Assignment 4

Group 2

2023-10-30

Task 1

A binary dependent variable is generated by:

$$Pr(Y = 1|X) = q + (1 - 2q) \cdot \mathbb{1}\left[\sum_{j=1}^{J} X_j > \frac{J}{2}\right]$$

where $1[]$ is the indicator function$X \sim U(0,1)^p$, $0 \leq q \leq \frac{1}{2}$, $J \leq p$ is some predefined (even) number. Describe the probability surface and give the Bayes error rate.

Solution

The sum of n independent random variables $X_j \sim U(0,1)$ is Irwin-Hall Distributed, with cdf:

$$F_X = \frac{1}{n!}\sum_{k=0}^{|x|}(-1)^k \binom{n}{k}(x-k)^{n-1}$$

with support $x \in [0, n]$. Also, $E[X] = \frac{n}{2}$. Given the median being also $\frac{n}{2}$, we know the distribution function to be symmetric, further implying that we have 50% probability to get 1 out of the indicator function, and 50% to get a 0, no matter the J.

From Chapter 2 we have the Bayes Error defined as:

$$Err_{Bayes} = 1 - E\left[max_{j\in(0,1)}\ P(Y = j|X)\right]$$

We also have that:

$$max_{j\in(0,1)}\ P(Y = j|X) = \begin{cases} P(Y = 1|X) \ \ if \ \ P(Y = 1|X) > 0.5 \\ \\ 1 - P(Y = 1|X) \ \ if \ \ P(Y = 1|X) \leq 0.5 \end{cases}$$

Since the $\mathbb{1}[]$ takes values of either 0 or 1 50% of the times and $0 \leq q \leq 0.5$, we have that:

$$max_{j\in(0,1)}\ P(Y = j|X) = \begin{cases} P(Y = 1|X) = 1 - q \ \ if \ \ \mathbb{1}[] = 1 \\ \\ P(Y = 0|X) = 1 - q \ \ if \ \ \mathbb{1}[] = 0 \end{cases}$$

Clearly, we have that $max_{j\in(0,1)}\ P(Y = j|X) = 1 - q$ and therefore:

$$Err_{Bayes} = 1 - E\left[max_{j\in(0,1)}\ P(Y = j|X)\right] = q$$

Task 2

Without loss of generality, we can look at the case where $\mu = 0$,$\sigma^2 = 1$:

Let $x_i^{(k)}$ be a sample i from the k th bootstrap.

then $\overline{x}_k^* = \frac{1}{n}\sum_i x_i^{(k)}$

Key values we need to compute are:

$$Ex_i^k = E(E[x_i^{(k)}|x]) = E\overline{x} = 0$$

$$var(x_i^{(k)}) = E(var(x_i^{(k)}|x)) + var(E[x_i^{(k)}|x]) = \frac{n-1}{n} + \frac{1}{n} = 1$$

$$cov(x_i^{(1)}, x_j^{(2)}) = E(x_i^{(1)} - 0)(x_j^{(2)} - 0) = E(E[x_i^{(1)}, x_j^{(2)}|x]) = E\overline{x}^2 = \frac{1}{n}$$

the same is true for the $(x_i^{(k)}, x_j^{(k)})$ where $i \neq j$

With above the calculation is going like this:

$$cov(\overline{x}_1^*, \overline{x}_2^*) = \frac{1}{n^2}(\sum_{i,j} cov(x_i^{(1)}, x_j^{(2)})) = \frac{1}{n}$$

$$var(\overline{x}_i^*) = \frac{1}{n^2}(x * var(x_1^{(1)}) + n(n-1) * cov(x_1^{(1)}, x_2^{(1)})) = \frac{2n-1}{n^2}$$

$$\rightarrow corr(\overline{x}_1^*, \overline{x}_2^*) = \frac{n}{2n-1}$$

We have already derived $\operatorname{Var}(\bar{x}_1^*)$ above. For $\bar{x}_{\text{bag}}$, assume we have B realizations, then

$$\begin{aligned}
\operatorname{Var}(\bar{x}_{\text{bag}}) &= \operatorname{Var}\left(\frac{1}{B}\sum_{i=1}^{B}\bar{x}_i^*\right) \\
&= \frac{1}{B^2}\sum_{i=1}^{B}\operatorname{Var}(\bar{x}_i^*) + \frac{1}{B^2}\sum_{j\neq k}^{B}\operatorname{Cov}(\bar{x}_j^*, \bar{x}_k^*) \\
&= \frac{1}{B} * \frac{(N-1)\sigma^2}{N^2} + \frac{B-1}{B} * \frac{\sigma^2}{N} \\
&= \frac{(2N-1)+(B-1)N}{BN^2} * \sigma^2
\end{aligned}$$

Task 3

Suppose we fit a linear regression model to N observations with response y_i and predictors $x_{i1}, ..., x_{ip}$. Assume that all variables are standardized such that for example for y it holds $y^T 1 = 0$ and $\frac{1}{N}y^T y = 1$. Let RSS be the mean-squared residuals on the training data, and $\hat{\beta}$ the estimated OLS coefficient. Denote by $RSS_j^\star$ the mean-squared residuals on the training data using the same $\hat{\beta}$, but with the N values for the j-th variable randomly permuted before the predictions are calculated. Show that

$$E_P\left[RSS_j^\star - RSS\right] = 2\hat{\beta}_j^2,$$

where E_P denotes expectation with respect to the permutation distribution.

The residual sum of is given by:

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

For $RSS_j^\star$ we use X_j instead of X, where X_j is X but with the j-th variable randomly permuted. Therefore if we multiply it out and then factor out we get that:

$$E_P[RSS_j^\star - RSS] = E_P[2y^T(X - X_j)\hat{\beta} + \hat{\beta}^T(X_j^T X_j - X^T X)\hat{\beta}] = E_P[2y^T(X - X_j)\hat{\beta}] + E_P[\hat{\beta}^T(X_j^T X_j - X^T X)\hat{\beta}]$$

Because X_j and X only differ in the j-th column, $x_j^\star$ and x_j respectively, it holds that:

$$2y^T(X - X_j)\hat{\beta} = 2\hat{\beta}_j y^T(x_j - x_j^\star)$$

If we assume that $X^T X = 1$ we can see that $E_P[x_j^\star] = \bar{x_j} = 0$ (zero vector). Therefore we get:

$$E_P[2\hat{\beta}_j y^T(x_j - x_j^\star)] = 2\hat{\beta}_j y^T x_j$$

Now the OLS esimator $\hat{\beta}$ is

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

which leads to

$$2\hat{\beta}_j y^T x_j = 2\hat{\beta}_j \hat{\beta}_j = 2\hat{\beta}_j^2$$

Also because $X^T X = 1$ we have that

$$E_P[\hat{\beta}^T(X_j^T X_j - X^T X)\hat{\beta}] = 0.$$

Overall we therefore have:

$$E_P[RSS_j^\star - RSS] = 2\hat{\beta}_j^2$$

Task 4

a) part

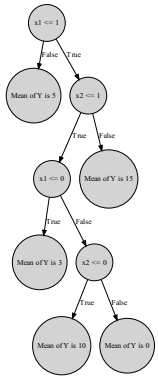
Sketch the tree corresponding to the partition of the predictor space illustrated on the left of the figure. The numbers inside the boxes indicate the mean of Y within each region.

```
# Install and load the necessary packages
if (!requireNamespace("DiagrammeR", quietly = TRUE)) {
  install.packages("DiagrammeR")
}

library(DiagrammeR)
```

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

```
# Create a decision tree structure with circles of the same size and True/False labels
decision_tree <- "
digraph DecisionTree {
  node [shape=circle, style=filled, fillcolor=lightgray, width=1, height=1];
  root [label=\"x1 <= 1\"];
  decision1 [label=\"Mean of Y is 5\"];
  decision2 [label=\"x2 <= 1\"];
  leaf3 [label=\"x1 <= 0\"];
```

```
  leaf4 [label=\"Mean of Y is 15\"];
  decision3 [label=\"Mean of Y is 3\"];
  decision4 [label=\"x2 <= 0\"];
  leaf5 [label=\"Mean of Y is 10\"];
  leaf6 [label=\"Mean of Y is 0\"];
  root -> decision1 [label=\"False\"];
  root -> decision2 [label=\"True\"];

  decision2 -> leaf3 [label=\"True\"];
  decision2 -> leaf4 [label=\"False\"];

  leaf3 -> decision3 [label=\"True\"];
  leaf3 -> decision4 [label=\"False\"];

  decision4 -> leaf5 [label=\"True\"];
  decision4 -> leaf6 [label=\"False\"];
}
"

# Create a graph from the decision tree structure
tree_graph <- grViz(decision_tree)

# Display the decision tree
tree_graph
```

