min lambda model more complex. Which was expected as we increase the lambda, the penalty increases and more coefficients become zero and that removes less efficient features from the model. Choosing between the two models, we can say that the minimum lambda model is more prone to overfitting and the 1-SE model is a more conservative choice with fewer features eliminated.

```
preds_min <- ifelse(predict(model_cv, X, s = "lambda.min", type = "response")>0.5, 1, 0)

preds_1se <- ifelse(predict(model_cv, X, s = "lambda.1se", type = "response")>0.5, 1, 0)

expected=df$chd
temp = table(expected,preds_min)
kable(temp, caption="Confusion Matrix - Min")
```

Table 4: Confusion Matrix - Min

	0	1
0	261	41
1	76	84

```
temp2 = table(expected,preds_1se)
kable(temp2, caption="Confusion Matrix - 1-SE")
```

Table 5: Confusion Matrix - 1-SE

	0	1
0	282	20
1	99	61

From Table 4 and Table 5, we can see that the minimum lambda model did a better job correctly classifying the 1s(Yes) and the 1-SE model did a better job classifying the 0s(No). Minimum lambda model classified more observations as 1s and the 1-SE model classified more observations as 0s.

```r
cat("Misclassification error for minimum lambda: ", (temp[2]+temp[3])/sum(temp))
```

```
## Misclassification error for minimum lambda:  0.2532468
```

```r
cat("Misclassification error for 1-SE lambda: ", (temp2[2]+temp2[3])/sum(temp2))
```

```
## Misclassification error for 1-SE lambda:  0.2575758
```

In terms of misclassification error, minimum lambda model did a better job by correctly classifying the observations. However, the difference is not significant. Overall, in this case we do not want to miss the true 1s considering the medical context, therefore the choice should be the the model minimizing cross-validation loss.

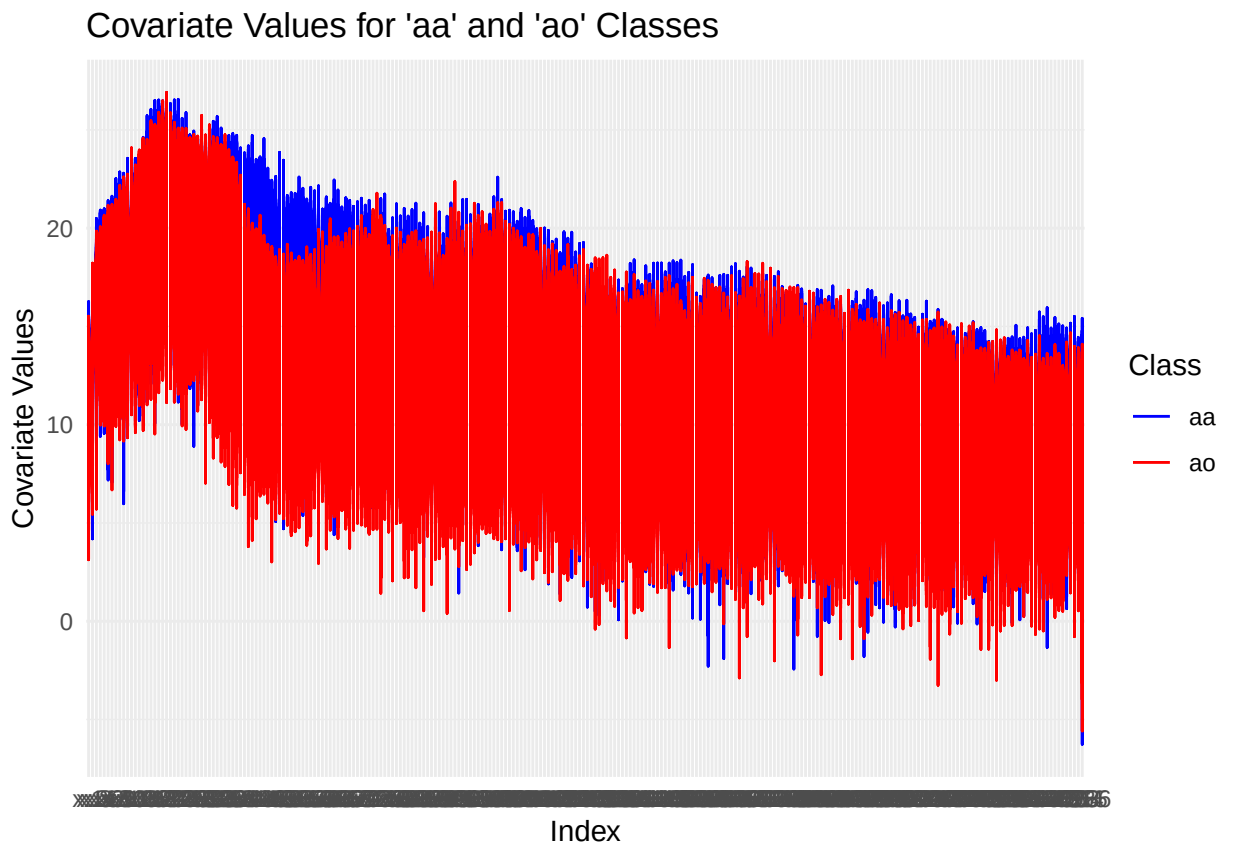
Task 10

```r
if(require("ElemStatLearn") == F) install.packages("ElemStatLearn_2015.6.26.1.tar.gz", repos = NULL)
# Check if the "ElemStatLearn" package is installed, and if not, install it from a local file.

# Load the ElemStatLearn package
library(ElemStatLearn)
# Load the "ElemStatLearn" package, which is used for machine learning and statistical analysis.

# Load the phoneme dataset
data(phoneme)
# Load the "phoneme" dataset, which is included in the "ElemStatLearn" package.
```

