Assignment 1

Group 2

2023-10-09

Task 1

Consider a balanced regression problem where for each input $x_i, i = 1, ..., N$, one has J repeated outputs $y_i j, j = 1, ..., J$ and one fits a parameterized model $f_\theta(x)$ by least squares.

- Show that the fit can be obtained from a least squares problem involving only xi and the average values $\overline{y_i} = \frac{1}{J} \sum_{j=1}^{J} y_{ij}$

The problem basically says that we have N groups, and for each we have J realizations y_{ij}. Therefore, we will indicate from now on the total number of observations as $K = N \cdot J$. In the first step we subtract and add the groups means. We thus define the RSS as the following:

$$
\begin{aligned}
&RSS = \sum_{k}^{K} (y_k - f_\theta(x_k))^2 = \\
&\sum_{k}^{K} (y_k - \overline{y_k} + \overline{y_k} - f_\theta(x_k))^2 = \\
&\sum_{k}^{K} (y_k - \overline{y_k})^2 + 2 \sum_{k}^{K} (y_k - \overline{y_k})(\overline{y_k} - f_\theta(x_k)) + \sum_{k}^{K} (\overline{y_k} - f_\theta(x_k))^2 = \\
&\sum_{k}^{K} (y_k - \overline{y_k})^2 + 2 \sum_{i}^{N} (\overline{y_i} - f_\theta(x_i)) \sum_{j}^{J} (y_{ij} - \overline{y_i}) + \sum_{i}^{N} \sum_{j}^{J} (\overline{y_i} - f_\theta(x_i))^2
\end{aligned}
$$

Now we make two observations. The first is that in the first term we obtain a constant which does not depend on θ. Secondly, the term $\sum_{j}^{J} (y_{ij} - \overline{y_i})$ equals 0, since they are distances from the mean. Hence, we remain with the last term, which is precisely what we want to show.

- Explain how the least squares problem changes if the design is not balanced, i.e., one has different number of repetitions for each input x_i

If we add the point that we have different number of observations for each group, then we have that:

$$
RSS = \sum_{i}^{N} w_i (\overline{y_i} - f_\theta(x_i))^2
$$

where w_i is the number of observations for each group i.

Task 2

Consider a set of training data $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)$ drawn at random from a population and some test data $(\tilde{\boldsymbol{x}}_1, \tilde{y}_1), \ldots, (\tilde{\boldsymbol{x}}_M, \tilde{y}_M)$ also drawn at random from the same population as the training data are given.

We want to show that:

$$\mathrm{E}\left[R_{tr}(\hat{\beta})\right] \leq \mathrm{E}\left[R_{tr}\left(\mathrm{E}\left(\hat{\beta}\right)\right)\right] = \mathrm{E}\left[R_{te}\left(\mathrm{E}\left(\hat{\beta}\right)\right)\right] \leq \mathrm{E}\left[R_{te}(\hat{\beta})\right]$$

where $R_{tr}(\hat{\beta}) = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - \boldsymbol{x}_i^\top \hat{\beta}\right)^2$ and $R_{tr}(\hat{\beta}) = \frac{1}{M}\sum_{i=1}^{M}\left(y_i - \boldsymbol{x}_i^\top \hat{\beta}\right)^2$

Solution:

1. **Proving the left inequality:**

   Starting with the definition of $\hat{\beta}$ obtained from the training data:

   $$\hat{\beta} = \arg\min_{\beta'} \frac{1}{N}\sum_{i=1}^{N}\left(y_i - \boldsymbol{x}_i^\top \beta'\right)^2$$

   Since $\hat{\beta}$ minimizes the training error, we have:

   $$R_{tr}(\hat{\beta}) \leq R_{tr}(\beta)$$

   Now, taking the expectation of both sides:

   $$\mathrm{E}\left[R_{tr}(\hat{\beta})\right] \leq \mathrm{E}\left[R_{tr}(\beta)\right]$$

   The random variable $\hat{\beta}$ depends on the training data, but we can take its expectation, resulting in a fixed, non-random vector $\mathrm{E}(\hat{\beta})$. Substituting this into the above inequality:

   $$\mathrm{E}\left[R_{tr}(\hat{\beta})\right] \leq \mathrm{E}\left[R_{tr}(\mathrm{E}(\hat{\beta}))\right]$$

   This proves the left inequality.

2. **Proving the middle equality:**

   For any fixed vector β, the expected training error $\mathrm{E}[R_{tr}(\beta)]$ and the expected test error $\mathrm{E}[R_{te}(\beta)]$ are given by:

   $$\mathrm{E}[R_{tr}(\beta)] = \frac{1}{N}\sum_{i=1}^{N}\mathrm{E}\left[(y_i - \boldsymbol{x}_i^\top \beta)^2\right] = \mathrm{E}\left[(Y - \boldsymbol{X}^\top \beta)^2\right]$$

   $$\mathrm{E}[R_{te}(\beta)] = \frac{1}{M}\sum_{i=1}^{M}\mathrm{E}\left[(\tilde{y}_i - \tilde{\boldsymbol{x}}_i^\top \beta)^2\right] = \mathrm{E}\left[(\tilde{Y} - \tilde{\boldsymbol{X}}^\top \beta)^2\right]$$

   This equality holds because both the training and test data come from the same distribution.

3. **Proving the right inequality:**

   We treat $\hat{\beta}$ as a random vector independent from the test data due to the independence between the training and test data. For this part, we can forget about the training data and think of $\hat{\beta}$ as a random vector independent of the (test) data.

   Starting with the expected test error:

$$
\begin{aligned}
\mathrm{E}[R_{te}(\hat{\beta})] &= \mathrm{E}\left(\tilde{Y} - \tilde{\boldsymbol{X}}^\top \hat{\beta}\right)^2 \\
&= \mathrm{E}\left(\tilde{Y}^2 - 2\tilde{Y}\tilde{\boldsymbol{X}}^\top \hat{\beta} + \left(\tilde{\boldsymbol{X}}^\top \hat{\beta}\right)^2\right) \\
&= \tilde{Y}^2 - 2\tilde{Y}\tilde{\boldsymbol{X}}^\top \mathrm{E}\left(\hat{\beta}\right) + \tilde{\boldsymbol{X}}^\top \mathrm{E}\left(\hat{\beta}\hat{\beta}^\top\right)\tilde{\boldsymbol{X}} \\
&= \tilde{Y}^2 - 2\tilde{Y}\tilde{\boldsymbol{X}}^\top \mathrm{E}\left(\hat{\beta}\right) + \tilde{\boldsymbol{X}}^\top \left[\mathrm{E}\left(\hat{\beta}\right)\mathrm{E}\left(\hat{\beta}^\top\right) + Cov\left(\hat{\beta}\right)\right]\tilde{\boldsymbol{X}}
\end{aligned}
$$