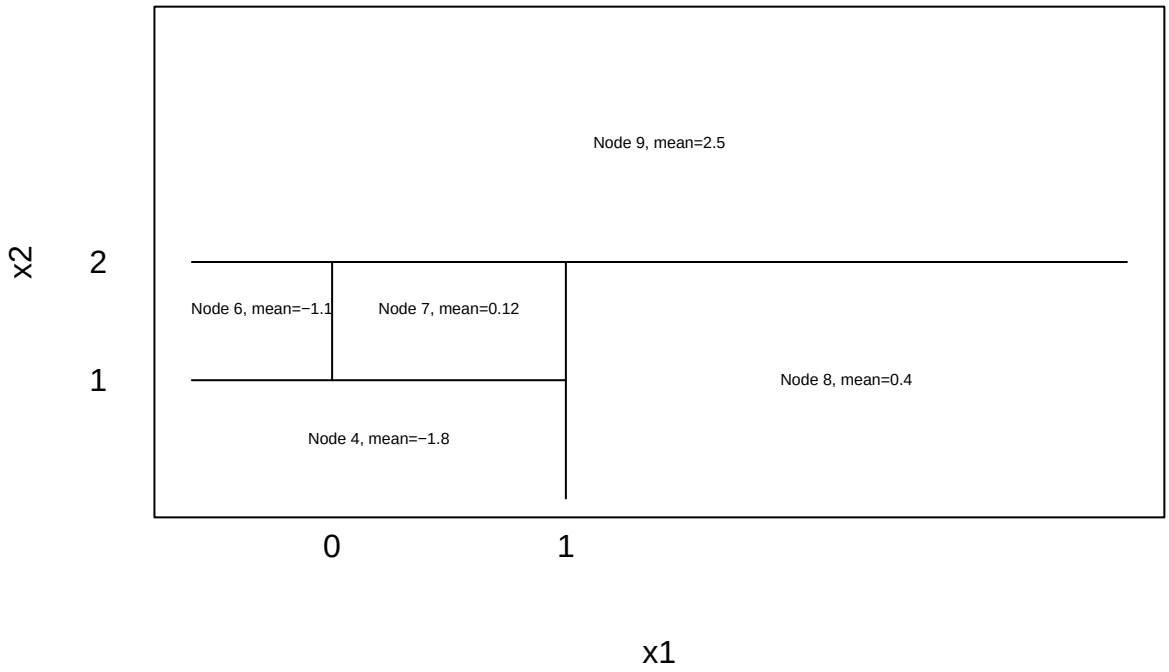


b) part

Create a diagram similar to the plot on the left in the figure, using the tree illustrated on the right of the same figure. You should divide up the predictor space into the correct regions, and indicate the mean for

each region. Determine also the fitted function.

```
par(xpd = NA)
plot(NA, NA, type = "n", xlim = c(0, 1), ylim = c(0, 1), xlab = "x1", ylab = "x2", xaxt = "n", yaxt = "
lines(x = c(40,40)/100, y = c(0,50)/100)
lines(x = c(0,100)/100, y = c(50,50)/100)
lines(x= c(0,40)/100,y=c(25,25)/100)
lines(x=c(15,15)/100,y=c(25,50)/100)

text(x = 40/100, y = -0.1, labels = c("1"), col = "black")
text(x = 15/100, y = -0.1, labels = c("0"), col = "black")
text(x = -0.1, y = 25/100, labels = c("1"), col = "black")
text(x = -0.1, y = 50/100, labels = c("2"), col = "black")


text(x = 50/100, y = 75/100, labels = c("Node 9, mean=2.5"),cex=0.5)
text(x = 7.5/100, y = 40/100, labels = c("Node 6, mean=-1.1"),cex=0.5)
text(x = 27.5/100, y = 40/100, labels = c("Node 7, mean=0.12"),cex=0.5)
text(x = 70/100, y = 25/100, labels = c("Node 8, mean=0.4"),cex=0.5)
text(x = 20/100, y = 12.5/100, labels = c("Node 4, mean=-1.8"),cex=0.5)
```



Provides the mean value of y for a specific region.

```
f_function <- function(x1, x2) {
  sapply(1:length(x1), function(i) {
    if (x2[i] < 2) {
```

```
      if (x1[i] < 1) {
        if (x2[i] < 1) {
          -1.8
        } else {
          if (x1[i] < 0.003) {
            -1.1
          } else {
            0.12
          }
        }
      } else {
        0.4
      }
    } else {
      2.5
    }
  })
}

result <- f_function(c(-1, 0, 1, 2), c(0, 1, 2, 3))
```

Task 5

```
suppressMessages(library("ISLR"))
```

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

```
suppressMessages(library("rpart"))
suppressMessages(library("tree"))
```

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

```
suppressMessages(library("dplyr"))
suppressMessages(library("rpart.plot"))
```

```
## Warning: Paket 'rpart.plot' wurde unter R Version 4.2.3 erstellt
```

```
suppressMessages(library("knitr"))
```

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

```
data("Carseats", package="ISLR")
df=Carseats
```

a) Split the data set into a training set and a test set. – (70% train-30% test)

```r
set.seed(123)
df$id = 1:nrow(df)
train = df %>% dplyr::sample_frac(0.70)
test = dplyr::anti_join(df, train, by = 'id')
train = train[-c(12)]
test = test[-c(12)]
```

b) Fit a regression tree to the training set. Plot the tree, and interpret the results. What test MSE do you obtain?

```r
set.seed(123)
tree = rpart(Sales ~ ., data = train)
summary(tree)
```

```
## Call:
## rpart(formula = Sales ~ ., data = train)
##   n= 280
##
##            CP nsplit rel error    xerror       xstd
## 1  0.25462228      0 1.0000000 1.0091773 0.08510095
## 2  0.09215223      1 0.7453777 0.7573932 0.05919359
## 3  0.07090167      2 0.6532255 0.7416373 0.05767966
## 4  0.04324517      3 0.5823238 0.6393493 0.04881898
## 5  0.03604927      4 0.5390787 0.6672966 0.05248835
## 6  0.03227129      5 0.5030294 0.6551342 0.05261532
## 7  0.02428572      7 0.4384868 0.5957057 0.04893045
## 8  0.01748177      8 0.4142011 0.5958479 0.05110351
## 9  0.01591543      9 0.3967193 0.6162069 0.05494734
## 10 0.01562578     10 0.3808039 0.6195996 0.05473044
## 11 0.01413317     11 0.3651781 0.6135908 0.05355575
## 12 0.01354372     12 0.3510449 0.6083060 0.05298598
## 13 0.01265304     14 0.3239575 0.6031406 0.05311808
## 14 0.01046095     15 0.3113044 0.6012050 0.05337364
## 15 0.01000000     16 0.3008435 0.6055665 0.05434016
##
## Variable importance
##   ShelveLoc       Price   CompPrice         Age  Population Advertising
##          35          31          17           5           4           3
##   Education      Income
##           3           3
##
## Node number 1: 280 observations,    complexity param=0.2546223
##   mean=7.437786, MSE=7.954364
##   left son=2 (222 obs) right son=3 (58 obs)
##   Primary splits:
##       ShelveLoc   splits as  LRL,       improve=0.25462230, (0 missing)
##       Price       < 105.5 to the right, improve=0.13694340, (0 missing)
##       Age         < 65.5  to the right, improve=0.10010780, (0 missing)
##       Advertising < 7.5   to the left,  improve=0.06180591, (0 missing)
##       Income      < 61.5  to the left,  improve=0.03311595, (0 missing)
##
```

```
## Node number 2: 222 observations,    complexity param=0.09215223
##   mean=6.71036, MSE=5.627182
##   left son=4 (150 obs) right son=5 (72 obs)
##   Primary splits:
##       Price       < 105.5 to the right, improve=0.16429540, (0 missing)
##       ShelveLoc   splits as  L-R,       improve=0.11604670, (0 missing)
##       Age         < 68.5  to the right, improve=0.09259253, (0 missing)
##       Advertising < 7.5   to the left,  improve=0.08661964, (0 missing)
##       Income      < 61.5  to the left,  improve=0.07886769, (0 missing)
##   Surrogate splits:
##       CompPrice  < 109.5 to the right, agree=0.761, adj=0.264, (0 split)
##       Population < 507.5 to the left,  agree=0.685, adj=0.028, (0 split)
##       Income     < 22.5  to the right, agree=0.680, adj=0.014, (0 split)
##
## Node number 3: 58 observations,    complexity param=0.07090167
##   mean=10.22207, MSE=7.084261
##   left son=6 (38 obs) right son=7 (20 obs)
##   Primary splits:
##       Price       < 109.5 to the right, improve=0.38432390, (0 missing)
##       Age         < 61.5  to the right, improve=0.15967180, (0 missing)
##       Education   < 11.5  to the right, improve=0.11849500, (0 missing)
##       Advertising < 13.5  to the left,  improve=0.11063440, (0 missing)
##       CompPrice   < 131.5 to the left,  improve=0.08607235, (0 missing)
##   Surrogate splits:
##       Population  < 92.5  to the right, agree=0.741, adj=0.25, (0 split)
##       Education   < 11.5  to the right, agree=0.707, adj=0.15, (0 split)
##       CompPrice   < 102   to the right, agree=0.672, adj=0.05, (0 split)
##       Advertising < 15.5  to the left,  agree=0.672, adj=0.05, (0 split)
##
## Node number 4: 150 observations,    complexity param=0.04324517
##   mean=6.0442, MSE=4.453584
##   left son=8 (44 obs) right son=9 (106 obs)
##   Primary splits:
##       ShelveLoc   splits as  L-R,       improve=0.14417840, (0 missing)
##       CompPrice   < 124.5 to the left,  improve=0.11790140, (0 missing)
##       Advertising < 7.5   to the left,  improve=0.10645280, (0 missing)
##       Age         < 65.5  to the right, improve=0.08327840, (0 missing)
##       Income      < 61.5  to the left,  improve=0.08264313, (0 missing)
##   Surrogate splits:
##       Population < 15    to the left,  agree=0.720, adj=0.045, (0 split)
##       Age        < 28.5  to the left,  agree=0.720, adj=0.045, (0 split)
##       Price      < 162.5 to the right, agree=0.713, adj=0.023, (0 split)
##
## Node number 5: 72 observations,    complexity param=0.03227129
##   mean=8.098194, MSE=5.221573
##   left son=10 (51 obs) right son=11 (21 obs)
##   Primary splits:
##       CompPrice  < 123.5 to the left,  improve=0.19013200, (0 missing)
##       Age        < 54.5  to the right, improve=0.18899550, (0 missing)
##       ShelveLoc  splits as  L-R,       improve=0.16987950, (0 missing)
##       Price      < 88    to the right, improve=0.09985730, (0 missing)
##       Population < 162   to the right, improve=0.08620326, (0 missing)
##   Surrogate splits:
##       Price      < 103.5 to the left,  agree=0.750, adj=0.143, (0 split)
```