a) part

```r
# Filter the dataset to include only "aa" and "ao" classes
subset_phoneme <- phoneme[phoneme$g %in% c("aa", "ao"), ]
# Create a subset of the "phoneme" dataset, containing only the "aa" and "ao" classes.

# Create a line plot
plot_data <- subset_phoneme[, 1:256]  # Select the covariate columns
plot_data <- cbind(Class = subset_phoneme$g, plot_data)
# Prepare the data for plotting, selecting the covariate columns and adding a "Class" column.

# Load the ggplot2 library for data visualization
library(ggplot2)
# Load the "ggplot2" library for creating data visualizations.

# Reshape the data for plotting
plot_data <- reshape2::melt(plot_data, id.vars = "Class")
# Reshape the data for use in a line plot.

# Create a line plot
ggplot(plot_data, aes(x = variable, y = value, color = Class)) +
  geom_line() +
```

```
  labs(title = "Covariate Values for 'aa' and 'ao' Classes",
       x = "Index", y = "Covariate Values") +
  scale_color_manual(values = c("aa" = "blue", "ao" = "red")) +
  theme_minimal()
```




```
# Create and customize a line plot using ggplot2, showing the covariate values for "aa" and "ao" classe
```

b) part

```
# Set a random seed for reproducibility
set.seed(123)
# Set a random seed to ensure reproducibility of random processes.

data_aa_ao <- phoneme[(phoneme$g == "aa" | phoneme$g == "ao"), ]
# Create a subset of the "phoneme" dataset, containing only the "aa" and "ao" classes.

covariate_data <- data_aa_ao[, -ncol(data_aa_ao)]
# Extract the covariate data, excluding the last column.

covariate_matrix <- as.matrix(covariate_data)
int_sample <- sample.int(n = nrow(data_aa_ao), size = 1000, replace = F)
# Randomly sample 1000 rows from the covariate data for training.
```

```r
training_data <- covariate_data[int_sample,]
test_data <- covariate_data[-int_sample,]
# Split the data into training and test sets.

# Verify the dimensions of the training and test datasets
dim(training_data)
```

```
## [1] 1000  257
```

```r
dim(test_data)
```

```
## [1] 717 257
```

```r
# Check and display the dimensions of the training and test datasets.
```