Since the covariance matrix is positive semi-definite, $X^T \operatorname{Cov}(\beta) X \geq 0$, we have:

$$
\begin{aligned}
\mathrm{E}[R_{te}(\hat{\beta})] &\geq \tilde{Y}^2 - 2\tilde{Y}\tilde{\boldsymbol{X}}^\top \mathrm{E}(\hat{\beta}) + \tilde{\boldsymbol{X}}^\top \mathrm{E}(\hat{\beta})\mathrm{E}(\hat{\beta}^\top)\tilde{\boldsymbol{X}} \\
&\geq \left(\tilde{Y} - \tilde{\boldsymbol{X}}^\top \mathrm{E}(\hat{\beta})\right)^2
\end{aligned}
$$

This implies:

$$
\mathrm{E}[R_{te}(\hat{\beta})] \geq \mathrm{E}\left(\tilde{Y} - \tilde{\boldsymbol{X}}^\top \mathrm{E}\left(\hat{\beta}\right)\right)^2 = \mathrm{E}\left(R_{te}(\mathrm{E}\left(\hat{\beta}\right))\right)
$$

Thus, we've proved the right inequality.

Therefore, we have established that:

$$
\mathrm{E}\left[\frac{1}{N}\sum_{i=1}^{N}\left(y_i - \boldsymbol{x}_i^\top \hat{\beta}\right)^2\right] \leq \mathrm{E}\left[\frac{1}{M}\sum_{i=1}^{M}\left(y_i - \boldsymbol{x}_i^\top \hat{\beta}\right)^2\right]
$$

Task 3

We have:

$$
Y = \beta_0 + \beta_1 X + \epsilon
$$

where:

Y is the dependent variable,
X is the independent variable,
β_0 and β_1 are the regression coefficients, and
ϵ is the error term.

The predicted value of Y can be expressed as:

$$
\hat{Y} = \beta_0 + \beta_1 X
$$

The sum of squared errors (SSE) can be expressed as:

$$
SSE = \sum (Y_i - \hat{Y}_i)^2
$$

The total sum of squares (SST) can be expressed as:

$$SST = \sum (Y_i - \bar{Y})^2$$

where:

Y_i is the observed value of Y,
$\hat{Y}_i$ is the predicted value of Y,
$\bar{Y}$ is the mean of Y.

The coefficient of determination (R^2) is defined as:

$$R^2 = 1 - \frac{SSE}{SST}$$

To find the values of β_0 and β_1 that minimize SSE, we take the partial derivatives of SSE with respect to β_0 and β_1 and set them equal to zero. This gives us the following equations:

$$\sum Y_i = n\beta_0 + \beta_1 \sum X_i$$

$$\sum (Y_i X_i) = \beta_0 \sum X_i + \beta_1 \sum (X_i^2)$$

Solving for β_0 and β_1, we get:

$$\beta_1 = \frac{\sum (X_i - \bar{X})(Y_i - \bar{Y})}{\sum (X_i - \bar{X})^2}$$

$$\beta_0 = \bar{Y} - \beta_1 \bar{X}$$

where:

$\bar{X}$ is the mean of X,
$\bar{Y}$ is the mean of Y.

The correlation coefficient between X and Y can be expressed as:

$$r_{xy} = \frac{\sum (X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum (X_i - \bar{X})^2 \sum (Y_i - \bar{Y})^2}}$$

Substituting the values of β_0 and β_1 into the formula for $\hat{Y}_i$, we get:

$$\hat{Y}_i = \beta_0 + \beta_1 X_i = \bar{Y} + r_{xy} \left(Y_i - \bar{Y}\right) / (s_Y \sqrt{s_X^2})$$

where:

s_X is the standard deviation of X,
s_Y is the standard deviation of Y.

Substituting this into the formula for SST, we get:

$$SST = \sum (Y_i - \bar{Y})^2 = \sum (Y_i - \hat{Y}_i + \hat{Y}_i - \bar{Y})^2 = \sum (Y_i - \hat{Y}_i)^2 + \sum (\hat{Y}_i - \bar{Y})^2$$

The second term in this equation can be expressed as:

$$\sum (\hat{Y}_i - \bar{Y})^2 = r_{xy}^2 \sum (Y_i - \bar{Y})^2 / (s_Y^2)$$

Substituting this into the formula for R^2, we get:

$$R^2 = 1 - \frac{SSE}{SST} = 1 - \frac{\sum (Y_i - \hat{Y}_i)^2}{\sum (Y_i - \bar{Y})^2} = r_{xy}^2$$

Therefore, we have shown that the coefficient of determination given by the R^2 statistic is equal to the square of the correlation between X and Y in the simple linear regression case, where $Y = \beta_0 + \beta_1 X + \epsilon$ and the regression coefficients (β_0, β_1) are estimated using ordinary least squares.

Task 4

OLS estimator:

$$\hat{\beta} = (X^T X)^{-1} X^T y$$

Ridge estimator:

$$\hat{\beta}^{ridge}(\lambda) = (X^T X + \lambda I)^{-1} X^T y$$

For X orthonormal we get:

OLS estimator:

$$X^T y$$

Ridge estimator:

$$\hat{\beta}^{ridge}(\lambda) = ((1 + \lambda) I)^{-1} X^T y$$

Now we look at the MSE, which we can get by the formula:

$$\mathbb{E}[(\hat{\beta} - \beta)^T (\hat{\beta} - \beta)]$$

Therefore we get:

$$MSE_{OLS} = \mathbb{E}[(X^T y - \beta)^T (X^T y - \beta)]$$
$$MSE_{ridge} = \mathbb{E}[(((1 + \lambda) I)^{-1} X^T y - \beta)^T (((1 + \lambda) I)^{-1} X^T y - \beta)]$$

Now for $\lambda = 0$ we get that $MSE_{OLS} = MSE_{ridge}$ and otherwise it is always possible to find a λ, so that we can scale $X^T y$ to make the MSE of the ridge estimator smaller, as the MSE_{OLS} gets bigger, if $X^T y - \beta$ gets bigger.

Task 5

The k-nearest-neighbour representation is defined as:

$$\hat{f}(x_0) = \frac{1}{k} \sum_{i=1}^{N} \mathbf{1}_{x_i \in N_k(x_0)} y_i$$

where $N_k(x_0)$ is the neighborhood of x_0 defined by the k closest points x_i. Therefore the weights are

$$\ell_i(x_0; \mathcal{X}) = \frac{1}{k} \mathbf{1}_{x_i \in N_k(x_0)}$$

and the k-nearest-neighbor regression is a member of this class.