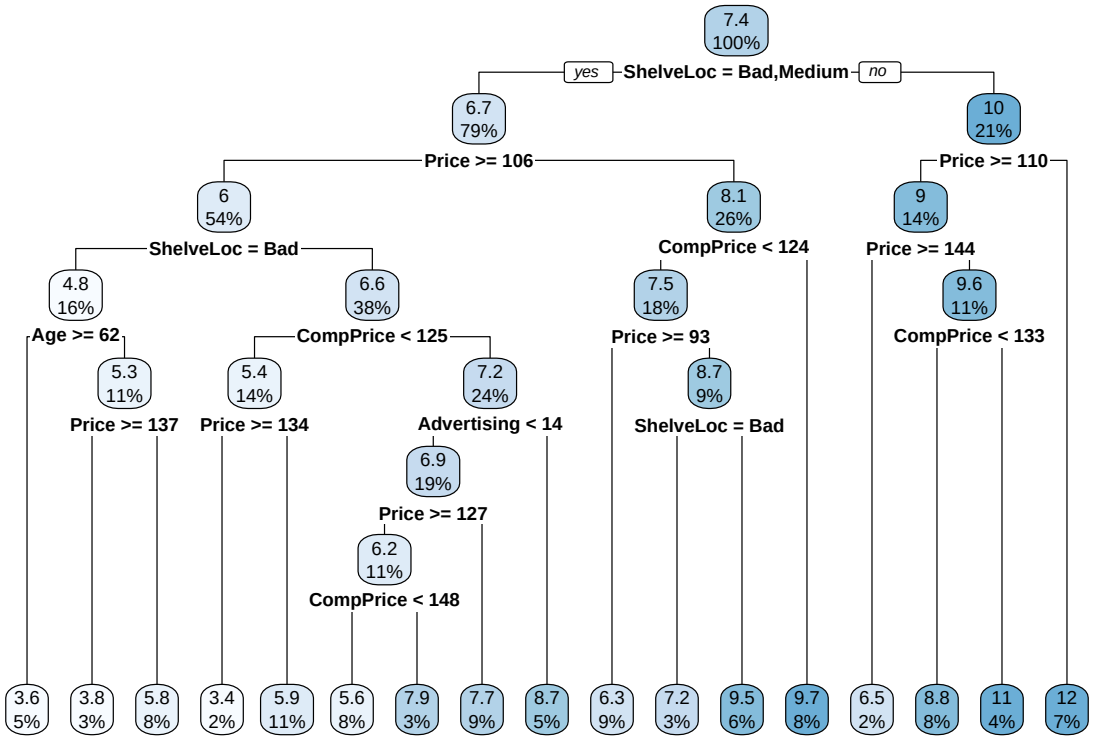
```
##       Income     < 34.5  to the right, agree=0.722, adj=0.048, (0 split)
##       Population < 494   to the left,  agree=0.722, adj=0.048, (0 split)
##
## Node number 6: 38 observations,    complexity param=0.02428572
##   mean=9.025, MSE=4.931751
##   left son=12 (7 obs) right son=13 (31 obs)
##   Primary splits:
##       Price       < 144   to the right, improve=0.2886222, (0 missing)
##       Age         < 63.5  to the right, improve=0.1833129, (0 missing)
##       US          splits as  LR,        improve=0.1796395, (0 missing)
##       Advertising < 0.5   to the left,  improve=0.1796395, (0 missing)
##       CompPrice   < 121.5 to the left,  improve=0.1434922, (0 missing)
##   Surrogate splits:
##       Income < 104.5 to the right, agree=0.842, adj=0.143, (0 split)
##
## Node number 7: 20 observations
##   mean=12.4965, MSE=3.278343
##
## Node number 8: 44 observations,    complexity param=0.01265304
##   mean=4.800455, MSE=3.601359
##   left son=16 (13 obs) right son=17 (31 obs)
##   Primary splits:
##       Age        < 61.5  to the right, improve=0.17784400, (0 missing)
##       CompPrice  < 144   to the left,  improve=0.16485990, (0 missing)
##       Population < 283   to the left,  improve=0.11292850, (0 missing)
##       Price      < 132.5 to the right, improve=0.11036090, (0 missing)
##       Income     < 101   to the left,  improve=0.09771273, (0 missing)
##   Surrogate splits:
##       CompPrice < 124.5 to the left,  agree=0.773, adj=0.231, (0 split)
##       Income    < 33.5  to the left,  agree=0.727, adj=0.077, (0 split)
##       Price     < 119   to the left,  agree=0.727, adj=0.077, (0 split)
##
## Node number 9: 106 observations,    complexity param=0.03604927
##   mean=6.560472, MSE=3.898691
##   left son=18 (39 obs) right son=19 (67 obs)
##   Primary splits:
##       CompPrice   < 124.5 to the left,  improve=0.19428320, (0 missing)
##       Income      < 61.5  to the left,  improve=0.15169500, (0 missing)
##       Advertising < 6.5   to the left,  improve=0.12475180, (0 missing)
##       Age         < 49.5  to the right, improve=0.12023020, (0 missing)
##       Price       < 135.5 to the right, improve=0.07821028, (0 missing)
##   Surrogate splits:
##       Price      < 111.5 to the left,  agree=0.736, adj=0.282, (0 split)
##       Population < 499.5 to the right, agree=0.651, adj=0.051, (0 split)
##       Income     < 29.5  to the left,  agree=0.642, adj=0.026, (0 split)
##       Age        < 78.5  to the right, agree=0.642, adj=0.026, (0 split)
##
## Node number 10: 51 observations,    complexity param=0.03227129
##   mean=7.458824, MSE=4.963912
##   left son=20 (26 obs) right son=21 (25 obs)
##   Primary splits:
##       Price     < 92.5  to the right, improve=0.2854717, (0 missing)
##       Income    < 100.5 to the left,  improve=0.2085942, (0 missing)
##       ShelveLoc splits as  L-R,       improve=0.1890332, (0 missing)
```

```
##       Age       < 35.5  to the right, improve=0.1583237, (0 missing)
##       Education < 11.5  to the left,  improve=0.1570051, (0 missing)
##   Surrogate splits:
##       CompPrice  < 99.5  to the right, agree=0.667, adj=0.32, (0 split)
##       Education  < 13.5  to the left,  agree=0.667, adj=0.32, (0 split)
##       Population < 271   to the right, agree=0.647, adj=0.28, (0 split)
##       Age        < 49    to the right, agree=0.627, adj=0.24, (0 split)
##       Income     < 50.5  to the left,  agree=0.588, adj=0.16, (0 split)
##
## Node number 11: 21 observations
##   mean=9.650952, MSE=2.443475
##
## Node number 12: 7 observations
##   mean=6.514286, MSE=2.881396
##
## Node number 13: 31 observations,    complexity param=0.01748177
##   mean=9.591935, MSE=3.649906
##   left son=26 (21 obs) right son=27 (10 obs)
##   Primary splits:
##       CompPrice   < 132.5 to the left,  improve=0.3441166, (0 missing)
##       Age         < 61.5  to the right, improve=0.1886891, (0 missing)
##       Advertising < 12.5  to the left,  improve=0.1215375, (0 missing)
##       US          splits as  LR,        improve=0.1184813, (0 missing)
##       Income      < 41.5  to the left,  improve=0.1115535, (0 missing)
##   Surrogate splits:
##       Price < 138   to the left,  agree=0.742, adj=0.2, (0 split)
##
## Node number 16: 13 observations
##   mean=3.564615, MSE=1.418609
##
## Node number 17: 31 observations,    complexity param=0.01046095
##   mean=5.31871, MSE=3.607637
##   left son=34 (8 obs) right son=35 (23 obs)
##   Primary splits:
##       Price       < 136.5 to the right, improve=0.2083292, (0 missing)
##       Population  < 283   to the left,  improve=0.1647044, (0 missing)
##       CompPrice   < 144   to the left,  improve=0.1201417, (0 missing)
##       Advertising < 8.5   to the left,  improve=0.1047802, (0 missing)
##       Income      < 87    to the left,  improve=0.1020453, (0 missing)
##   Surrogate splits:
##       Age       < 27.5  to the left,  agree=0.871, adj=0.500, (0 split)
##       Education < 11.5  to the left,  agree=0.774, adj=0.125, (0 split)
##
## Node number 18: 39 observations,    complexity param=0.01562578
##   mean=5.419744, MSE=3.270602
##   left son=36 (7 obs) right son=37 (32 obs)
##   Primary splits:
##       Price       < 133.5 to the right, improve=0.2728430, (0 missing)
##       Advertising < 6     to the left,  improve=0.2590152, (0 missing)
##       Income      < 83.5  to the left,  improve=0.1940935, (0 missing)
##       US          splits as  LR,        improve=0.1728556, (0 missing)
##       Age         < 68    to the right, improve=0.1210741, (0 missing)
##
## Node number 19: 67 observations,    complexity param=0.01591543
```