c) part

```r
training_data$g2 <- ifelse(training_data$g == "aa", 0, 1) # aa corresponds to 0, ao to 1
training_data$g <- NULL
# Create a binary response variable, "g2," and remove the original "g" variable.

train_model <- glm(g2 ~ ., data = training_data, family = binomial(link = "logit"))
```

```
## Warning: glm.fit: Angepasste Wahrscheinlichkeiten mit numerischem Wert 0 oder 1
## aufgetreten
```

```r
summary(train_model)
```

```
##
## Call:
## glm(formula = g2 ~ ., family = binomial(link = "logit"), data = training_data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.7063  -0.0601   0.0031   0.1703   2.4018
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  9.382337   5.038925   1.862 0.062607 .
## x.1          0.140687   0.137948   1.020 0.307799
## x.2          0.307023   0.184628   1.663 0.096326 .
## x.3          0.103666   0.282869   0.366 0.714008
## x.4         -0.275962   0.288621  -0.956 0.339002
## x.5          0.406944   0.246803   1.649 0.099176 .
## x.6         -0.029014   0.232143  -0.125 0.900538
## x.7         -0.042102   0.290218  -0.145 0.884656
## x.8         -0.186789   0.250999  -0.744 0.456767
## x.9         -0.292370   0.241955  -1.208 0.226906
```

```
## x.10       -0.270683   0.281165  -0.963 0.335688
## x.11       -0.240639   0.237483  -1.013 0.310923
## x.12       -0.150193   0.261728  -0.574 0.566069
## x.13       -0.330645   0.230686  -1.433 0.151769
## x.14        0.566273   0.273333   2.072 0.038290 *
## x.15        0.425491   0.221052   1.925 0.054248 .
## x.16        0.500968   0.244088   2.052 0.040130 *
## x.17        0.880725   0.271249   3.247 0.001167 **
## x.18       -0.272031   0.211432  -1.287 0.198230
## x.19        0.105899   0.183723   0.576 0.564343
## x.20       -0.174805   0.200451  -0.872 0.383178
## x.21       -0.428002   0.202998  -2.108 0.034996 *
## x.22        0.085478   0.202485   0.422 0.672921
## x.23       -0.338885   0.175853  -1.927 0.053968 .
## x.24        0.086358   0.171195   0.504 0.613952
## x.25        0.300945   0.173708   1.732 0.083188 .
## x.26       -0.276805   0.210254  -1.317 0.187997
## x.27        0.601353   0.199626   3.012 0.002592 **
## x.28       -0.109007   0.191437  -0.569 0.569074
## x.29        0.281276   0.207738   1.354 0.175738
## x.30        0.153260   0.192266   0.797 0.425380
## x.31        0.049522   0.178590   0.277 0.781553
## x.32        0.406864   0.184742   2.202 0.027642 *
## x.33       -0.231869   0.185848  -1.248 0.212168
## x.34        0.214765   0.173743   1.236 0.216418
## x.35       -0.082703   0.170902  -0.484 0.628441
## x.36        0.223491   0.168924   1.323 0.185828
## x.37       -0.223614   0.170102  -1.315 0.188648
## x.38        0.096598   0.154740   0.624 0.532458
## x.39        0.201344   0.165915   1.214 0.224923
## x.40       -0.697597   0.204848  -3.405 0.000661 ***
## x.41        0.211453   0.193613   1.092 0.274769
## x.42       -0.492585   0.208549  -2.362 0.018178 *
## x.43        0.125067   0.198069   0.631 0.527757
## x.44        0.454600   0.211886   2.145 0.031913 *
## x.45       -0.849715   0.200251  -4.243  2.2e-05 ***
## x.46        0.228553   0.193499   1.181 0.237540
## x.47       -0.800150   0.231336  -3.459 0.000543 ***
## x.48       -0.586891   0.199675  -2.939 0.003290 **
## x.49        0.166542   0.189738   0.878 0.380083
## x.50       -0.214067   0.240858  -0.889 0.374128
## x.51        0.346747   0.174066   1.992 0.046366 *
## x.52        0.028606   0.191448   0.149 0.881221
## x.53       -0.084134   0.175535  -0.479 0.631726
## x.54        0.058661   0.192988   0.304 0.761157
## x.55        0.254824   0.194455   1.310 0.190042
## x.56       -0.622007   0.205503  -3.027 0.002472 **
## x.57        0.125975   0.195714   0.644 0.519792
## x.58       -0.337155   0.197131  -1.710 0.087209 .
## x.59       -0.430616   0.204188  -2.109 0.034952 *
## x.60        0.197600   0.193890   1.019 0.308139
## x.61       -0.083474   0.185607  -0.450 0.652903
## x.62        0.233235   0.202956   1.149 0.250477
## x.63       -0.326185   0.181071  -1.801 0.071636 .
```

```
## x.64         0.299429   0.150427   1.991 0.046533 *
## x.65        -0.407494   0.180860  -2.253 0.024254 *
## x.66         0.171180   0.185052   0.925 0.354947
## x.67        -0.146641   0.160541  -0.913 0.361025
## x.68         0.105955   0.185690   0.571 0.568268
## x.69         0.348798   0.187033   1.865 0.062195 .
## x.70        -0.101873   0.184600  -0.552 0.581044
## x.71        -0.027020   0.180383  -0.150 0.880927
## x.72         0.188095   0.159919   1.176 0.239518
## x.73        -0.343377   0.174743  -1.965 0.049409 *
## x.74         0.381139   0.179783   2.120 0.034007 *
## x.75        -0.195586   0.163256  -1.198 0.230904
## x.76        -0.070470   0.156139  -0.451 0.651753
## x.77        -0.222006   0.186593  -1.190 0.234130
## x.78        -0.117821   0.170490  -0.691 0.489521
## x.79         0.403711   0.177932   2.269 0.023274 *
## x.80         0.194839   0.158209   1.232 0.218125
## x.81         0.395231   0.171747   2.301 0.021378 *
## x.82         0.328710   0.179934   1.827 0.067725 .
## x.83        -0.021504   0.174865  -0.123 0.902129
## x.84         0.190985   0.182469   1.047 0.295251
## x.85        -0.295098   0.196267  -1.504 0.132696
## x.86        -0.341184   0.212310  -1.607 0.108053
## x.87        -0.273005   0.189435  -1.441 0.149542
## x.88        -0.432350   0.203820  -2.121 0.033902 *
## x.89         0.667848   0.199270   3.351 0.000804 ***
## x.90        -0.303706   0.180751  -1.680 0.092910 .
## x.91        -0.055404   0.168521  -0.329 0.742332
## x.92         0.108695   0.180379   0.603 0.546780
## x.93        -0.608188   0.189667  -3.207 0.001343 **
## x.94         0.281827   0.176979   1.592 0.111287
## x.95         0.138096   0.176037   0.784 0.432763
## x.96         0.046527   0.181710   0.256 0.797912
## x.97        -0.146877   0.182112  -0.807 0.419943
## x.98         0.407627   0.207545   1.964 0.049526 *
## x.99        -0.202307   0.200557  -1.009 0.313108
## x.100        0.051464   0.195052   0.264 0.791897
## x.101        0.109683   0.190447   0.576 0.564666
## x.102       -0.177412   0.212078  -0.837 0.402851
## x.103       -0.037153   0.185873  -0.200 0.841573
## x.104        0.310382   0.173722   1.787 0.073994 .
## x.105       -0.529881   0.196520  -2.696 0.007011 **
## x.106        0.172792   0.177289   0.975 0.329741
## x.107       -0.321074   0.184732  -1.738 0.082201 .
## x.108        0.326706   0.172717   1.892 0.058548 .
## x.109        0.250274   0.167826   1.491 0.135892
## x.110        0.154745   0.187699   0.824 0.409695
## x.111       -0.368676   0.204465  -1.803 0.071368 .
## x.112       -0.399234   0.186074  -2.146 0.031907 *
## x.113       -0.330445   0.187298  -1.764 0.077685 .
## x.114        0.031791   0.165633   0.192 0.847793
## x.115        0.427780   0.179224   2.387 0.016994 *
## x.116        0.128691   0.168427   0.764 0.444822
## x.117       -0.183748   0.158877  -1.157 0.247458
```

```
## x.118        0.223971   0.163955   1.366 0.171922
## x.119       -0.433688   0.190179  -2.280 0.022583 *
## x.120        0.175102   0.174428   1.004 0.315443
## x.121       -0.187846   0.183597  -1.023 0.306241
## x.122        0.289274   0.188516   1.534 0.124911
## x.123        0.179082   0.183071   0.978 0.327969
## x.124       -0.171273   0.180895  -0.947 0.343735
## x.125        0.490096   0.184787   2.652 0.007996 **
## x.126       -0.629356   0.183189  -3.436 0.000591 ***
## x.127        0.058045   0.188708   0.308 0.758394
## x.128       -0.028056   0.166406  -0.169 0.866112
## x.129        0.387657   0.181706   2.133 0.032889 *
## x.130        0.117677   0.165842   0.710 0.477971
## x.131       -0.288163   0.165287  -1.743 0.081261 .
## x.132        0.240731   0.167084   1.441 0.149648
## x.133       -0.358007   0.170104  -2.105 0.035323 *
## x.134        0.051494   0.153823   0.335 0.737805
## x.135        0.125636   0.164368   0.764 0.444654
## x.136        0.011685   0.168309   0.069 0.944651
## x.137       -0.279574   0.174395  -1.603 0.108911
## x.138       -0.338196   0.184325  -1.835 0.066538 .
## x.139        0.027819   0.170572   0.163 0.870446
## x.140       -0.090061   0.170073  -0.530 0.596427
## x.141        0.235748   0.192752   1.223 0.221307
## x.142        0.115505   0.168372   0.686 0.492707
## x.143        0.001752   0.177668   0.010 0.992134
## x.144        0.160517   0.160930   0.997 0.318551
## x.145       -0.138028   0.185728  -0.743 0.457377
## x.146       -0.117326   0.177921  -0.659 0.509621
## x.147        0.208488   0.167517   1.245 0.213287
## x.148        0.044196   0.172551   0.256 0.797848
## x.149        0.304735   0.161342   1.889 0.058925 .