For the linear regression the we can predict f by

$$\hat{f}(x_0) = x_0^T \beta$$

where $\beta = (X^T X)^{-1} X^T y$. Therefore we have

$$\hat{f}(x_0) = \sum_{i=1}^{N} \left(x_0^T (X^T X)^{-1} X^T \right)_i y_i.$$

Hence the weights are

$$\ell_i(x_0; \mathcal{X}) = \left(x_0^T (X^T X)^{-1} X^T \right)_i$$

and the linear regression is part of this class.

Task 6

First we write functions to perform the steps of the precodure.

```
library(caret) # we get knnreg from here
```

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

```
## Warning: Paket 'ggplot2' wurde unter R Version 4.2.3 erstellt
```

```
# function to get y and X
get_xy <- function(p, sigma, N = 500) {
  X <- sapply(1:p, function(x) runif(n = N, min = -1, max = 1))
  epsilon <- rnorm(n = N, sd = sigma)
  Y <- exp(-8 * apply(X^2, MARGIN = 1, FUN = sum)) + epsilon
  df <- data.frame(Y = Y, X = X)
  return(df)
}

# function to get predictions at x0 = 0 for every f hat
get_fx0 <- function(x0 = 0, sigma, data) {
  x0_df <- as.data.frame(matrix(rep(x0, ncol(data)-1), nrow = 1))
  names(x0_df) <- names(data)[-1]
  f <- as.formula(paste0(names(data)[1], "~", paste(names(data)[-1], collapse = "+")))
  # estimate linear model
  l <- lm(data = data, formula = f)
  l_x0 <- predict.lm(l, newdata = x0_df) # its just the intercept what was clear
                                         # but if we specify other x0 the function
                                         # still works
  # estimate knn
  knn_mod <- knnreg(formula = f, data = data, k = 1)
  knn_x0 <- predict(knn_mod, newdata = x0_df)

  # output the fx0 value
  data.frame(linear = l_x0, knn = as.numeric(knn_x0))
}
```

```
  # function to calculate EPE
 epe <- function(fx0, x_0 = 0, p, mu = 0, sigma, f = function(x) exp(-8 * sum(x^2))) {
   x_0 <- rep(x_0, p)
   noise <- rnorm(n = nrow(fx0), mean = mu, sd = sigma)
   epe_lm <- mean((f(x_0) + noise  - fx0$linear)^2)
   epe_knn <- mean((f(x_0) + noise - fx0$knn)^2)
   return(data.frame(epe_linear = epe_lm, epe_knn = epe_knn))
}
```

Now that we have the functions we can iterate over p and σ and estimate in each of the 1000 iterations a linear model as well as a $KNN(1)$.

```
run <- F # only change if you want to do the simulation again

if(run == T) {
# get all combinations of p and sigma
grid <- expand.grid(p = 1:10, sigma = 0:1)

grid_split <- with(grid, split(x = grid, f = list(p, sigma)))

# create 1000 datasets per p-sigma combination

results <- lapply(grid_split, function(g) {
  # generate specific data sets
  spec <- vector(mode = "list", length = 1000)
  for(m in 1:1000) {
    spec[[m]] <- get_xy(p = g$p, sigma = g$sigma)
  }

  # evaluate f hat at x_0 = 0
  fx0 <- do.call("rbind", lapply(spec, function(s) get_fx0(data = s, sigma = g$sigma)))

  epe_run <- epe(fx0 = fx0, x_0 = 0, p = g$p, mu = 0, sigma = g$sigma)

  data.frame(p = g$p, sigma = g$sigma, epe_run)
})


results_bind <- do.call("rbind", results)

saveRDS(results_bind, file = "results_task6.rds")
}
```

We can now inspect the results of the simulation.

```
# make a nice plot
library(ggplot2)
library(tidyr)
```

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

```
library(ggthemes)
```

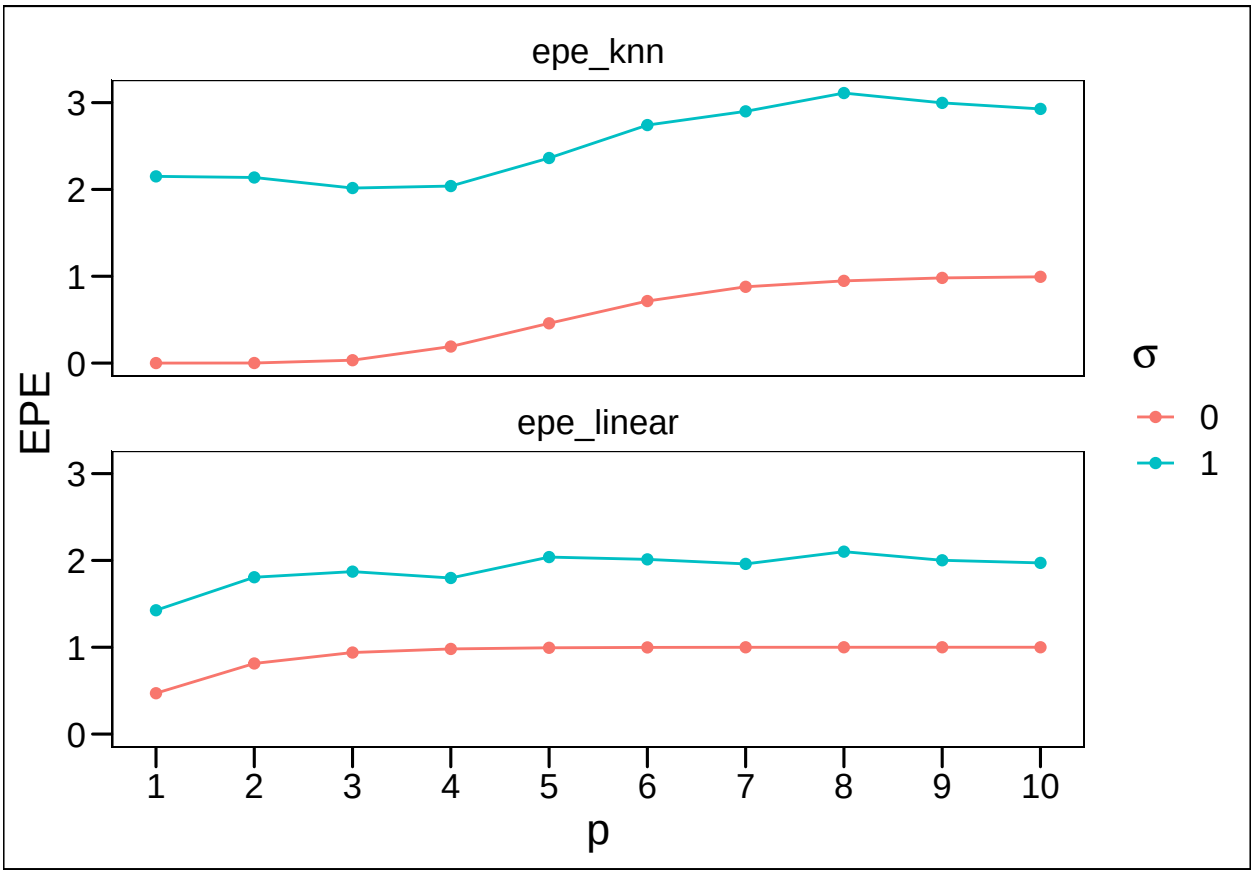
```
## Warning: Paket 'ggthemes' wurde unter R Version 4.2.2 erstellt
```

```
library(latex2exp)
```

```
## Warning: Paket 'latex2exp' wurde unter R Version 4.2.2 erstellt
```

```
# load the saved results
results_bind <- readRDS(file = "results_task6.rds")

results_bind |>
  pivot_longer(cols = contains("epe")) |>
  ggplot(aes(x = p, y = value, color = as.factor(sigma), group = as.factor(sigma))) +
  geom_point() +
  geom_line() +
  scale_x_continuous(breaks = 1:10) +
  facet_wrap(~ name, nrow = 2) +
  labs(y = "EPE", color = TeX("$\\sigma$")) +
  theme_base()
```