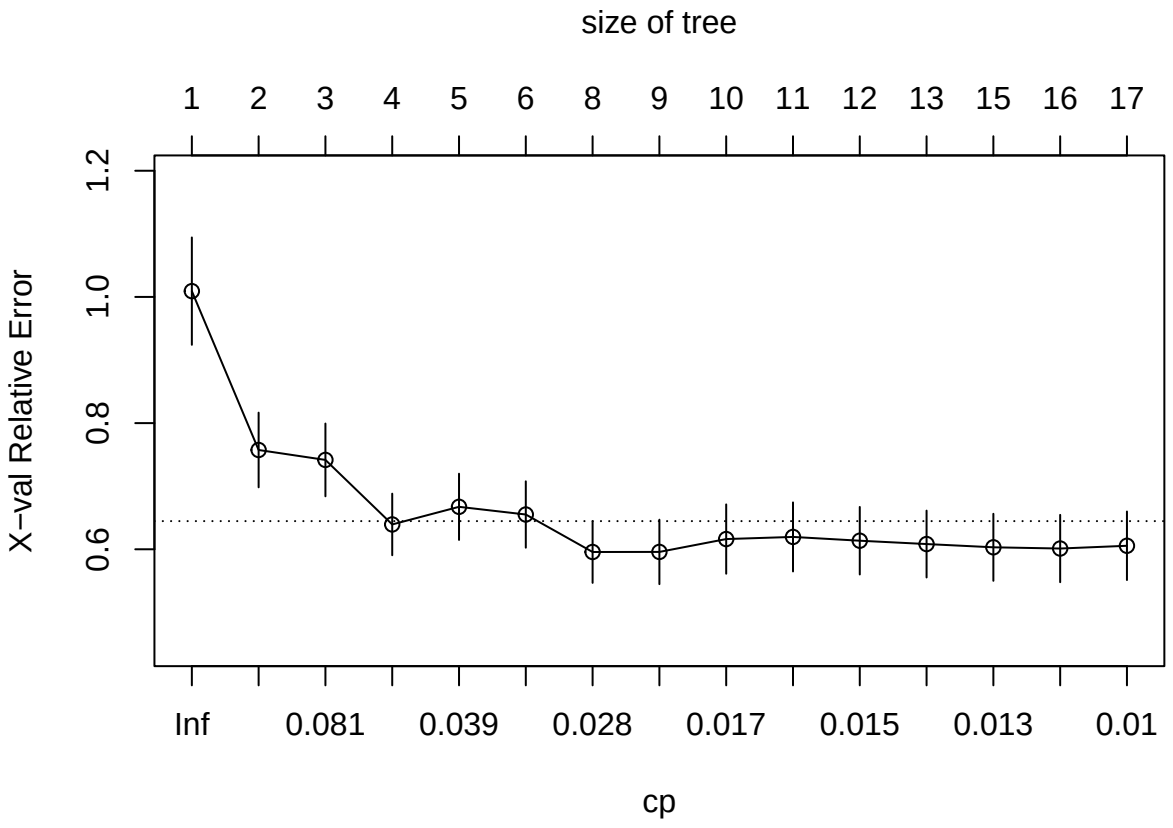
```
##   mean=7.224478, MSE=3.065941
##   left son=38 (54 obs) right son=39 (13 obs)
##   Primary splits:
##       Advertising < 13.5  to the left,  improve=0.1725612, (0 missing)
##       Price       < 127   to the right, improve=0.1665229, (0 missing)
##       Income      < 57.5  to the left,  improve=0.1333498, (0 missing)
##       Age         < 54.5  to the right, improve=0.1172588, (0 missing)
##       Education   < 16.5  to the right, improve=0.1134184, (0 missing)
##   Surrogate splits:
##       CompPrice < 127.5 to the right, agree=0.821, adj=0.077, (0 split)
##
## Node number 20: 26 observations
##   mean=6.291538, MSE=3.624328
##
## Node number 21: 25 observations,    complexity param=0.01413317
##   mean=8.6728, MSE=3.466284
##   left son=42 (9 obs) right son=43 (16 obs)
##   Primary splits:
##       ShelveLoc   splits as  L-R,       improve=0.36324440, (0 missing)
##       Price       < 75.5  to the right, improve=0.17532440, (0 missing)
##       Income      < 62    to the left,  improve=0.15169510, (0 missing)
##       Population  < 336   to the left,  improve=0.08664599, (0 missing)
##       Advertising < 11.5  to the left,  improve=0.05254227, (0 missing)
##   Surrogate splits:
##       Income      < 47.5  to the left,  agree=0.76, adj=0.333, (0 split)
##       Age         < 45    to the left,  agree=0.76, adj=0.333, (0 split)
##       CompPrice   < 90.5  to the left,  agree=0.68, adj=0.111, (0 split)
##       Advertising < 15    to the right, agree=0.68, adj=0.111, (0 split)
##       Population  < 296   to the right, agree=0.68, adj=0.111, (0 split)
##
## Node number 26: 21 observations
##   mean=8.818571, MSE=2.168469
##
## Node number 27: 10 observations
##   mean=11.216, MSE=2.867344
##
## Node number 34: 8 observations
##   mean=3.84875, MSE=2.911736
##
## Node number 35: 23 observations
##   mean=5.83, MSE=2.836696
##
## Node number 36: 7 observations
##   mean=3.4, MSE=2.206314
##
## Node number 37: 32 observations
##   mean=5.861562, MSE=2.415851
##
## Node number 38: 54 observations,    complexity param=0.01354372
##   mean=6.867593, MSE=2.775033
##   left son=76 (30 obs) right son=77 (24 obs)
##   Primary splits:
##       Price     < 127   to the right, improve=0.19144250, (0 missing)
##       Age       < 65    to the right, improve=0.14766490, (0 missing)
```

```
##       CompPrice < 147.5 to the left,  improve=0.12784600, (0 missing)
##       Education < 16.5  to the right, improve=0.10972240, (0 missing)
##       Income    < 41    to the left,  improve=0.09634824, (0 missing)
##   Surrogate splits:
##       CompPrice   < 132.5 to the right, agree=0.685, adj=0.292, (0 split)
##       Income      < 41    to the left,  agree=0.667, adj=0.250, (0 split)
##       Advertising < 4.5   to the right, agree=0.667, adj=0.250, (0 split)
##       Age         < 38    to the right, agree=0.611, adj=0.125, (0 split)
##       Education   < 16.5  to the right, agree=0.611, adj=0.125, (0 split)
##
## Node number 39: 13 observations
##   mean=8.706923, MSE=1.547621
##
## Node number 42: 9 observations
##   mean=7.176667, MSE=2.858156
##
## Node number 43: 16 observations
##   mean=9.514375, MSE=1.841
##
## Node number 76: 30 observations,    complexity param=0.01354372
##   mean=6.215667, MSE=2.711025
##   left son=152 (22 obs) right son=153 (8 obs)
##   Primary splits:
##       CompPrice < 147.5 to the left,  improve=0.38905020, (0 missing)
##       Age       < 65    to the right, improve=0.13371790, (0 missing)
##       Income    < 83.5  to the left,  improve=0.08814781, (0 missing)
##       Price     < 142.5 to the right, improve=0.08113575, (0 missing)
##       Education < 15.5  to the right, improve=0.05311452, (0 missing)
##   Surrogate splits:
##       Age        < 33.5  to the right, agree=0.833, adj=0.375, (0 split)
##       Population < 358.5 to the left,  agree=0.800, adj=0.250, (0 split)
##       Income     < 30.5  to the right, agree=0.767, adj=0.125, (0 split)
##       Price      < 158.5 to the left,  agree=0.767, adj=0.125, (0 split)
##
## Node number 77: 24 observations
##   mean=7.6825, MSE=1.65971
##
## Node number 152: 22 observations
##   mean=5.596364, MSE=1.783096
##
## Node number 153: 8 observations
##   mean=7.91875, MSE=1.307611
```

```
rpart.plot(tree)
```


7.4
100%
yes
ShelveLoc = Bad,Medium
no
6.7
79%
10
21%
Price >= 106
Price >= 110
6
54%
8.1
26%
9
14%
ShelveLoc = Bad
CompPrice < 124
Price >= 144
4.8
16%
6.6
38%
7.5
18%
9.6
11%
Age >= 62
CompPrice < 125
Price >= 93
CompPrice < 133
5.3
11%
5.4
14%
7.2
24%
8.7
9%
Price >= 137
Price >= 134
Advertising < 14
ShelveLoc = Bad
6.9
19%
Price >= 127
6.2
11%
CompPrice < 148
3.6
5%
3.8
3%
5.8
8%
3.4
2%
5.9
11%
5.6
8%
7.9
3%
7.7
9%
8.7
5%
6.3
9%
7.2
3%
9.5
6%
9.7
8%
6.5
2%
8.8
8%
11
4%
12
7%

```
plotcp(tree)
```




```
test$preds = predict(tree, test)
mse = mean((test$preds - test$Sales)^2)
cat("Mean squared error is: ", mse)
```

```
## Mean squared error is:  3.784419
```

- From the plot, we can see that the root node, the variable with the highest feature importance value is ShelveLoc. It is the best predictor of the model.
- Price has the second highest value for feature importance.
- The decision tree only used 5 features out of 10.
- The algorithm splits the data based on "ShelveLoc" into two categories: Bad-Medium or not. If it's either bad or medium, the next node checks if the "Price" is higher than 106. If it is, in the next node the algorithm again goes back to "ShelveLoc" and checks if the value is bad or not.

c) Use cross-validation in order to determine the optimal level of tree complexity. Does pruning the tree improve the test MSE?

```
set.seed(123)
cv_min = tree$cptable[which.min(tree$cptable[,"xerror"]),"xerror"]
cat("Lowest cross validated error is: ", cv_min)
```

```
## Lowest cross validated error is:  0.5957057
```

```
tc_min = tree$cptable[which.min(tree$cptable[,"xerror"]),"CP"]
cat("Optimal level of tree complexity is: ",tc_min)
```

```
## Optimal level of tree complexity is:  0.02428572
```

```
### Pruning the tree
imin = which.min(tree$cptable[, "xerror"])
select = which(
  tree$cptable[, "xerror"] <
    sum(tree$cptable[imin, c("xerror", "xstd")]))[1]
ptree = prune(tree, cp = tree$cptable[select, "CP"])

test$pruned_preds =  predict(ptree, test)
mse_pruned = mean((test$pruned_preds - test$Sales)^2)
cat("Mean squared error is: ", mse_pruned)
```

```
## Mean squared error is:  4.979248
```

According to the results from part b, we can say that pruning the tree did not improve the test MSE. There may be different reasons for such a case. One possible explanation is that pruning simplifies the tree by removing some of the complex branches, reducing the model's overfitting problem. However, if the tree was suffering from sever overfitting, pruning may decrease the predictive power, increasing test MSE.

Another reason may be that pruning can remove important splits that were important and the removed splits might have been capturing meaningful patterns or relationships in the data, and when we eliminate them via pruning, the model may become less accurate, which explains the increased test MSE.

Task 6

The dataset *icu* in package aplore3 contains information on patients who were admitted to an adult intensive care unit (ICU). The aim is to develop a predictive model for the probability of survival to hospital discharge of these patients.