## x.150       -0.218393   0.179251  -1.218 0.223084
## x.151       -0.019114   0.165899  -0.115 0.908276
## x.152       -0.077360   0.156921  -0.493 0.622020
## x.153       -0.304112   0.185405  -1.640 0.100951
## x.154        0.166424   0.187391   0.888 0.374482
## x.155       -0.038422   0.179367  -0.214 0.830382
## x.156        0.047103   0.149062   0.316 0.752004
## x.157       -0.267683   0.159935  -1.674 0.094189 .
## x.158        0.501146   0.176911   2.833 0.004615 **
## x.159        0.189011   0.174406   1.084 0.278481
## x.160        0.035043   0.163771   0.214 0.830567
## x.161        0.231514   0.184263   1.256 0.208959
## x.162       -0.123075   0.165982  -0.741 0.458393
## x.163        0.171480   0.158160   1.084 0.278269
## x.164        0.103409   0.190037   0.544 0.586337
## x.165       -0.180865   0.172961  -1.046 0.295700
## x.166        0.090625   0.163700   0.554 0.579849
## x.167       -0.545655   0.194457  -2.806 0.005015 **
## x.168        0.067356   0.186335   0.361 0.717741
## x.169        0.323400   0.173631   1.863 0.062522 .
## x.170       -0.285252   0.174188  -1.638 0.101503
## x.171       -0.168903   0.177572  -0.951 0.341513
```

```
## x.172       -0.207856   0.186991  -1.112 0.266319
## x.173       -0.207833   0.185956  -1.118 0.263717
## x.174        0.316495   0.180976   1.749 0.080321 .
## x.175        0.159098   0.184001   0.865 0.387227
## x.176        0.113790   0.167517   0.679 0.496966
## x.177       -0.108144   0.164940  -0.656 0.512045
## x.178        0.266156   0.159230   1.672 0.094619 .
## x.179        0.110571   0.186615   0.593 0.553511
## x.180        0.059291   0.172859   0.343 0.731595
## x.181       -0.138429   0.170872  -0.810 0.417863
## x.182       -0.287423   0.172594  -1.665 0.095850 .
## x.183       -0.121369   0.165455  -0.734 0.463225
## x.184       -0.012881   0.177453  -0.073 0.942136
## x.185        0.086481   0.177727   0.487 0.626547
## x.186        0.145936   0.157866   0.924 0.355260
## x.187       -0.147712   0.177931  -0.830 0.406445
## x.188       -0.202827   0.183898  -1.103 0.270058
## x.189        0.117366   0.185462   0.633 0.526845
## x.190       -0.071827   0.184392  -0.390 0.696882
## x.191        0.270621   0.183353   1.476 0.139956
## x.192        0.231848   0.181027   1.281 0.200285
## x.193        0.275122   0.173593   1.585 0.112996
## x.194       -0.411692   0.165693  -2.485 0.012967 *
## x.195        0.072078   0.160998   0.448 0.654374
## x.196        0.080073   0.176540   0.454 0.650139
## x.197        0.417319   0.198660   2.101 0.035670 *
## x.198       -0.401349   0.173557  -2.312 0.020751 *
## x.199        0.289651   0.190424   1.521 0.128237
## x.200       -0.208717   0.177922  -1.173 0.240763
## x.201       -0.551363   0.160299  -3.440 0.000583 ***
## x.202       -0.435283   0.183703  -2.369 0.017812 *
## x.203        0.242334   0.156341   1.550 0.121133
## x.204        0.098467   0.187383   0.525 0.599248
## x.205        0.196260   0.177329   1.107 0.268400
## x.206       -0.204867   0.147096  -1.393 0.163699
## x.207       -0.280012   0.176071  -1.590 0.111759
## x.208        0.028886   0.152773   0.189 0.850030
## x.209        0.132443   0.171661   0.772 0.440387
## x.210        0.457772   0.179656   2.548 0.010833 *
## x.211       -0.076419   0.192835  -0.396 0.691890
## x.212        0.094230   0.168685   0.559 0.576427
## x.213       -0.751197   0.193644  -3.879 0.000105 ***
## x.214        0.087420   0.178068   0.491 0.623474
## x.215        0.706913   0.209342   3.377 0.000733 ***
## x.216        0.155720   0.170370   0.914 0.360711
## x.217       -0.114304   0.182077  -0.628 0.530148
## x.218       -0.186998   0.186489  -1.003 0.315991
## x.219        0.502447   0.177130   2.837 0.004560 **
## x.220        0.067234   0.173030   0.389 0.697593
## x.221        0.355406   0.183942   1.932 0.053339 .
## x.222       -0.374230   0.165249  -2.265 0.023535 *
## x.223       -0.237813   0.186616  -1.274 0.202541
## x.224        0.017949   0.188729   0.095 0.924231
## x.225       -0.388635   0.181639  -2.140 0.032388 *
```

```
## x.226        0.510235   0.166034   3.073 0.002119 **
## x.227       -0.707384   0.185130  -3.821 0.000133 ***
## x.228       -0.175206   0.187122  -0.936 0.349108
## x.229        0.453087   0.176842   2.562 0.010404 *
## x.230        0.311308   0.172576   1.804 0.071248 .
## x.231        0.075715   0.189943   0.399 0.690174
## x.232       -0.070532   0.186224  -0.379 0.704874
## x.233       -0.045861   0.187551  -0.245 0.806822
## x.234       -0.162875   0.175421  -0.928 0.353156
## x.235       -0.272684   0.174725  -1.561 0.118607
## x.236        0.458390   0.167873   2.731 0.006322 **
## x.237        0.213015   0.186576   1.142 0.253578
## x.238       -0.121987   0.183743  -0.664 0.506755
## x.239       -0.006045   0.197330  -0.031 0.975561
## x.240       -0.190168   0.158391  -1.201 0.229897
## x.241       -0.217798   0.182296  -1.195 0.232185
## x.242        0.586161   0.212806   2.754 0.005879 **
## x.243       -0.107697   0.179949  -0.598 0.549514
## x.244       -0.133465   0.201855  -0.661 0.508489
## x.245       -0.157479   0.172782  -0.911 0.362068
## x.246       -0.132900   0.162348  -0.819 0.413008
## x.247       -0.413955   0.194976  -2.123 0.033745 *
## x.248       -0.140025   0.171153  -0.818 0.413285
## x.249        0.473107   0.196936   2.402 0.016291 *
## x.250        0.013097   0.180014   0.073 0.941999
## x.251       -0.257561   0.165483  -1.556 0.119608
## x.252       -0.101056   0.166328  -0.608 0.543470
## x.253       -0.122391   0.169609  -0.722 0.470536
## x.254        0.007692   0.155572   0.049 0.960564
## x.255        0.056575   0.190471   0.297 0.766445
## x.256        0.033462   0.100360   0.333 0.738814
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1351.50  on 999  degrees of freedom
## Residual deviance:  399.65  on 743  degrees of freedom
## AIC: 913.65
##
## Number of Fisher Scoring iterations: 9
```

```r
# Fit a logistic regression model to the training data and display the summary of the model.

predicted_probs_train <- predict.glm(train_model, newdata = training_data, type="response")
predicted_probs_test <- predict.glm(train_model, newdata = test_data, type="response")
# Generate predicted probabilities for both the training and test datasets.

predicted_labels_train <- ifelse(predicted_probs_train > 0.5, 1, 0)
predicted_labels_test <- ifelse(predicted_probs_test > 0.5, 1, 0)
# Convert predicted probabilities to binary labels based on a threshold of 0.5.

misclass_rate_train <- mean(predicted_labels_train != training_data$g2)
misclass_rate_test <- mean(predicted_labels_test != ifelse(test_data$g == "aa", 0, 1))
```

```r
# Calculate misclassification rates for training and test datasets.

misclass_rate_train
```

```
## [1] 0.082
```

```r
misclass_rate_test
```

```
## [1] 0.2426778
```

```r
# Display the misclassification rates for training and test datasets.

result_df <- t(data.frame("Misclassification.Rate_train" = round(misclass_rate_train,4),
                          "MisclassificationRate_test" = round(misclass_rate_test,4)))
# Create a data frame to store the misclassification rates and round the values to 4 decimal places.

avg_loglik <- function(actual, predicted_prob) {
    sum(actual * log(predicted_prob) + (1-actual) * log(1-predicted_prob)) / length(actual)
}
# Define a function to calculate the average log-likelihood.

test_data$g2 <- ifelse(test_data$g == "aa", 0 ,1)
# Create a binary response variable for the test data.

avg_loglik_train <- avg_loglik(training_data$g2, predicted_probs_train)
avg_loglik_train
```

```
## [1] -0.1998255
```

```r
avg_loglik_test <- avg_loglik(test_data$g2, predicted_probs_test)
avg_loglik_test
```

```
## [1] -1.155733
```

```r
# Calculate average log-likelihood for training and test datasets.

result_df <- rbind.data.frame(result_df, t(data.frame("Average.LogLikelihood_train" = round(avg_loglik_
                                                      "Average.LogLikelihood_test" =  round(avg_loglik_
colnames(result_df) <- "Full Model"
# Add the average log-likelihood values to the result data frame and assign column names.
```