There are two observations here. First, dimensionality increases EPE for both methods. However the linear model experiences the increase earlier but then further increases become smaller while for KNN the increases are smaller but last longer with increasing p. Second, an increase in σ shifts up EPE almost by its magnitude. This illustrates the irreducible error that is introduced by the noise term which is usually part of the models we think of when describing data generating processes.

Task 7

```
# load data
data("diabetes", package = "lars")

# matrices can apparently be columns in df..
# good to know
y <- rnorm(100)
x1 = rnorm(100)
x2 = rnorm(100)
x <- cbind(x1, x2)
df <- data.frame(y = y, x = I(x))

# but this complicates things so we break up the
# structure and make a nice df
# function to get rid of Asis
unAsIs <- function(X) {
    if("AsIs" %in% class(X)) {
        class(X) <- class(X)[-match("AsIs", class(X))]
    }
    X
}

# extract y and x
y <- diabetes$y
x <- unAsIs(diabetes$x)

# make a new df with all the data
diabetes_df <- as.data.frame(cbind(y, x))
```

As instructed, we now set a seed and sample row indices from the set of integers running from 1 to the number of observations with equal probability.

```
# set seed
set.seed(12)

# split the data into train and test
# step 1: sample 400 indices
ind <- sample(x = 1:nrow(diabetes_df), size = 400)

# subset the datasets as instructed
train <- diabetes_df[ind, ]
test <- diabetes_df[-ind, ]
```

The reason why random sampling is a good idea is that we are not really familiar with the dataset. Specifically we do not know whether observations were sorted by any of the variables available and if so we do not know at all by which one. Just taking the 400 first observations then would lead the information contained in training and test data to be biased by sorting leading ultimately to sampling bias in our estimations.

With respect to the question about standardized data lets start what happens if we use unstandardized data to be able to highlight why it is sensible to use standardized data instead in the subsequent analysis. Lets say we have a regression of the form $y = X\beta + \epsilon$. If X contains unstandardized data, the unit of measurement is likely to be different between variables, for instance age is measured in years but height is measured in

Table 1: Correlation matrix for y and X

	y	age	sex	bmi	map	tc	ldl	hdl	tch	ltg	glu
y											
age	0.18										
sex	0.07	0.21									
bmi	0.6	0.17	0.09								
map	0.44	0.31	0.26	0.41							
tc	0.22	0.26	0.06	0.25	0.25						
ldl	0.19	0.23	0.16	0.26	0.21	0.9					
hdl	-0.4	-0.09	-0.39	-0.36	-0.21	0.04	-0.2				
tch	0.42	0.22	0.36	0.41	0.29	0.56	0.67	-0.74			
ltg	0.56	0.27	0.16	0.44	0.39	0.52	0.33	-0.4	0.61		
glu	0.37	0.29	0.24	0.38	0.4	0.33	0.28	-0.27	0.41	0.48	

cm. As regression coefficients can be interpreted ceteris paribus as the change of y wrt to a change in X_i but dependent on the unit of measurement. Thus it is not possible to disentangle the strength of the effects based on unstandardized regression coefficients. This is especially a problem if we want to perform variable selection, as we want to select those variables with the greatest influence on the response variable. But as we cannot really measure how large the influence (read: effect) of the variable is based on unstandardized regression, we cannot perform variable selection like this in a sensible way.

Standardized regression coefficients however represent the change y in terms of standard deviations for a one-standard-deviation change in the corresponding standardized X_i. They allow for direct comparison of the relative importance of different variables and help assess the impact of predictors while accounting for differences in scale and units. Thus standardization solves the problem caused by differences in units of measurement. This ultimately allows sensible variable selection.

To analyse the correlation structures we simply calculate a matrix with correlation of all columns in our dataset. The first column contains the correlations of the variables in X with y and the other columns and rows respectively contain the correlations between the columns in X. In general high absolute values of $corr(X_k, y)$ are desirable because this implies high co- or countermovement of the dependent and independent variables. This at least hints at predictive power of X_k, where k is the column index. Contrary, low values for $corr(X_k, X_j)$, $k \neq j$ are desirable as high values would introduce all the problems associated with multicollinearity, most prominently however the variance of the estimates will become inflated. This means nothing else than a loss in precision of estimates. Another huge problem is that multicollinearity is associated with "almost rank deficient" $X'X$ what can lead to problems if we run our model on a computer system.

```
library(kableExtra)
```

```
## Warning: Paket 'kableExtra' wurde unter R Version 4.2.2 erstellt
```

```
# exploration of correlation
correlation_matrix <- round(cor(train), 2)
# eliminate redundancies and make a nice table for the pdf
correlation_matrix[!lower.tri(correlation_matrix)] <- ""
kable(correlation_matrix, booktabs = T, caption = "Correlation matrix for y and X")
```