Fit a classification tree to the data without pre-processing:

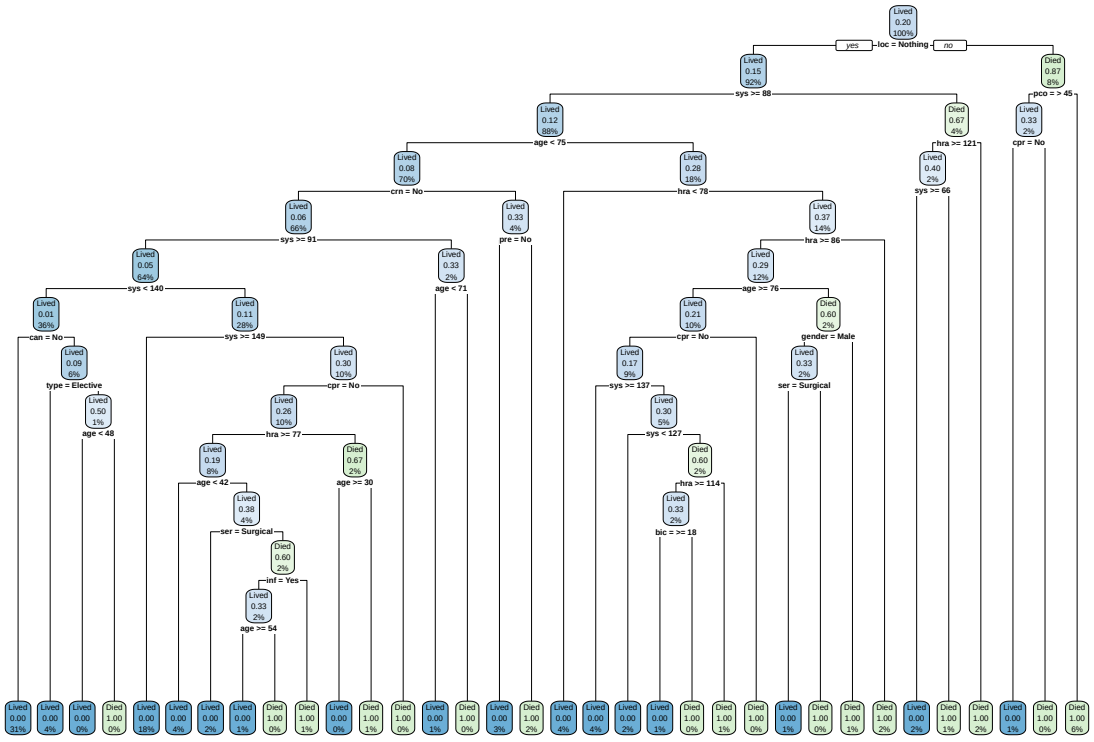
- Use very loose stopping criteria such that the tree might be overfitting.
- Inspect the fitted tree and describe it.

```
library(aplore3)
```

```
## Warning: Paket 'aplore3' wurde unter R Version 4.2.3 erstellt
```

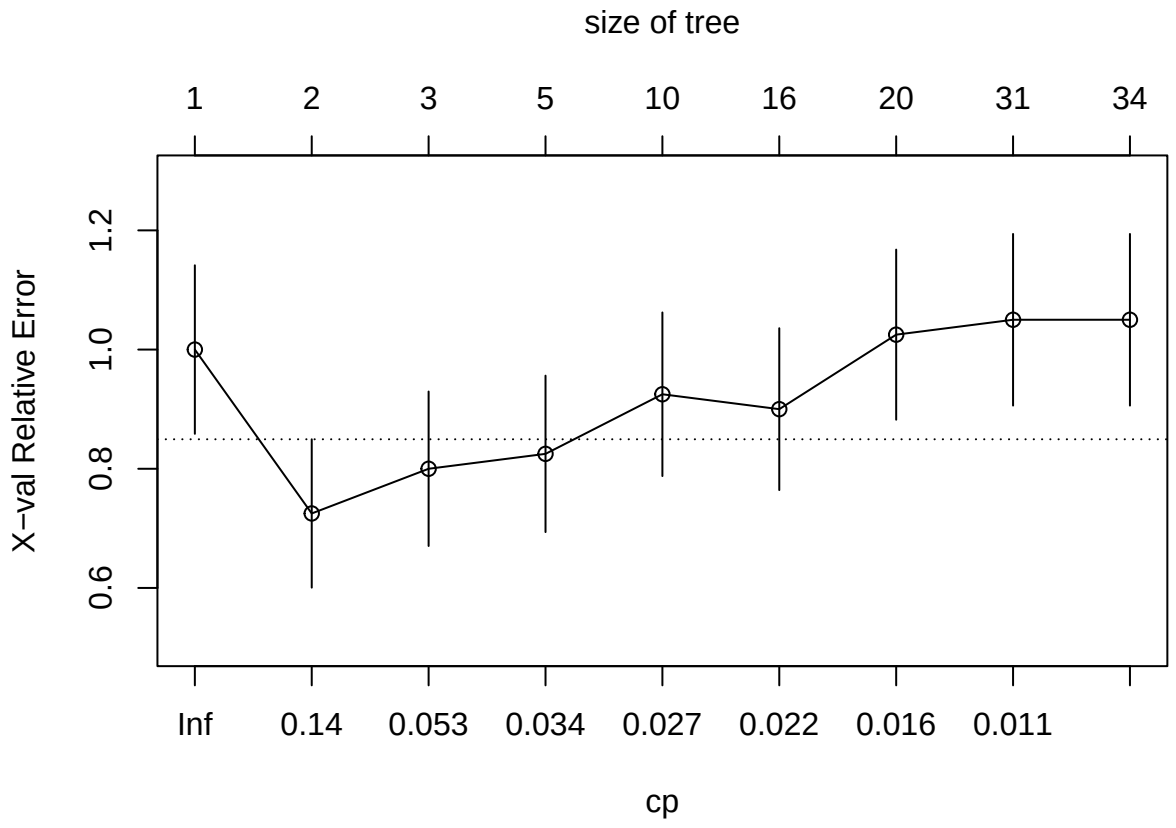
```
library(rpart)
library(rpart.plot)
data <- icu
# Fit classification tree without column "id"
```

```
# minsplit = 2 means that the node can split if there are at least 2 observations
# Small value for cp means that we allow for a more complex tree,
# therefore more likely to overfit
tree_overfit <- rpart(sta ~ ., data = icu[, -1],
                      control = rpart.control(minsplit = 2, cp = 0.0001))
rpart.plot(tree_overfit)
```



We fit a classification tree to the data using very loose stopping criteria. The response is binary, if the patient dies or lives. As we can see the tree has a depth of 14 levels which means that we might have up to 13 decisions before getting to the prediction. This is also an index that we might have overfitting (which we wanted). On the first level we start with the variable loc, which has the information wheter a patient is in a coma, stuporous or concious (nothing). This suggests that this has a huge impact on wheter a patient dies or lives. The leaf nodes exhibit either a 0% or 100% probability of death (0 or 1), with no intermediary values. This suggests a high level of confidence in the model's predictions, potentially indicating overconfidence. The absence of probabilities in the middle range could be a sign of overfitting, as it reflects a lack of allowance for uncertainty in predictions. Additionally, most terminal nodes contain a notably small sample size. Such limited observations in terminal nodes may imply that decisions are influenced by noise or outliers in the training data. Predictions stemming from these nodes might be less reliable, particularly when applied to new data. Also worth mentioning is that the variables age and sys appear multiple times across various branches.

```
plotcp(tree_overfit)
```




```
printcp(tree_overfit)
```

```
##
## Classification tree:
## rpart(formula = sta ~ ., data = icu[, -1], control = rpart.control(minsplit = 2,
##     cp = 1e-04))
##
## Variables actually used in tree construction:
##  [1] age    bic    can    cpr    crn    gender hra    inf    loc    pco
## [11] pre    ser    sys    type
##
## Root node error: 40/200 = 0.2
##
## n= 200
##
##          CP nsplit rel error xerror    xstd
## 1 0.2750000      0     1.000  1.000 0.14142
## 2 0.0750000      1     0.725  0.725 0.12449
## 3 0.0375000      2     0.650  0.800 0.12961
## 4 0.0300000      4     0.575  0.825 0.13123
## 5 0.0250000      9     0.425  0.925 0.13728
## 6 0.0187500     15     0.275  0.900 0.13583
## 7 0.0142857     19     0.200  1.025 0.14273
## 8 0.0083333     30     0.025  1.050 0.14401
## 9 0.0001000     33     0.000  1.050 0.14401
```

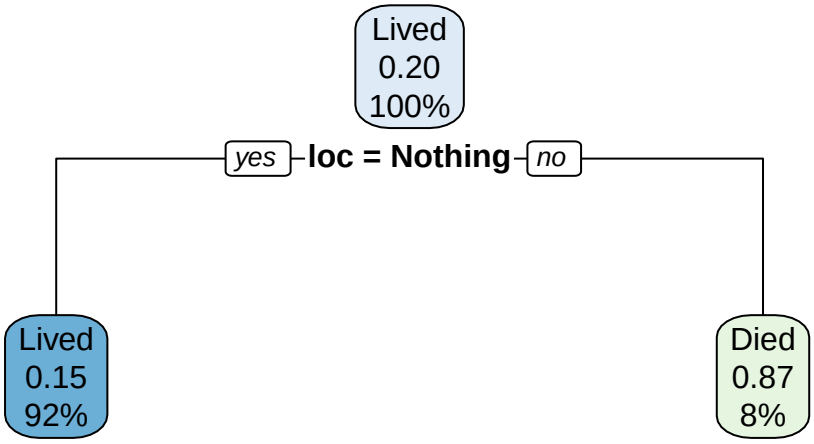
From the table we see that the relative error decreases the more splits we have. The cross-validated error on the other hand increases again after a certain complexity. This suggests that a more complex tree is not always better.

- Use pruning to select a tree with a suitable size. Determine this smaller tree and inspect and describe it.

```
imin <- which.min(tree_overfit$cptable[, "xerror"])
select <- which(tree_overfit$cptable[, "xerror"]
                < sum(tree_overfit$cptable[imin, c("xerror", "xstd")]))[1]
# Use pruning
cv_tree <- prune(tree_overfit, cp = tree_overfit$cptable[select, "CP"])
printcp(cv_tree)
```

```
##
## Classification tree:
## rpart(formula = sta ~ ., data = icu[, -1], control = rpart.control(minsplit = 2,
##     cp = 1e-04))
##
## Variables actually used in tree construction:
## [1] loc
##
## Root node error: 40/200 = 0.2
##
## n= 200
##
##      CP nsplit rel error xerror    xstd
## 1 0.275      0     1.000  1.000 0.14142
## 2 0.075      1     0.725  0.725 0.12449
```

```
rpart.plot(cv_tree)
```



As we can see we now only have two levels left, so the tree got a lot simpler. We only have the prediction if someone lives or dies, based on the variable loc, which holds the information if someone is in a coma, is stuporous or is concious (nothing). We can interpret the tree easily now, but we also don't consider the impact of other variables on the outcome.