d) part

```r
library(splines)
H <- ns(1:256, df = 12)
X.star <- as.matrix(covariate_data[,-ncol(covariate_data)]) %*% H
# Create a natural spline basis with 12 degrees of freedom and apply it to the covariate data.
```

```r
X.star <- as.data.frame(X.star)
X.star$g2 <- ifelse(covariate_data$g == "aa", 0, 1)
X.star_training <- X.star[int_sample,]
X.star_test <- X.star[-int_sample,]
# Prepare the data with the spline basis for modeling.

train_model_spline <- glm(g2 ~ ., data = X.star_training, family = binomial(link = "logit"))
summary(train_model_spline)
```

```
##
## Call:
## glm(formula = g2 ~ ., family = binomial(link = "logit"), data = X.star_training)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.9042  -0.5343   0.2193   0.6283   2.3608
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)  2.5205537  1.8722588   1.346  0.17822
## `1`         -0.1325426  0.0135463  -9.784  < 2e-16 ***
## `2`          0.0392775  0.0121071   3.244  0.00118 **
## `3`         -0.0006257  0.0086596  -0.072  0.94240
## `4`         -0.0091263  0.0089039  -1.025  0.30537
## `5`          0.0070676  0.0084221   0.839  0.40137
## `6`         -0.0048393  0.0094934  -0.510  0.61022
## `7`          0.0087804  0.0103364   0.849  0.39562
## `8`         -0.0129893  0.0111142  -1.169  0.24252
## `9`          0.0132786  0.0119824   1.108  0.26778
## `10`        -0.0351997  0.0134641  -2.614  0.00894 **
## `11`         0.0712453  0.0100292   7.104 1.21e-12 ***
## `12`        -0.0638403  0.0134233  -4.756 1.98e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1351.50  on 999  degrees of freedom
## Residual deviance:  789.99  on 987  degrees of freedom
## AIC: 815.99
##
## Number of Fisher Scoring iterations: 5
```

```r
# Fit a logistic regression model to the data with the spline basis and display the model summary.

predicted_probs_train_spline <- predict.glm(train_model_spline, newdata = X.star_training, type="respons
predicted_probs_test_spline <- predict.glm(train_model_spline, newdata = X.star_test, type="response")
# Generate predicted probabilities for training and test datasets with the spline model.

predicted_labels_train_spline <- ifelse(predicted_probs_train_spline > 0.5, 1, 0)
predicted_labels_test_spline <- ifelse(predicted_probs_test_spline > 0.5, 1, 0)
# Convert predicted probabilities to binary labels based on a threshold of 0.5.
```

```r
misclass_rate_train_spline <- mean(predicted_labels_train_spline != X.star_training$g2)
misclass_rate_test_spline <- mean(predicted_labels_test_spline != X.star_test$g2)
# Calculate misclassification rates for training and test datasets with the spline model.

misclass_rate_train_spline_12df <- misclass_rate_train_spline
misclass_rate_train_spline_12df
```

```
## [1] 0.182
```

```r
misclass_rate_test_spline_12df <- misclass_rate_test_spline
misclass_rate_test_spline_12df
```

```
## [1] 0.1603905
```

```r
# Display misclassification rates for training and test datasets with the spline model.

avg_loglik_train_spline_12df <- avg_loglik(X.star_training$g2, predicted_probs_train_spline)
avg_loglik_train_spline_12df
```

```
## [1] -0.3949975
```

```r
avg_loglik_test_spline_12df <- avg_loglik(X.star_test$g2, predicted_probs_test_spline)
avg_loglik_test_spline_12df
```

```
## [1] -0.3914479
```

```r
# Calculate average log-likelihood for training and test datasets with the spline model.

result_df$Spline.12df <- round(c(misclass_rate_train_spline_12df, misclass_rate_test_spline_12df,
                          avg_loglik_train_spline_12df, avg_loglik_test_spline_12df),4)
# Add the misclassification rates and average log-likelihood values to the result data frame for the sp
```