Looking at Table 1 we can clearly identify variables that seem to have explanatory power w.r.t. y, namely *bmi*, *ltg*, *map*, *tch*, *hdl* and *glue*. Looking at the correlation structure of this subset of the columns in X we find that especially those variables exhibit substantial correlation among themselves. In the light of variable

selection this causes the familiar problems of multicollinearity and specifically in terms of model selection the problem is, that the effect captured by a regression model with respect to a certain variable depends heavily on the inclusion of variables that are highly correlated with this particular variable. This can destabilize variable selection procedures.

```
# use training data to fit the full model
# get model formula from column names
f <- as.formula(paste0("y~", paste(colnames(train)[-1], collapse = "+")))

fit_full <- lm(data = train, formula = f)

# get variables significant at alpha = 0.05
summary_full <- summary(fit_full)
coefficients <- summary_full$coefficients
significant <- which(coefficients[, 4] < 0.05)[-1]

# in sample MSE
MSE_full_in <- mean(fit_full$residuals^2)

# out of sample MSE
pred_full <- predict(fit_full, newdata = test)
MSE_full_out <- mean((test$y - pred_full)^2)

message(paste("In sample MSE is", MSE_full_in, sep = ": "))
```

```
## In sample MSE is: 2876.35849729569
```

```
message(paste("Out of sample MSE is", MSE_full_out, sep = ": "))
```

```
## Out of sample MSE is: 2809.87939232709
```

```
# use the significant variables only
f2 <- as.formula(paste0("y~", paste(colnames(train)[significant], collapse = "+")))

# estimate smaller model
fit_sig <- lm(data = train, formula = f2)

# in sample MSE
MSE_sig_in <- mean(fit_sig$residuals^2)

# out of sample MSE
pred_sig <- predict(fit_sig, newdata = test)
MSE_sig_out <- mean((test$y - pred_sig)^2)

message(paste("In sample MSE is", MSE_sig_in, sep = ": "))
```

```
## In sample MSE is: 3030.1546413137
```

```
message(paste("Out of sample MSE is", MSE_sig_out, sep = ": "))
```

```
## Out of sample MSE is: 3222.23929484251
```

```r
# F-Test
anova(fit_sig, fit_full)
```

```
## Analysis of Variance Table
##
## Model 1: y ~ sex + bmi + map + ltg
## Model 2: y ~ age + sex + bmi + map + tc + ldl + hdl + tch + ltg + glu
##   Res.Df     RSS Df Sum of Sq      F   Pr(>F)
## 1    395 1212062
## 2    389 1150543  6     61518 3.4666 0.002383 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

We find that the additional variables in the full model yield a statistically significant improvement over the model with just the variables being statistically significant.

Task 8

The Akaike criterion is in general given by

$$AIC = 2K - ln(L)$$

where K is the number of regressors and L is the likelihood of the model. Since $2K$ is a penalty term, AIC is to be minimized to find the most appropriate model.

Best Subset Selection

Best subset selection is basically just getting all 2^P possible combinations of explanatory variables available to us and estimating the corresponding regression models. It would be conventient to use the leaps package for this task but there the AIC-criterion is not implemented but only SIC and BIC. So we have to construct the models ourselves, estimate them and calculate AIC.

First we write a function that calculates all possible combinations of regressors and gives back the associated regression formulas. We will not use the leaps and bounds algorithm as the dataset is not to big.

```r
# write function to get formulas
bs_formulas <- function(x = train, dep = "y", intercept_only = T) {
  # extract variables names
  vars <- names(x)
  exps <- vars[vars != dep]

  # get all combinations
 f <- lapply(1:length(exps), function(k) {
    # get combinations for given k
    combinations <- combn(exps, m = k, simplify = F)
    # make it a regression formula
    formulas <- lapply(combinations, function(c) {
      paste(dep, paste(c, collapse = "+"), sep = "~")
    })
    # make it a vector again
    unlist(formulas)
```

```
    })
  # dissolve list again
  output <- unlist(f)

  if(intercept_only == T) output <- c(as.formula(paste0(dep, "~ 1")), output)

  return(output)
}
```

Then we estimate the models

```
# get formulas
formulas <- bs_formulas() # look at defaults set above

# estimate all models
fits <- lapply(formulas, function(f) lm(data = train, formula = f))

# get log likelihoods
LL <- unlist(lapply(fits, function(f) logLik(f)[1]))

# get number of regressors (K)
K <- unlist(lapply(fits, function(fit) length(fit$coefficients)))

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

# get the most appropriate model
best_index <- which.min(AIC)

# display it
message(formulas[[best_index]])
```

```
## y~sex+bmi+map+tc+ldl+ltg
```

Now that we have identified the best model we can assess its in- and out-of-sample performance using MSE again.

```
# get in sample MSE
MSE_BS_in <- mean(fits[[best_index]]$residuals^2)
message(paste("Within sample MSE is for Best Subset Selection:", MSE_BS_in))
```

```
## Within sample MSE is for Best Subset Selection: 2880.81482384062
```

```
# get out of sample MSE
MSE_BS_out <- mean((test$y - predict(fits[[best_index]], newdata = test))^2)
message(paste("Out of sample MSE is for Best Subset Selection:", MSE_BS_out))
```

```
## Out of sample MSE is for Best Subset Selection: 2901.55718360118
```

Finally we conduct an F-Test of the identified model against the full model.

```
anova(fits[[best_index]], fits[[length(fits)]]) # last model is the full one by construction
```

```
## Analysis of Variance Table
##
## Model 1: y ~ sex + bmi + map + tc + ldl + ltg
## Model 2: y ~ age + sex + bmi + map + tc + ldl + hdl + tch + ltg + glu
##   Res.Df     RSS Df Sum of Sq      F Pr(>F)
## 1    393 1152326
## 2    389 1150543  4    1782.5 0.1507 0.9626
```

Based on the p-value of the F-Test we can conclude that adding the remaining variables to the model selected by best subset selection does not lead to a statically significant improvement.

Backward Stepwise

Here we are lucky because the MASS package provides us with a function that does stepwise regression based on AIC.

```
library(MASS)

# conduct backwards stepwise regression
backwards_step <- stepAIC(fit_full, direction = "backward", trace = F)

# compare number of coefficients
length(backwards_step$coefficients)
```

```
## [1] 7
```

```
length(fit_full$coefficients)
```

```
## [1] 11
```

```
# get in sample MSE
MSE_backwards_in <- mean(backwards_step$residuals^2)
message(paste("Within sample MSE is for Backwards Stepwise:", MSE_backwards_in))
```

```
## Within sample MSE is for Backwards Stepwise: 2880.81482384062
```

```
# get out of sample MSE

MSE_backwards_out <- mean((test$y - predict(backwards_step, newdata = test))^2)
message(paste("Out of sample MSE is for Backwards Stepwise:", MSE_backwards_out))
```

```
## Out of sample MSE is for Backwards Stepwise: 2901.55718360118
```

```
anova(backwards_step, fit_full)
```

```
## Analysis of Variance Table
##
## Model 1: y ~ sex + bmi + map + tc + ldl + ltg
## Model 2: y ~ age + sex + bmi + map + tc + ldl + hdl + tch + ltg + glu
##   Res.Df     RSS Df Sum of Sq      F Pr(>F)
## 1    393 1152326
## 2    389 1150543  4    1782.5 0.1507 0.9626
```

The result of the F-Test is the same as for best subset selection as the chosen model is the same again. The interpretation thus is the same as for best subset selection.

Finally all results can be found in Table 2.

