Mandatory 1

March 1, 2025

STK2100 - Håkon Ganes Kornstad

Problem 1

a) Given the regression models for the response Y and single covariate x, we can write the following as linear regression models
As is (i):

$$2. \quad Y = \beta_0 + \beta_1 \frac{1}{x} + \beta_2 x^2 + \varepsilon$$

If a parameter is kept fixed (ii):

$$\begin{aligned} 1. \quad Y &= \frac{\beta_0}{1+\beta_1 x} + \beta_2 x^{1/2} + \varepsilon \\ &\Downarrow \quad Z = \frac{1}{1+\beta_1 x} \quad | \ \beta_1 \text{ fixed} \\ Y &= \beta_0 Z + \beta_2 x^{1/2} + \varepsilon \end{aligned}$$

$$4. \quad Y = \beta_0 + \beta_1 x^{\beta_2} + \varepsilon \quad | \ \beta_2 \text{ fixed}$$

After a suitable transformation (iii):

$$\begin{aligned} 3. \quad Y &= \frac{1}{\beta_0 + \beta_1 x + \beta_2 x^2 + \varepsilon} \\ &\Downarrow \ Z = (Y^{-1}) \\ Z &= \beta_0 + \beta_1 x + \beta_2 x^2 + \varepsilon \end{aligned}$$

Both (ii) and (iii):

$$\begin{aligned} 5. \quad Y &= \beta_0 x^{\beta_1} \ \varepsilon \\ &\Downarrow \ Z = (\ln Y), \text{ and } \beta_0 \text{ fixed} \\ Z &= \ln(\beta_0) + \beta_1 \ln(x) + \ln(\varepsilon) \end{aligned}$$

Note: For 5. we are assuming here also that $\ln(\varepsilon)$ is from the same distribution as ε.

b) For the models that can be used, their dimensions will be * For 1. (after transformation) $\mathbf{X}:$ $(N \times 2)$ matric (without intercept), $\beta:$ (2×1) vector * For 2. $\mathbf{X}:$ $(N \times 3)$ matrix (with

intercept), β : (3×1) vector * For 4. $\mathbf{X}$: $(N \times 2)$ matrix (with intercept), β : (2×1) vector * For 3. (after transformation) $\mathbf{X}$: $(N \times 3)$ matrix (with intercept), β : (3×1) vector * For 5. (after transformation) $\mathbf{X}$: $(N \times 2)$ matrix (with intercept), β : (2×1) vector

Problem 2

a) Fit a linear regression model with $y = \log($ `cost` $)$ as the response, including all 10 covariates.

[1]:
```
options(warn = -1) # suppressing warnings
```

[2]:
```
nuclear.data <- read.table("https://www.uio.no/studier/emner/matnat/math/
↪STK2100/data/nuclear.dat", sep = "\t", header = FALSE)
```

[3]:
```
# Converting first row to column names, and then deleting it
colnames(nuclear.data) <- nuclear.data[1, ]
nuclear.data <- nuclear.data[-1, ]
head(nuclear.data) # Taking a look at the data

# Setting a seed for reproducibility
set.seed(1977)
```

A data.frame: 6×11

	cost <chr>	date <chr>	t1 <chr>	t2 <chr>	cap <chr>	pr <chr>	ne <chr>	ct <chr>	bw <chr>	cum <chr
2	460.05	68.58	14	46	687	0	1	0	0	14
3	452.99	67.33	10	73	1065	0	0	1	0	1
4	443.22	67.33	10	85	1065	1	0	1	0	1
5	652.32	68	11	67	1065	0	1	1	0	12
6	642.23	68	11	78	1065	1	1	1	0	12
7	345.39	67.92	13	51	514	0	1	1	0	3