e) part

```r
misclassification_rate <- function(data_training, data_test, dec.numbers = 4) {

  model_train <- glm(g2 ~ ., data = data_training, family = binomial(link = "logit"))

  predicted_probs_train <- predict.glm(model_train, newdata = data_training, type = "response")
  predicted_probs_test <- predict.glm(model_train, newdata = data_test, type = "response")

  predicted_labels_train <- ifelse(predicted_probs_train > 0.5, 1, 0)
  predicted_labels_test <- ifelse(predicted_probs_test > 0.5, 1, 0)

  misclass_rate_train <- mean(predicted_labels_train != data_training$g2)
  misclass_rate_test <- mean(predicted_labels_test != data_test$g2)

  return(list(model_train, round(misclass_rate_train, dec.numbers), predicted_probs_train,
```

```
              round(misclass_rate_test, dec.numbers), predicted_probs_test))
}


library(knitr)

spline_results_AIC <- numeric(8)
spline_results_misclassification <- c()
spline_results_avg_loglik <- c()
for (i in 1:8) {
  H.tmp <- ns(1:256, df = 2^i)

  X.star.tmp <- as.matrix(covariate_data[,-ncol(covariate_data)]) %*% H.tmp
  X.star.tmp <- as.data.frame(X.star.tmp)
  X.star.tmp$g2 <- ifelse(covariate_data$g == "aa", 0, 1)

  X.star.tmp_training <- X.star.tmp[int_sample,]
  X.star.tmp_test <- X.star.tmp[-int_sample,]

  tmp <- misclassification_rate(X.star.tmp_training, X.star.tmp_test)

  model_tmp <- tmp[[1]] # Extracting the model from the results
  spline_results_AIC[i] <- round(AIC(model_tmp),4)

  avg_loglik.tmp_train <- round(avg_loglik(X.star.tmp_training$g2, tmp[[3]]), 4)
  avg_loglik.tmp_test <- round(avg_loglik(X.star.tmp_test$g2, tmp[[5]]), 4)

  spline_results_avg_loglik <- rbind(spline_results_avg_loglik, c(avg_loglik.tmp_train, avg_loglik.tmp_
  spline_results_misclassification <- rbind(spline_results_misclassification, tmp[c(2,4)])
}
```

```
## Warning: glm.fit: Algorithmus konvergierte nicht
```

```
## Warning: glm.fit: Angepasste Wahrscheinlichkeiten mit numerischem Wert 0 oder 1
## aufgetreten
```

```
rownames(spline_results_misclassification) <- 1:8
colnames(spline_results_misclassification) <- c("Train.Data", "Test.Data")
colnames(spline_results_avg_loglik) <- c("Train.Data", "Test.Data")

result_df <- cbind(result_df,t(cbind(spline_results_misclassification, spline_results_avg_loglik)))

# Extract the column names
cols <- colnames(result_df)[3:10]

# Rename these columns to the desired format
new_names <- paste0("Spline.", 2^(1:8), "df", sep = "")

# Apply the new names to the dataframe
names(result_df)[names(result_df) %in% cols] <- new_names

result_df <- rbind(result_df, c(round(AIC(train_model),4), round(AIC(train_model_spline),4), spline_resu
```

```
result_df <- rbind(result_df, c(NA, 12, 2^(1:8)))
rownames(result_df) <- c("MR_Train", "MR_Test", "AvgLL_Train", "AvgLL_Test", "AIC", "DF")
```