```
# prepare the
df_table <- data.frame(variable = names(fit_full$coefficients),
                       full_model = fit_full$coefficients)

best_subset_df <- data.frame(variable = names(fits[[best_index]]$coefficients),
                             best_subset = fits[[best_index]]$coefficients)

backward_df <- data.frame(variable = names(backwards_step$coefficients),
                             backward = backwards_step$coefficients)

# merge tables
merge1 <- merge(df_table, best_subset_df, by = "variable", all.x = T)
df_table_final <- merge(merge1, backward_df, by = "variable", all.x = T)

# add MSE within and out of sample
MSE <- data.frame(variable = c("In Sample MSE",
                               "Out of Sample MSE"),
                  full_model = c(MSE_full_in,
                                 MSE_full_out),
                  best_subset = c(MSE_BS_in,
                                  MSE_BS_out),
                  backward = c(MSE_backwards_in,
                               MSE_backwards_out))

# bind together
df_table_final_mse <- rbind(df_table_final, MSE)

# do some formatting
df_table_final_mse[, -1] <- round(df_table_final_mse[, -1], 2)
df_table_final_mse[is.na(df_table_final_mse)] <- ""

# create table with kable
kable(df_table_final_mse, booktabs = T, caption = "Results for Different Selection Methods.",
      col.names = c("", "Full model", "Best Subset Selection", "Backwards Stepwise"), digits = 2) |>
  pack_rows("Regression Coefficients", 1, 11, hline_after = T, latex_align = "c") |>
  pack_rows("Performance", 12, 13, hline_after = T, latex_align = "c")
```

We see that Best Subset Selection as well as Backwards Stepwise Selection yield the same model and thus consequently the same MSE within and out of sample. The selected models are much more sparse than the full one. We can also see, that only those variables have been selected by the two methods which have large coefficients in absolute terms. This seems sensible in light of the discussion about standardized regression in Task 7. Interestingly however, both within and out of sample MSE are slightly better for the full model.

Table 2: Results for Different Selection Methods.

	Full model	Best Subset Selection	Backwards Stepwise
Regression Coefficients			
(Intercept)	153.13	153.15	153.15
age	-2.90		
bmi	548.13	554.38	554.38
glu	41.54		
hdl	14.33		
ldl	443.66	510.75	510.75
ltg	732.82	778.67	778.67
map	296.95	301.06	301.06
sex	-190.13	-180.49	-180.49
tc	-677.58	-715.46	-715.46
tch	66.48		
Performance			
In Sample MSE	2876.36	2880.81	2880.81
Out of Sample MSE	2809.88	2901.56	2901.56

Task 9

First we load the data.

```
data("Wage", package = "ISLR")
```

It is already in a nice data.frame such that we can directly dive into modeling. We first remove logwage and add the square of age to the data. Then we search for problematic columns by looking for constant variables, i.e. variables which have always the same value. If we find such a variable it is also deleted from the data.frame as it will cause problems when using lm() if the variable at hand is categorical.

```
# exclude logwage
Wage$logwage <- NULL

# add squared age to the data set
Wage$age_sq <- Wage$age^2

# count distinct values for each variable
count_uniq <- lapply(Wage, function(var) length(unique(var)))

# kick if there is a constant and print a message to know which one were  kicked
for(i in 1:length(count_uniq)) {
  if(count_uniq[[i]] == 1) {
    Wage[, names(count_uniq)[i]] <- NULL
    message(names(count_uniq)[i], " was kicked because it is a constant.")
  }
}
```

```
## region was kicked because it is a constant.
```

Then we estimate the full model with the Wage data.frame. We use Helmert-contrasts as we then can set $< HS\ Grad$ as the baseline and can interpret coefficients then successively in the order $< HS\ Grad <$

HS Grad < *Some College* < *College Grad* < *Advanced Degree* in the sense what the difference between the outcome of interest and the one below in the ranking is.

```
# set dependent
dep <- "wage"

# set independents
indep <- names(Wage)[names(Wage) != dep]

# get model formula as string
f_full <- paste0(dep, "~", paste(indep, collapse = "+"))

# fit model with all desired explanatories
fit_full <- lm(data = Wage, formula = f_full, contrasts = list(education = "contr.helmert"))
```

Now that we have the full model we can again use the function for best subset selection written for Task 8. We then use the AIC again to pick the best model

```
# get all formulas
wage_formulas <- bs_formulas(x = Wage, dep = "wage")

# estimate all models
fits_wage <- lapply(wage_formulas, function(f) lm(data = Wage, formula = f))

# get log likelihoods
LL_wage <- unlist(lapply(fits_wage, function(f) logLik(f)[1]))

# get number of regressors (K)
K_wage <- unlist(lapply(fits_wage, function(fit) length(fit$coefficients)))

# get AIC
AIC_wage <- 2 * K_wage - 2 * LL_wage

# get the most appropriate model
best_index_wage <- which.min(AIC_wage)

# display it
wage_formulas[[best_index_wage]]
```

[1] "wage~year+age+maritl+race+education+jobclass+health+health_ins+age_sq"

```
# get the chosen model
wage_bs_fit <- lm(data = Wage, formula = wage_formulas[[best_index_wage]], contrasts = list(education =

# make nice table with stargazer
suppressPackageStartupMessages(library(stargazer))
stargazer(wage_bs_fit, header = F, dep.var.labels = "wage", font.size = "footnotesize")
```

The selected model contains all possible explanatories. As we did not standardize the data, coefficients are in units of measurement. We will not interpret strength of effects but only the signs and significance. According to the model, wage depends positively and statistically significant on *year*, *age*, whether a person is married or not, on the level of *education* (compared to the baseline category every other jump in education gives positive returns),the *joblass* "Information" and very good *health*. Wage depends negatively on whether

Table 3:

	Dependent variable:
	wage
year	1.269***
	(0.305)
age	2.635***
	(0.360)
maritl2. Married	13.611***
	(1.790)
maritl3. Widowed	0.809
	(7.950)
maritl4. Divorced	0.323
	(2.918)
maritl5. Separated	7.414
	(4.849)
race2. Black	−4.682**
	(2.131)
race3. Asian	−2.755
	(2.584)
race4. Other	−5.808
	(5.626)
education1	3.769***
	(1.176)
education2	4.755***
	(0.600)
education3	5.508***
	(0.411)
education4	7.824***
	(0.381)
jobclass2. Information	3.516***
	(1.315)
health2. >=Very Good	6.258***
	(1.411)
health_ins2. No	−16.441***
	(1.403)
age_sq	−0.027***
	(0.004)
Constant	−2,502.209***
	(612.306)
Observations	3,000
R^2	0.349
Adjusted R^2	0.346
Residual Std. Error	33.756 (df = 2982)
F Statistic	94.173*** (df = 17; 2982)
Note:	*p<0.1; **p<0.05; ***p<0.01

a person is black, has no health insurace and on the square of *age*. The latter is a well known phenomenon in labour economics that can be described by means of diminishing returns of work experience (if you think that age is a good proxy for work experience of course!).

Finally, we check whether simply squaring the age variable to exploit diminishing returns of work experience yields different results from using orthogonal polynoms. We just amend the formula for the full model to achieve this.