[4]:
```
nuclear.data <- data.frame(apply(nuclear.data, 2, as.numeric)) # converting the␣
↪chr's to numbers
nuclear.fit <- lm(log(cost) ~ . , data = nuclear.data) # fitting the model
summary(nuclear.fit) # taking a look at the summary
```

```
Call:
lm(formula = log(cost) ~ ., data = nuclear.data)

Residuals:
      Min        1Q    Median        3Q       Max
-0.284032 -0.081677  0.009502  0.090890  0.266548

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) -1.063e+01  5.710e+00  -1.862  0.07662 .
date         2.276e-01  8.656e-02   2.629  0.01567 *
t1           5.252e-03  2.230e-02   0.236  0.81610
t2           5.606e-03  4.595e-03   1.220  0.23599
```

```
cap          8.837e-04  1.811e-04   4.878 7.99e-05 ***
pr          -1.081e-01  8.351e-02  -1.295  0.20943
ne           2.595e-01  7.925e-02   3.274  0.00362 **
ct           1.155e-01  7.027e-02   1.644  0.11503
bw           3.680e-02  1.063e-01   0.346  0.73261
cum.n       -1.203e-02  7.828e-03  -1.536  0.13944
pt          -2.220e-01  1.304e-01  -1.702  0.10352
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.1697 on 21 degrees of freedom
Multiple R-squared:  0.8635,        Adjusted R-squared:  0.7985
F-statistic: 13.28 on 10 and 21 DF,  p-value: 5.717e-07
```

To find the confidence interval for the coefficients β_j, we need to compute

$$\frac{\beta - \hat{\beta}}{\hat{\sigma}\sqrt{\mathbf{x}^T(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{x}}} \sim t_{N-p-1} \quad \rightarrow \quad \beta_j \pm t_{1-\alpha/2;N-p-1}\hat{\sigma}\sqrt{v_j}$$

[5]:
```r
# setting up the confidence interval manually
N <- nrow(nuclear.data)
p <- length(coef(nuclear.fit))
alpha <- 0.05 #95% CI

beta.hats <- coef(nuclear.fit)[c("t1", "t2", "bw")]

sigma.hat <- sqrt(sum(nuclear.fit$residuals ^ 2)/ (N - p - 1))

X <- model.matrix(nuclear.fit)
v <- solve(t(X)%*%X)

se.beta.hats <- sigma.hat * sqrt(diag(v)[c("t1", "t2", "bw")])

t.value <- qt(1 - alpha/2, df = N - p - 1) #t-value from the t-distribution␣
 ↪(using qt) here

conf.ints <- cbind(
    beta.hats - t.value * se.beta.hats, # lower bound
    beta.hats + t.value * se.beta.hats # upper bound
    )
conf.ints

# checking against the confint()-function
confint(nuclear.fit, level=0.95)[c("t1", "t2", "bw"), ] # constructing␣
 ↪confidence intervals for the given parameters
```

A matrix: 3 × 2 of type dbl

t1	-0.042413659	0.05291712
t2	-0.004215791	0.01542773
bw	-0.190361111	0.26395337

A matrix: 3 × 2 of type dbl

	2.5 %	97.5 %
t1	-0.041123331	0.05162679
t2	-0.003949911	0.01516185
bw	-0.184211843	0.25780411

Note: We observe that our manually constructed confidence intevals deviate slightly from the ones created by R's `confint()` function. This might be because of the discrepancy between `solve(t(X)%*%X)` and the internal variance computations in R. However, the deviations are acceptable, especially since zero is included in all intervals, which is perhaps the most important information for further analysis.

b) From the theory, we have that a $(1-\alpha)\cdot 100\%$ prediction interval for Y is given by

$$\hat{y} \pm t_{1-\alpha/2;N-p-1}\hat{\sigma}\sqrt{1+\mathbf{x}^T(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{x}}$$