Task 7

```
set.seed(123)
x_1 = runif(100)
x_2 = rnorm(100)
x_3 = as.integer(rbernoulli(100))
```

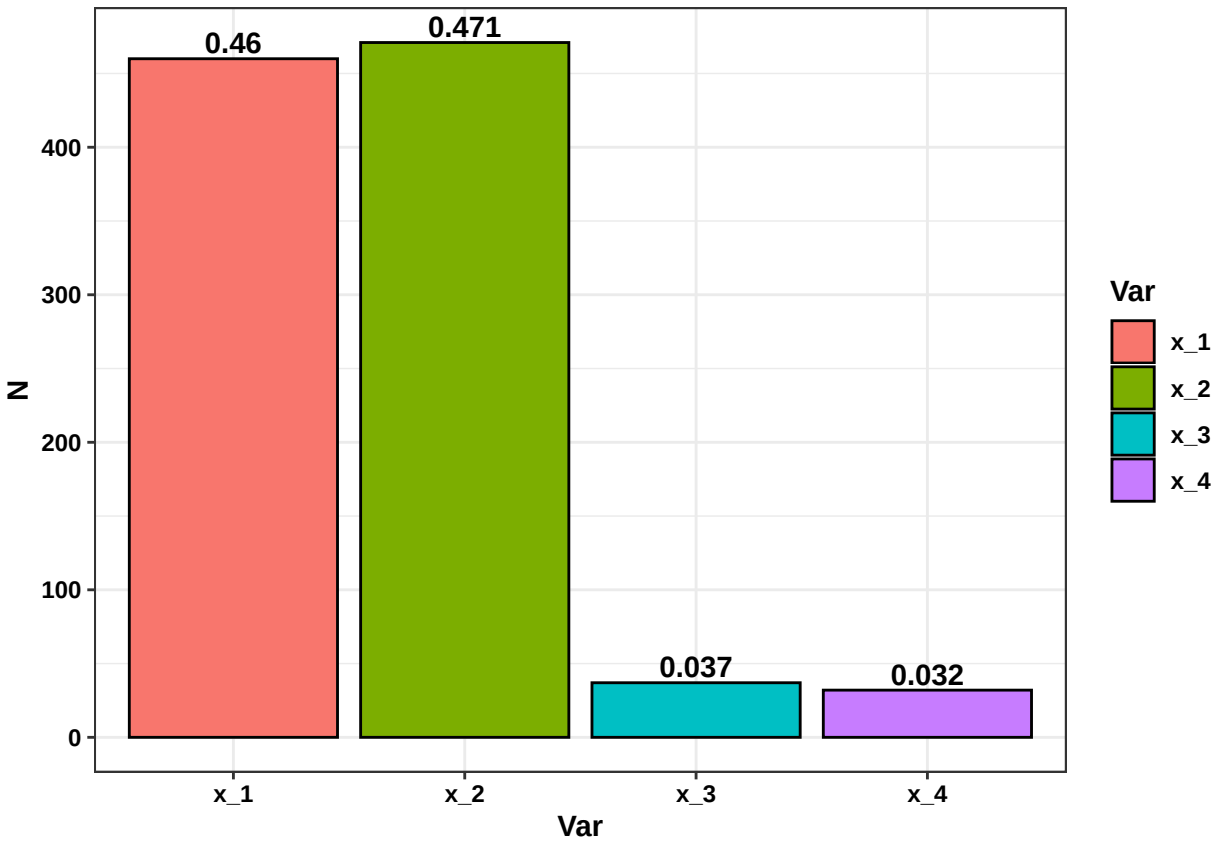
```
## Warning: ‘rbernoulli()‘ was deprecated in purrr 1.0.0.
## This warning is displayed once every 8 hours.
## Call ‘lifecycle::last_lifecycle_warnings()‘ to see where this warning was
## generated.
```

```
x_4 = as.integer(rbernoulli(100 , p=0.1))

df = data_frame(x_1=x_1, x_2=x_2,x_3=x_3, x_4=x_4)
```

```
## Warning: ‘data_frame()‘ was deprecated in tibble 1.1.0.
## i Please use ‘tibble()‘ instead.
## This warning is displayed once every 8 hours.
```

```
## Call 'lifecycle::last_lifecycle_warnings()' to see where this warning was
## generated.
```

```
result = NULL
for (i in 1:1000)
{
  y=rnorm(100)
  tree = rpart(y ~ ., data = df, control = list(maxdepth = 1))
  temp = paste0(".*(", paste(colnames(df), collapse="|"), ").*")
  result = rbind(result, unique(sub(temp,"\\1", labels(tree)[-1])))
}

result %>% as.data.frame %>%
  rename(Var='V1') %>%
  group_by(Var) %>% summarise(N=n()) %>%
  ggplot(aes(x=Var,y=N,fill=Var))+
  geom_bar(stat = 'identity',color='black')+
  scale_y_continuous(labels = scales::comma_format(accuracy = 2))+
  geom_text(aes(label=N/sum(N)),vjust=-0.25,fontface='bold')+
  theme_bw()+
  theme(axis.text = element_text(color='black',face='bold'),
        axis.title = element_text(color='black',face='bold'),
        legend.text = element_text(color='black',face='bold'),
        legend.title = element_text(color='black',face='bold'))
```



```
df_results=table(result)
kable(df_results)
```

result	Freq
x_1	460
x_2	471
x_3	37
x_4	32

According to the results, it's clear that variables X_1 and X_2 were selected more frequently for splitting in comparison to X_3 and X_4. This observation is in line with the fundamental behavior of decision trees, which tend to choose independent variables with distributions resembling that of the dependent variable y. Decision trees try to discover splits that minimize the variance, and given that y follows a normal distribution, it makes sense for the decision tree to favor independent variables with distributions closer to the normal distribution. Keeping that in mind, since X_2 follows a standard normal distribution which is more similar to the normal distribution compared to Bernoulli(Binomial) distribution the model also chooses X_1 more often for splitting. We would expect X_2 to be chosen more frequently than the other independent variables and the results are aligned with our expectations.

To summarize, the frequencies of X_1 and X_2, being higher than X_3 and X_4, can be explained by the distributional similarities to y.

Task 8

The data generating process is given by $y = x + \epsilon$, where $x \sim N(0, 1), \epsilon \sim N(0, 0.1)$. We first draw 100 samples of 100 observations for y and store it in a list.

```
set.seed(123)
# get 100 training data sets of length 100
training_data <- lapply(1:100, function(i) {
  x <- rnorm(100)
  noise <- rnorm(100, mean = 0, sd = sqrt(0.1))
  y <- x + noise
  data.frame(y = y, x = x)
})
```

To approximate the prediction error we draw 100 test data sets from the data generating process but of length 20 each.

```
# get 100 test data sets
test_data <- lapply(1:100, function(i) {
  x <- rnorm(20)
  noise <- rnorm(20, mean = 0, sd = sqrt(0.1))
  y <- x + noise
  data.frame(y = y, x = x)
})
```

We then first estimate 100 linear regressions of the form $y = \beta_0 + \beta_1 x + \epsilon$.

```r
# get OLS fits
linear_fits <- lapply(training_data, lm, formula = y ~ x)

lm_pred_err <- lapply(linear_fits, function(l) {
  # get mean squared loss for each test data set
  mean_squared_loss <- lapply(test_data, function(d) {
    pred <- predict(l, newdata = d)
    mean((d$y - pred)^2)
  })
  # prediction error
  mean(unlist(mean_squared_loss))
})
```

Next, we fit regression trees for each of the training data sets.

```r
library(caret)
```