```
f_full_orth <- sub(x = f_full, pattern = "age_sq", replacement = "poly(age, 2)")
f_full_orth <- sub(x = f_full_orth, pattern = "age\\+", replacement = "")

summary(lm(data = Wage, formula = f_full_orth))
```

```
##
## Call:
## lm(formula = f_full_orth, data = Wage)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -102.229  -18.435   -3.371   13.874  211.474
##
## Coefficients:
##                                Estimate Std. Error t value Pr(>|t|)
## (Intercept)                 -2463.7026   612.2858  -4.024 5.87e-05 ***
## year                            1.2686     0.3052   4.156 3.33e-05 ***
## maritl2. Married               13.6111     1.7896   7.606 3.78e-14 ***
## maritl3. Widowed                0.8094     7.9498   0.102  0.91891
## maritl4. Divorced               0.3235     2.9175   0.111  0.91173
## maritl5. Separated              7.4137     4.8485   1.529  0.12635
## race2. Black                   -4.6825     2.1311  -2.197  0.02808 *
## race3. Asian                   -2.7554     2.5845  -1.066  0.28645
## race4. Other                   -5.8085     5.6258  -1.032  0.30193
## education2. HS Grad             7.5382     2.3526   3.204  0.00137 **
## education3. Some College       18.0335     2.5024   7.207 7.24e-13 ***
## education4. College Grad       30.5568     2.5319  12.069  < 2e-16 ***
## education5. Advanced Degree    53.1502     2.7937  19.025  < 2e-16 ***
## jobclass2. Information          3.5163     1.3145   2.675  0.00752 **
## health2. >=Very Good            6.2577     1.4111   4.435 9.56e-06 ***
## health_ins2. No               -16.4410     1.4025 -11.723  < 2e-16 ***
## poly(age, 2)1                 210.6096    39.4980   5.332 1.04e-07 ***
## poly(age, 2)2                -240.0136    36.0092  -6.665 3.13e-11 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 33.76 on 2982 degrees of freedom
## Multiple R-squared:  0.3493, Adjusted R-squared:  0.3456
## F-statistic: 94.17 on 17 and 2982 DF,  p-value: < 2.2e-16
```

We see that using orthogonal polynoms changes the coefficients of both age terms if you compare the corresponding R output and Table 3 w.r.t. age. This is due to the fact that age^2 is correlated with *age*. Orthogonal polynomials have on the other side have zero correlation due to the very fact that they are orthogonal to each other. This avoids multicollinearity between the two terms completely and therefore leads to more precise estimates.

Task 10

We first implement the algorithm for forward stagewise regression.

```
# implementation of forwardstagewise
forwardstagewise <- function(x, y, tol = sqrt(.Machine$double.eps)) {

  # checks
  check_dim <- length(y) == nrow(x)
  check_num <- is.numeric(y) & is.numeric(x)

  if(check_dim == F) stop("Dimensions of x and y are not compatible.")
  if(check_dim == F) stop("x or y is not a numeric vector (y) or numeric matrix (x).")

  # center x and y
  y <- y - mean(y)
  x <- sweep(x, 2, colMeans(x), '-')

  # make y a n x 1 matrix
  y_m <- as.matrix(y)

  # init a coefficient vector b = 0
  b <- matrix(rep(0, ncol(x)), byrow = T)

  # set the residuals to y initially
  resid <- y_m

  # get absolute correlations between y and x as starting point
  res_cor <- abs(cor(resid, x))

  # initialize empty output matrix
  output <- matrix(b, ncol = nrow(b))

  count_iter <- 1
  repeat {
    # print(count_iter)
    # identify col index of variable with highest correlation
    index_high <- which.max(res_cor)

    # regress residuals on chosen variable
    b_add <- sum(x[, index_high] * resid) / sum(x[, index_high]^2)

    # add coefficient to b at the appropriate index
    b[index_high, ] <- b[index_high,] + b_add

    output <- rbind(output, matrix(b, ncol = nrow(b)))

    # update residuals
    resid <- y_m - x %*% b

    # update absolute residual correlations
    res_cor <- abs(cor(resid, x))

    # now check whether we stop
```

```
    if(max(res_cor) < tol) break
    count_iter <- count_iter + 1
  }

  # return output
  return(output)
}
```

Then we conduct the simulation as described and calculate the expected sum of squares between the true coefficients and the coefficients resulting from forward stagewise regression and best subset selection respectively. We then visualize the distribution of the sum of squares over the 50 simulation steps to see whether one methods systematically outperforms the other.

```
library(MASS)
library(leaps)
```

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

```
# specify params of multivariate normal
mu <- rep(0, 31) # as standard normal has mu = 0
sigma <- diag(x = 1, nrow = 31) # sigma is 1 in standard normal
sigma[!diag(x = T, nrow = 31)] <- 0.85 # add covariances to off diags

# Conduct the simulation
b_list <- vector(mode = "list", length = 50)
for(i in 1:50) {
   message(paste("Simulation Step:", i))
  x <- mvrnorm(n = 300, mu = mu, Sigma = sigma)
  colnames(x) <- 1:31 # name by col index

  # sample column indices to select randomly which of the 31 variables define y
  col_ind <- sort(sample(1:31, size = 10))

  # draw coefficient from N(0, 0.4)
  random_coef <- rnorm(n = 10, mean = 0, sd = sqrt(0.4)) # assuming that you give the variance in the i

  # make true coefficient vector
  true_b <- matrix(rep(0, 31), ncol = 1)
  true_b[col_ind,] <- random_coef

  # get noise
  noise <- rnorm(300, mean = 0, sd = sqrt(6.25))

  # get y
  y <- x %*% true_b + noise

  # center x and y
  x_c <- sweep(x, 2, colMeans(x), "-")
  y_c <- y - mean(y)

  # calculate best subset
  best_subset <- regsubsets(x = x_c, y = y_c, nvmax = 31)
```

```
  # choose the best overall model via minimal BIC
  best_model_bs <- which.min(summary(best_subset)$bic)
  best_bs_coef <- coef(best_subset, best_model_bs)[-1] # remove intercept as 0 anyway