In our case, we have that $Y = \log(Z)$, and transform using $Z = \exp(Y)$. Furthermore, since the exp-function is monotonous, the probability content of the prediction interval remains unchanged when transforming both bounds. This follows from the property that monotone transformations of an interval preserve the probability:

$$P(L < Y < U) = P(\exp L < Z < \exp U)$$

We then end up with:

$$\exp\left(\hat{y} \pm t_{1-\alpha/2;N-p-1}\hat{\sigma}\sqrt{1+\mathbf{x}^T(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{x}}\right) = (\exp(L), \exp(U)),$$

where $\exp(L)$ is the transformation of the lower bound and $\exp(U)$ is the transformation of the upper bound of this prediction interval. We have now shown that the $(1-\alpha)\cdot 100\%$ prediction interval for Z is given by $(\exp(L), \exp(U))$.

[6]:
```
# creating a data frame from the given x covariate vector
new.x <- data.frame(date = 70.0, t1 = 13, t2 = 50, cap = 800, pr = 1, ne = 0,␣
 ↪ct = 0, bw = 1, cum.n = 8, pt = 1)

pred.y <- predict(nuclear.fit, new.x, interval="predict")

# getting the cost z=(exp(y)) by transforming y
pred.z <- exp(pred.y)
pred.z
```

A matrix: 1 × 3 of type dbl

	fit	lwr	upr
1	389.2163	220.1366	688.1607

c) Individual hypothesis tests for `t1`, `t2`, `bw`

[7]:
```r
t_values <- beta.hats / se.beta.hats
p_values <- 2 * pt(-abs(t_values), (N - p - 1))

# printing in table form
results <- data.frame(
  Estimate = beta.hats,
  Std_Error = se.beta.hats,
  t_value = t_values,
  p_value = p_values
)
print(results)

# comparing with R's summary function
summary(nuclear.fit)$coefficients[c("t1", "t2", "bw"), ]
```

```
     Estimate  Std_Error   t_value   p_value
t1 0.005251730 0.02285054 0.2298296 0.8205584
t2 0.005605968 0.00470850 1.1906059 0.2477485
bw 0.036796131 0.10889800 0.3378954 0.7389630
```

A matrix: 3 × 4 of type dbl

	Estimate	Std. Error	t value	Pr(>|t|)
t1	0.005251730	0.022299843	0.2355053	0.8160981
t2	0.005605968	0.004595026	1.2200080	0.2359862
bw	0.036796131	0.106273564	0.3462397	0.7326075

- `t1`: $p(>|t|) = 0.816 \gg 0.05 \longrightarrow$ `t1` is not significant (H_0 can not be discarded)
- `t2`: $p(>|t|) = 0.236 \gg 0.05 \longrightarrow$ `t2` is not significant
- `bw`: $p(>|t|) = 0.733 \gg 0.05 \longrightarrow$ `bw` is not significant

Again, we see a slight deviation from our computed results and R's "native" results. And again, this is due to how we computed the variance. We then find a **joint test** by computing the F-statistic and $p(>|F|)$ between the reduced set and the original set, by using the formula

$$F = \frac{(RSS_0 - RSS)/q}{RSS/(N-p-1)},$$

where RSS_0 is the residual sum of squares from the reduced model, RSS is from the full model, q is the amount of removed covariates, N is the total observations (rows), and p are the number of covariates in the full model (excluding intercept). Furthermore, we subtract 1, which is the df belonging to the intercept. In R we may obtain the same values with the `anova()` function.

[8]:
```r
# the full model with its rss
full_model <- nuclear.fit
rss_full <- sum(residuals(full_model)^2)

# making a reduced model by subtracting t1, t2, and its rss
reduced_model <- lm(log(cost) ~ . -t1 -t2 -bw, data = nuclear.data)
rss_reduced <- sum(residuals(reduced_model)^2)
```

```r
q <- 3 # number of covariates we want to remove
N <- nrow(nuclear.data) # number of observations
p <- length(coef(full_model)) - 1 # number of parameters in full model

F_stat <- ((rss_reduced - rss_full) / q) / (rss_full / (N - p - 1))
print(paste("F-stat:", F_stat))

p_value <- pf(F_stat, df1 = q, df2 = (N - p - 1), lower.tail = FALSE)
print(paste("p-value:", p_value))

# We can use ANOVA for a comparison
anova(reduced_model, full_model)
```

```
[1] "F-stat: 0.781959483788322"
[1] "p-value: 0.517266415865109"
```