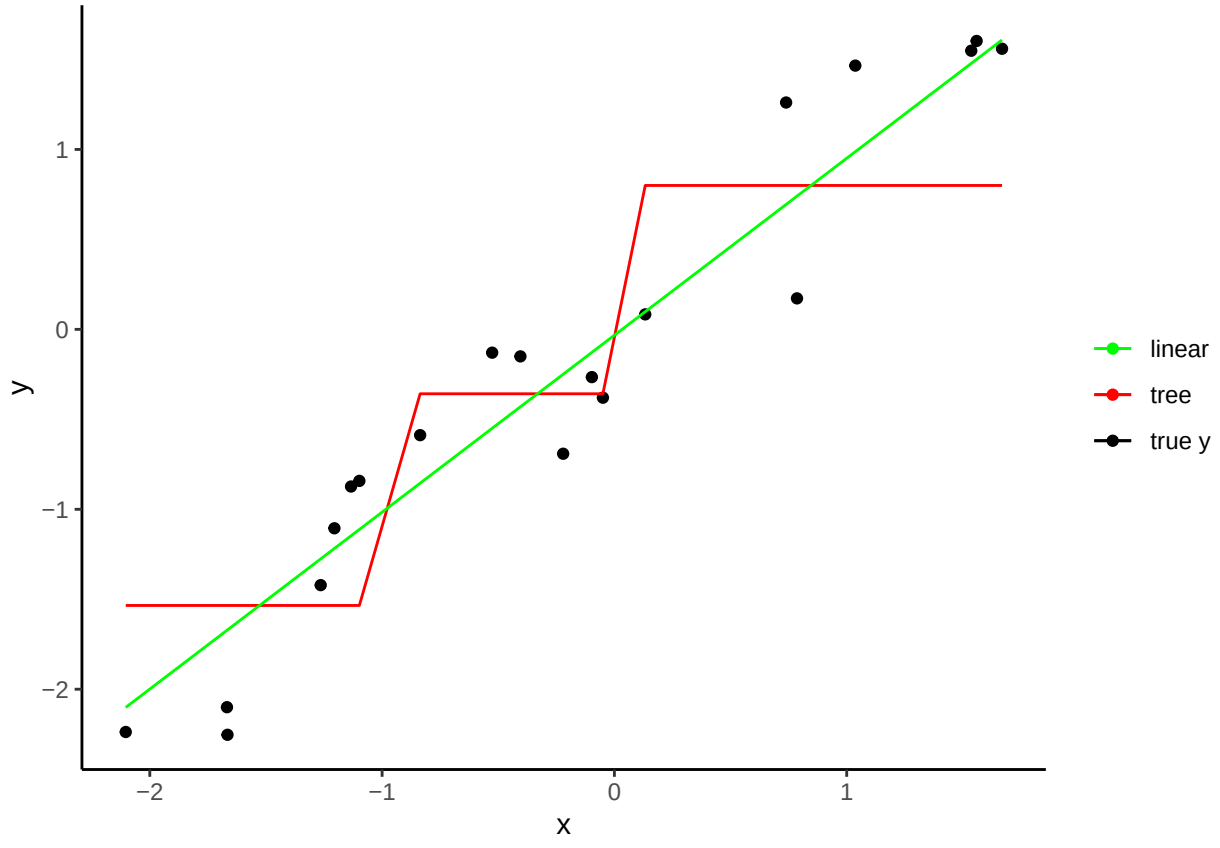
```
## Lade nötiges Paket: lattice

##
## Attache Paket: 'caret'

## Das folgende Objekt ist maskiert 'package:purrr':
##
##     lift
```

```r
# fit trees
tree_fits <- lapply(training_data, function(d) {
  # use caret for convenience, cost complexity
  # pruning is conducted automatically and best
  # model is by default chosen by RMSE
  train(y ~ x,
        data = d,
        method = "rpart")
})

# determine size as the number of nodes
tree_sizes <- lapply(tree_fits, function(t) {
  # get frame and count non leaf entries
  sum(t$finalModel$frame$var != "<leaf>")
})

# get the prediction error by monte carlo simulation,
# i.e. by calculating the mean squared error for 1000 test
# data sets and averaging those again
tree_pred_err <- lapply(tree_fits, function(t) {
  # get mean squared loss for each test data set
  mean_squared_loss <- lapply(test_data, function(d) {
    pred <- predict(t, newdata = d)
    mean((d$y - pred)^2)
  })
  # prediction error
```

```
  mean(unlist(mean_squared_loss))
})
```

As instructed we now pick a test data seit, in our case the very first one and look at the actual values vs the predicted values for both the linear model as well as the regression tree.

```
pred_l <- predict(linear_fits[[1]], newdata = test_data[[1]])

pred_t <- predict(tree_fits[[1]], newdata = test_data[[1]])

data <- test_data[[1]]
data$l <- pred_l
data$t <- pred_t

data %>%
  ggplot(aes(x = x)) +
  geom_point(aes(y = y, color = "true y")) +
  geom_line(aes(y = t, color = "tree")) +
  geom_line(aes(y = l, color = "linear")) +
  scale_color_manual(values = c("true y" = "black", "tree" = "red", "linear" = "green")) +
  theme_classic() +
  theme(legend.title = element_blank())
```



Then we summarise different tree sizes

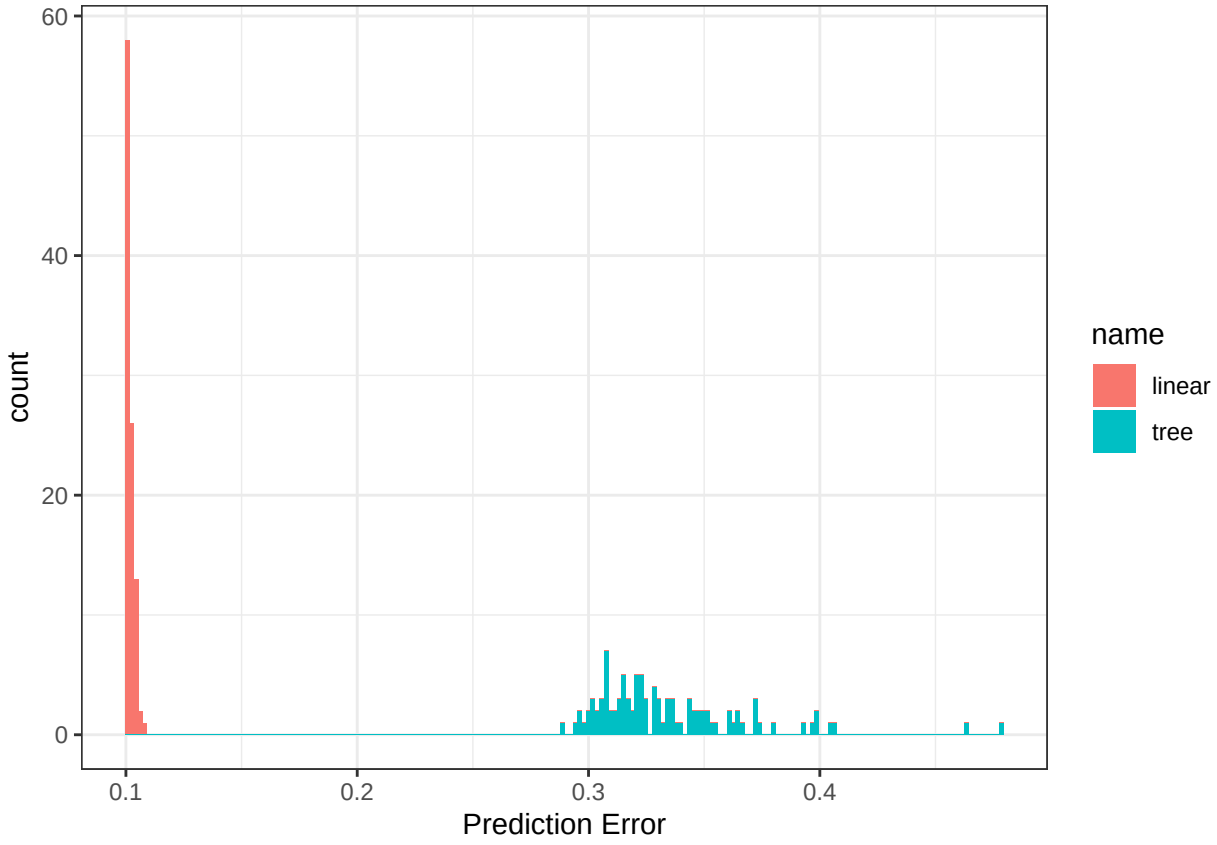
```
table(unlist(tree_sizes))
```

```
##
##  1  2
##  2 98
```

and observe that there are never more than 2 nodes in the trees.

Next we summarise the estimated prediction errors. We use a two variable histogram to do so.

```
prediction_errors <- data.frame(linear = unlist(lm_pred_err),
                                tree = unlist(tree_pred_err))

prediction_errors %>%
  pivot_longer(cols = everything()) %>%
  ggplot(aes(x = value, fill = name)) +
  geom_histogram(bins = 200) +
  theme(legend.title = element_blank()) +
  theme_bw() +
  labs(x = "Prediction Error")
```



What we observe is that the tree indeed has problems to capture the clearly linear relationship between y and x. This is because the tree takes the originally continuous space and discretizes it, such that predictions do not correspond to a certain value of x but rather a range within the support of x. One can see that quite clearly in the plot that visualizes the predictions vs the true value of y in the test data set. The tree predicts in a stepwise manner while the linear regression perfectly captures the linear relationship. Also,

we can see that the prediction errors are much lower for linear regression compared to the regression trees which is no surprise looking at the difficulties regression trees have with linear relationships.

Task 9

We assume data with the following data generation process:

$$x = y + \epsilon$$

where y is a categorical variable with values 1, 2, 3, which occur with equal probability and $\epsilon \sim N(0, 0.2)$ independent.

- Draw 100 data sets of size 100

Here we simply sample the classes in y, the white noise ϵ to generate the observations x.

```
set.seed(137)
data_gen <- function(n_samples){

  i <- 1
  vmat <- list()

  repeat{

    # Break Condition
    if(i == (n_samples + 1)) break

    # Generate 100 random numbers
    y <- sample(c(1, 2, 3), size = 100, replace = T)
    eps <- rnorm(100, mean = 0, sd = sqrt(0.2))
    x <- y + eps

    # Put it in a matrix
    tmp <- data.frame(x, y, eps)
    vmat[[i]] <-  tmp

    i <- i + 1
  }

  return(vmat)
}

vmat <- data_gen(n_samples = 100)
```

- Determine the sum of the misclassification rates, Gini indices and deviance criteria weighted with the number of observations in each subgroup for the subgroups obtained when splitting the observations using x with thresholds 1.5, 2, and 2.5 and y as dependent variable in the classification problem.

```
# Impurity measures functions
misc_err <- function(p) return( 1 - max(p) )
gini <- function(p) return( sum(p * (1-p)) )
deviance <- function(p) return( (-1) *sum(p * log(p)) )
```

```r
# Given thresholds
thresholds <- c(1.5, 2, 2.5)

# Function that calculates the impurity given the impurty measure function,
# the threshold for the split for one single dataset.

impurity_calculate <- function(df, class = "y", col = "x", thresh, FUN){

  # Slice the dataset given the threshold into 2 subgroups
  df$split <- cut(df[, col], c(-Inf, thresh, Inf))

  # 0 is the <= thresh , 1 is the >thresh
  levels(df$split) <- c(0, 1)

  # Number of observations for weighting later on
  nobs <- table(df$split)

  # Derive a matrix to count the observations for each split
  class_mat <- table(df[, c("split", class)])
  calc_mat <- t(apply(class_mat, 1, function(x) x/sum(x)))

  # Impurity function act here
  res <- apply(calc_mat, 1, FUN)

  # Pay attention to NaN Values
  res[is.nan(res)] <- 0

  # Weighted impurity measure for the split.
  res_w <- res*nobs

  # Return the overall impurity as the sum for each group
  return(sum(res_w))
}

generate_error_tables <- function(impurity_criterion, thresholds, df_list){

  j <- 1

  out_mat <- matrix(rep(0, 300), nrow = 100)
  colnames(out_mat) <- thresholds

  for(t in thresholds){

    for(i in 1:length(df_list)){

      tmp <- impurity_calculate(df_list[[i]], class = "y", col = "x", thresh = t, FUN = impurity_criteri
      out_mat[i, j] <- tmp

    }

    j <- j + 1

  }
```