f) part

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

```
## Warning: Paket 'ggthemes' wird gerade benutzt und deshab nicht installiert
```

```
library(ggthemes)

spline.data <- as.data.frame(t(result_df[,-1]))

# Convert DF to numeric
spline.data$DF <- as.numeric(spline.data$DF)

# Convert list columns to numeric vectors
cols_to_convert <- colnames(spline.data)[1:(ncol(spline.data)-1)]
spline.data[cols_to_convert] <- lapply(spline.data[cols_to_convert], unlist)
spline.data[cols_to_convert] <- lapply(spline.data[cols_to_convert], as.numeric)

# Confirm the conversion
str(spline.data)
```

```
## 'data.frame':    9 obs. of  6 variables:
##  $ MR_Train   : num  0.182 0.332 0.248 0.224 0.184 0.171 0.162 0.134 0.079
##  $ MR_Test    : num  0.16 0.331 0.243 0.198 0.16 ...
##  $ AvgLL_Train: num  -0.395 -0.609 -0.513 -0.463 -0.394 ...
##  $ AvgLL_Test : num  -0.391 -0.613 -0.507 -0.464 -0.392 ...
##  $ AIC        : num  816 1223 1035 945 821 ...
##  $ DF         : num  12 2 4 8 16 32 64 128 256
```