  # get the appropriate model according to forwardstagewise
  evolution_fsw <- forwardstagewise(x = x, y = y)
  model_fsw <- evolution_fsw[nrow(evolution_fsw), ]

  # make all coef objects vectors
  true_b_v <- true_b[, 1]
  best_bs_v <- rep(0, 31)
  best_bs_v[as.integer(names(best_bs_coef))] <- best_bs_coef
  fsw_v <- model_fsw

  b_list[[i]] <- cbind(true_b_v, best_bs_v, fsw_v)
}
```

```
## Simulation Step: 1

## Simulation Step: 2

## Simulation Step: 3

## Simulation Step: 4

## Simulation Step: 5

## Simulation Step: 6

## Simulation Step: 7

## Simulation Step: 8

## Simulation Step: 9

## Simulation Step: 10

## Simulation Step: 11

## Simulation Step: 12

## Simulation Step: 13

## Simulation Step: 14

## Simulation Step: 15

## Simulation Step: 16
```

```
## Simulation Step: 17

## Simulation Step: 18

## Simulation Step: 19

## Simulation Step: 20

## Simulation Step: 21

## Simulation Step: 22

## Simulation Step: 23

## Simulation Step: 24

## Simulation Step: 25

## Simulation Step: 26

## Simulation Step: 27

## Simulation Step: 28

## Simulation Step: 29

## Simulation Step: 30

## Simulation Step: 31

## Simulation Step: 32

## Simulation Step: 33

## Simulation Step: 34

## Simulation Step: 35

## Simulation Step: 36

## Simulation Step: 37

## Simulation Step: 38

## Simulation Step: 39

## Simulation Step: 40
```

```
## Simulation Step: 41

## Simulation Step: 42

## Simulation Step: 43

## Simulation Step: 44

## Simulation Step: 45

## Simulation Step: 46

## Simulation Step: 47

## Simulation Step: 48

## Simulation Step: 49

## Simulation Step: 50
```

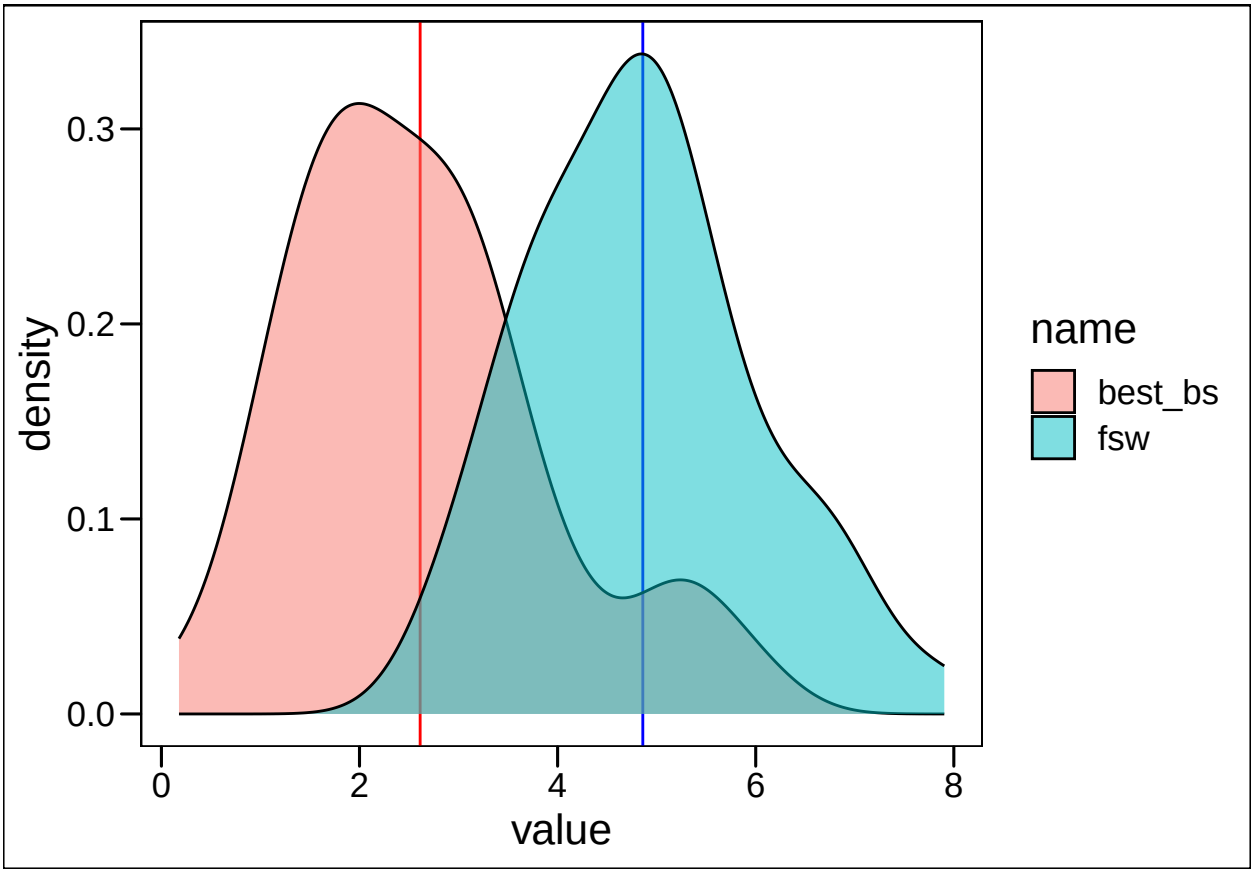
```r
# calculate sum of squares for each iteration
sum_squares_list <- lapply(b_list, function(b) {
  best_bs <- sum((b[,1] - b[,2])^2)
  fsw <- sum((b[,1] - b[,3])^2)
  c(best_bs = best_bs, fsw = fsw)
})

sum_squares <- as.data.frame(do.call("rbind", sum_squares_list))

# calculate expectation
expect <- apply(sum_squares, MARGIN = 2, mean)
expect
```

```
##  best_bs      fsw
## 2.612040 4.860062
```

```r
# visualize sum of squares from simulation steps with histograms
sum_squares |>
  pivot_longer(cols = everything()) |>
  ggplot(aes(x = value, fill = name)) +
  geom_vline(xintercept = expect[1], color = "red") +
  geom_vline(xintercept = expect[2], color = "blue") +
  geom_density(alpha = 0.5) +
  theme_base()
```



We can see that forward stagewise regression has notably higher sum of squares between true coefficients and the ones determined via forward stagewise regression compared to best subset selection. Also when looking at the visualisation we can observe that the distribution of the sum of squares is shifted to the right for foreward stagewise regression compared to best subset selection.