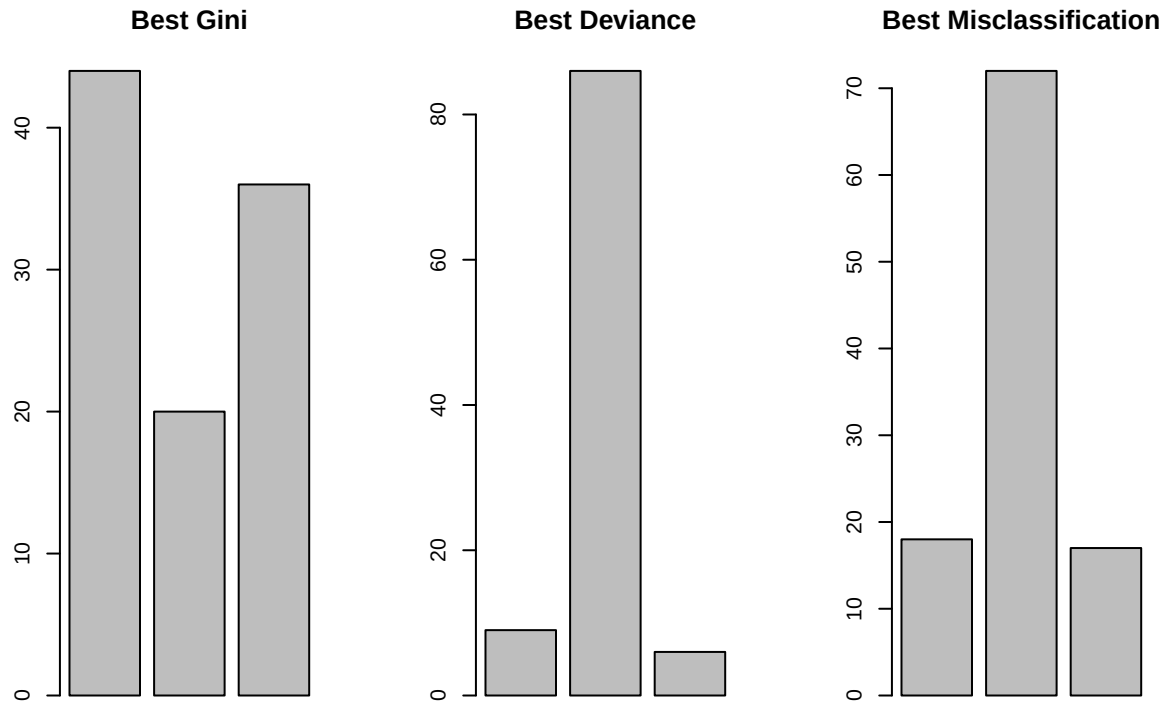
```
  return(out_mat)

}

# Generate error tables: these contain the sum of the errors for each split for each dataset
deviance_table <- generate_error_tables(deviance, thresholds, vmat)
gini_table <- generate_error_tables(gini, thresholds, vmat)
misc_table <- generate_error_tables(misc_err, thresholds, vmat)

sum_table <- rbind(colSums(deviance_table), colSums(gini_table), colSums(misc_table))
rownames(sum_table) <- c("Deviance", "Gini", "Misclassification")

kableExtra::kable(sum_table)
```

	1.5	2	2.5
Deviance	5751.995	2381.412	5848.391
Gini	4386.788	4440.964	4411.143
Misclassification	3628.000	3402.000	3647.000

- Calculate the best threshold according to each of the three impurity measures for each of the 100 data sets. Summarize and interpret the results.

```
best <- function(vector) as.integer(vector == min(vector))

best_deviance <- apply(t(apply(deviance_table, 1, best)), 2, sum)
best_gini <- apply(t(apply(gini_table, 1, best)), 2, sum)
best_misc <- apply(t(apply(misc_table, 1, best)), 2, sum)

best_matrix <- matrix(c(best_misc, best_gini, best_deviance), nrow = 3, byrow = T)
rownames(best_matrix) <- c("Misclasssification", " Gini", "Deviance")
colnames(best_matrix) <- c("1.5", "2", "2.5")

kableExtra::kable(best_matrix)
```

	1.5	2	2.5
Misclasssification	18	72	17
Gini	44	20	36
Deviance	9	86	6



From the results, we see that the for both the Deviance and the Misclassification error, the threshold at 2 is performs significantly better, whether for the Gini Index, this is not true anymore and the extreme split points are better performing. One explanation we can give around this, is the fact that for the Misclassification error, the scaled up coefficients turn out to be larger for the subgroup containing the majority of observations, i.e. the ones subgroups $\{obs_i > 1.5\}$, $\{obs_i < 2.5\}$. Hence, this is clear from the results. We note that the Deviance behaves in a similar way in comparison with the Misclassification error, selecting 2 as the best threshold. The Gini Index is the one which selects the split at 2 the least amount of times. We interpret this with the fact that by construction having one class generally not appearing in the split, is weighted much more in the calculation rather than having a slightly more impure second node.

Task 10

We draw 10 observations from the data generating process $y \sim N(0, 3)$.

```
set.seed(123)
# draw sample s
n = 10
y <- rnorm(n = n, mean = 0, sd = sqrt(3))

# calculate mean
mean_sample <- mean(y)

# bootstrap: draw 10 times with replacement from the sample,
# do this 1000 times
B = 1000
bootstrap_samples <- lapply(1:B, function(i) {
```

```
  sample(x = y, size = n, replace = T)
})

# calculate the means for each bootstrap sample
means_b <- do.call("c", lapply(bootstrap_samples, mean))

# calculate the mean over the means
mean_sample
```

```
## [1] 0.1292554
```

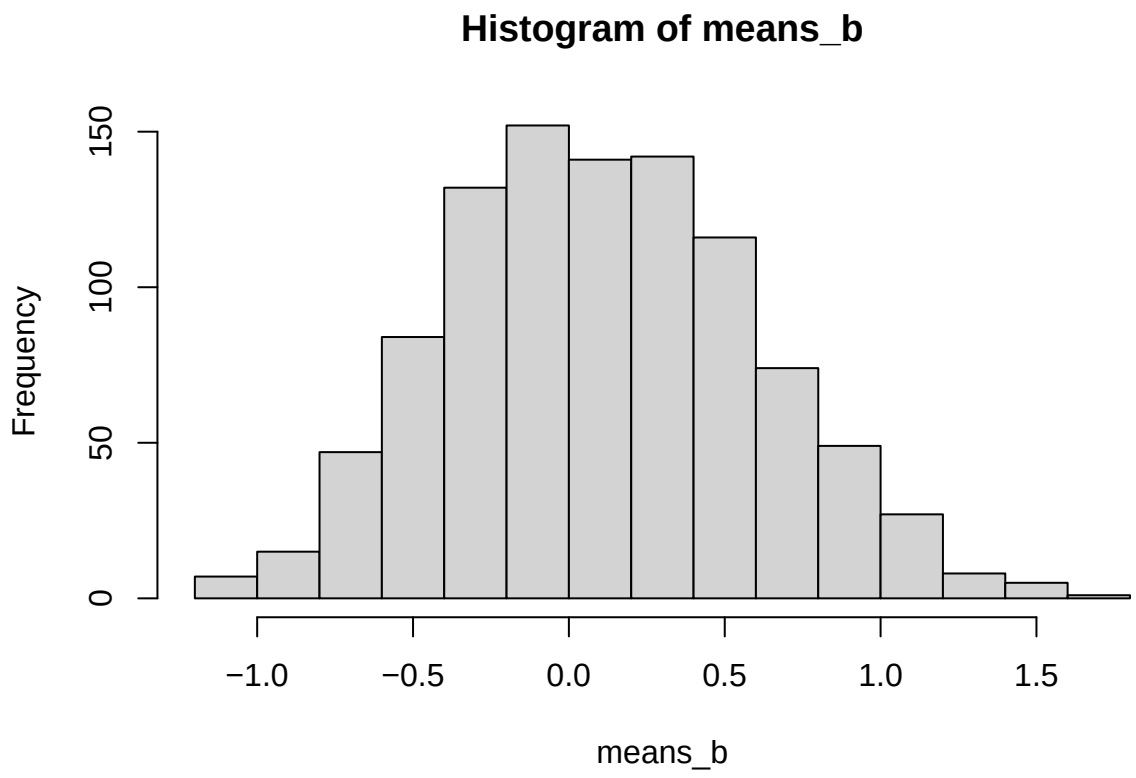
```
mean(means_b)
```

```
## [1] 0.1079687
```

```
sd(means_b)
```

```
## [1] 0.4960259
```

```
hist(means_b)
```



In the above we relied on the empirical distribution of y_s and estimated the mean by Monte-Carlo simulation by creating a distribution of means and taking the mean of this distribution as our bootstrapping estimate

for the popopulation mean. We see that the sample mean and the bootstrapping estimate are very similar but quite far from the actual mean of the DGP. This observation is not surprising as as the initial sample has very limited information given $n = 10$ while the support of y is $(-\infty, \infty)$. Thus we know that this just can be an approximation of the true bootstrapping distribution.

The true bootstrapping distribution of the mean of the initially drawn sample would be the set of means drawn from all possible combinations of the values in the initial sample with replacement and without ordering. The number of these possible combinations in this case is $\binom{n+k-1}{k} = \binom{19}{10} = 92378$ if the order does not matter. If the order matters it would be just $n^n = 10^{10}$. However determining both sets is computationally demanding if not even infeasible. The bootstrapping estimate for the mean from the true bootstrapping distribution would be the population mean, so the bootstrapping procedure we used is just an approximation.