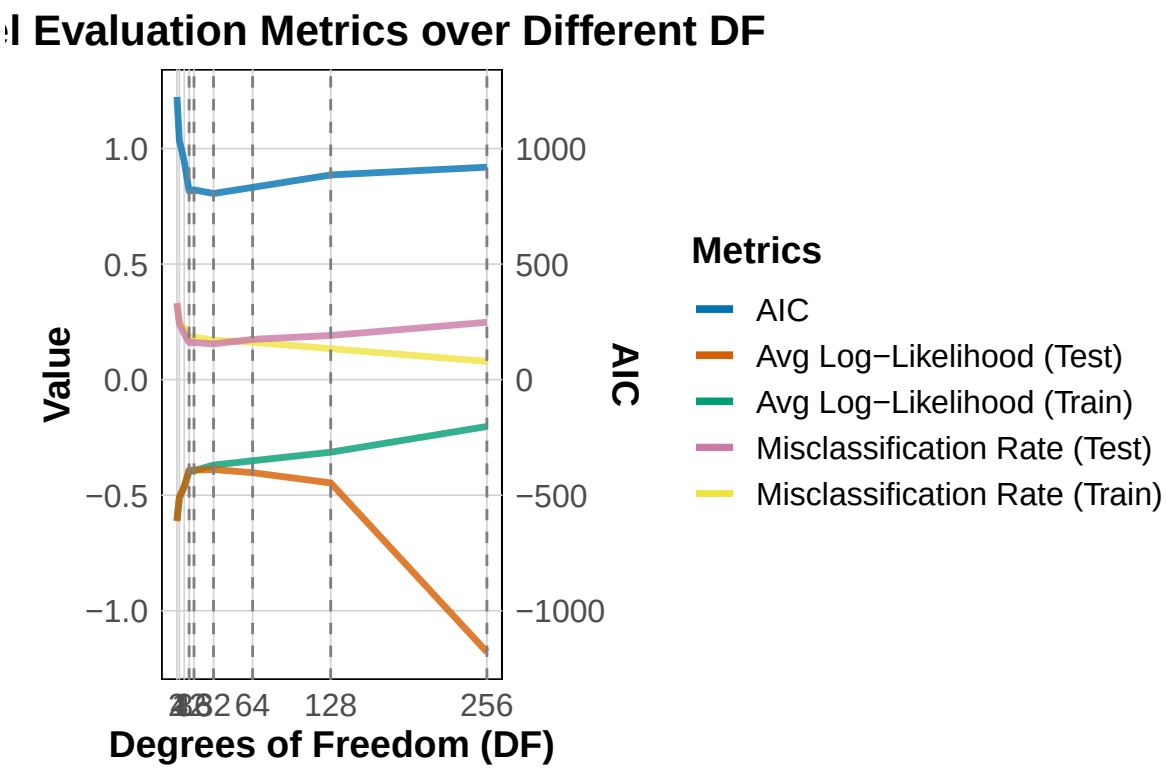
```
# Define a custom color palette
custom_palette <- c("#0072B2", "#D55E00", "#009E73", "#CC79A7", "#F0E442")

# Create a ggplot object
plot <- ggplot(spline.data, aes(x = DF)) +
  geom_line(aes(y = MR_Train, color = "Misclassification Rate (Train)"), size = 1.2, alpha = 0.8) +
  geom_line(aes(y = MR_Test, color = "Misclassification Rate (Test)"), size = 1.2, alpha = 0.8) +
  geom_line(aes(y = AvgLL_Train, color = "Avg Log-Likelihood (Train)"), size = 1.2, alpha = 0.8) +
  geom_line(aes(y = AvgLL_Test, color = "Avg Log-Likelihood (Test)"), size = 1.2, alpha = 0.8) +
  geom_line(aes(y = AIC/1000, color = "AIC"), size = 1.2, alpha = 0.8) +
  labs(title = "Model Evaluation Metrics over Different DF",
       x = "Degrees of Freedom (DF)",
       y = "Value",
       color = "Metrics") +
  scale_color_manual(values = custom_palette) +
  scale_x_continuous(name = "Degrees of Freedom (DF)", breaks = spline.data$DF) +
  theme_minimal() +
```

```
  theme(
    plot.title = element_text(size = 16, face = "bold", hjust = 0.5),
    axis.title.x = element_text(size = 14, face = "bold"),
    axis.title.y = element_text(size = 14, face = "bold"),
    axis.text = element_text(size = 12),
    legend.title = element_text(size = 14, face = "bold"),
    legend.text = element_text(size = 12),
    panel.grid.major = element_line(color = "lightgray", size = 0.3),
    panel.grid.minor = element_blank(),
    panel.background = element_rect(fill = "white"),
    plot.margin = unit(c(1, 1, 0.5, 0.5), "cm")
  ) +
  scale_y_continuous(sec.axis = sec_axis(~.*1000, name = "AIC")) +
  geom_vline(aes(xintercept = 12), linetype = "dashed", color = "grey50", size = 0.5) +
  geom_vline(aes(xintercept = 16), linetype = "dashed", color = "grey50", size = 0.5) +
  geom_vline(aes(xintercept = 32), linetype = "dashed", color = "grey50", size = 0.5) +
  geom_vline(aes(xintercept = 64), linetype = "dashed", color = "grey50", size = 0.5) +
  geom_vline(aes(xintercept = 128), linetype = "dashed", color = "grey50", size = 0.5) +
  geom_vline(aes(xintercept = 256), linetype = "dashed", color = "grey50", size = 0.5)
```

```
## Warning: Using ‘size‘ aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use ‘linewidth‘ instead.
## This warning is displayed once every 8 hours.
## Call ‘lifecycle::last_lifecycle_warnings()‘ to see where this warning was
## generated.
```

```
## Warning: The ‘size‘ argument of ‘element_line()‘ is deprecated as of ggplot2 3.4.0.
## i Please use the ‘linewidth‘ argument instead.
## This warning is displayed once every 8 hours.
## Call ‘lifecycle::last_lifecycle_warnings()‘ to see where this warning was
## generated.
```

```
# Print the plot
print(plot)
```



Upon closer examination of the plot, a notable trend emerges: an increase in degrees of freedom corresponds to a decline in the misclassification rate on the training subset. In other words, as the model becomes more complex, it becomes better at fitting the training data.

However, this apparent training improvement comes at a price. As degrees of freedom increase, the misclassification rate on the test subset also rises, suggesting that the model is starting to overfit the data. In essence, it's becoming too tailored to the training set and is losing its ability to generalize to new, unseen data.

A similar pattern arises when considering the average log-likelihood. With more degrees of freedom, the model's performance on the training set improves, with the average log-likelihood approaching 0. This implies an excellent fit to the training data. But, once again, this progress is accompanied by overfitting. The average log-likelihood on the test set diverges from 0 as degrees of freedom exceed 12, signifying a loss of generalization.

The AIC, a measure that balances model fit and complexity, follows a specific trajectory. It consistently decreases until reaching the spline model with 12 degrees of freedom. Beyond this point, the AIC begins to rise, indicating that the model's complexity is outweighing its benefit in explaining the data.

In summary, the evidence suggests that the spline model with 12 degrees of freedom strikes an ideal balance. It offers a reasonably complex model that fits the data well without falling into the trap of overfitting. This model appears to be the best compromise between complexity and generalization.