A anova: 2 × 6

	Res.Df <dbl>	RSS <dbl>	Df <dbl>	Sum of Sq <dbl>	F <dbl>	Pr(>F) <dbl>
1	24	0.6719537	NA	NA	NA	NA
2	21	0.6044333	3	0.06752034	0.7819595	0.5172664

The 95% confidence intervals for all three covariates include 0, indicating that the effect of the covariates is uncertain. In the individual H_0-tests, their $Pr(>|t|)$-values were relatively high, indicating weak evidence against the null hypothesis $H_0 : \beta_j = 0$. This is also reflected in their low t-values (`t2` being the least likely to exclude here, with a significantly lower P-value and a higher t-value).

When comparing the full model with the reduced model (excluding `t1, t2, bw`), the residual sum of squares (RSS) slightly decreases, meaning that removing these variables, the model performs marginally better. Also, the F-test shows a relatively low F-value ($F = 0.7820$) and a high p-value ($p = 0.5173$). This confirms that the difference between the full and reduced models is not significant.

Both individual and joint hypothesis tests show that `t1, t2, bw` do not provide a significant improvement to the model. This suggests that they can be removed without loss of predictive performance.

d) Forward selection

[9]:
```r
install.packages("leaps")
library(leaps)
```

```
The downloaded binary packages are in
/var/folders/dp/x5lf9lp142l0p7rp61v608h00000gn/T//Rtmpr7jwqa/downloaded_packages
```

[10]:
```r
nvmax = ncol(nuclear.data) - 1
nuclear.fwd <- regsubsets(log(cost) ~ ., data=nuclear.data, nvmax=nvmax,␣
 ↪method="forward")
summary.nuclear.fwd <- summary(nuclear.fwd)
```

```
summary.nuclear.fwd
```

```
Subset selection object
Call: regsubsets.formula(log(cost) ~ ., data = nuclear.data, nvmax = nvmax,
    method = "forward")
10 Variables  (and intercept)
      Forced in Forced out
date      FALSE      FALSE
t1        FALSE      FALSE
t2        FALSE      FALSE
cap       FALSE      FALSE
pr        FALSE      FALSE
ne        FALSE      FALSE
ct        FALSE      FALSE
bw        FALSE      FALSE
cum.n     FALSE      FALSE
pt        FALSE      FALSE
1 subsets of each size up to 10
Selection Algorithm: forward
          date t1  t2  cap pr  ne  ct  bw  cum.n pt
1  ( 1 )  " "  " " " " " " " " " " " " " " " "   "*"
2  ( 1 )  " "  " " " " " "*" " " " " " " " " "   "*"
3  ( 1 )  "*"  " " " " " "*" " " " " " " " " "   "*"
4  ( 1 )  "*"  " " " " " "*" " " "*" " " " " "   "*"
5  ( 1 )  "*"  " " " " " "*" " " "*" "*" " " " "   "*"
6  ( 1 )  "*"  " " " " " "*" " " "*" "*" " " "*"   "*"
7  ( 1 )  "*"  " " " " " "*" " " "*" "*" "*" "*"   "*"
8  ( 1 )  "*"  " " " " " "*" "*" "*" "*" "*" "*"   "*"
9  ( 1 )  "*"  " " "*" "*" "*" "*" "*" "*" "*"   "*"
10  ( 1 ) "*"  "*" "*" "*" "*" "*" "*" "*" "*"   "*"
```

The lower table in the summary indicates that the optimal forward selection was performed in this order: `pt, cap, date, ne, ct, cum.n, bw, pr, t2, t1`

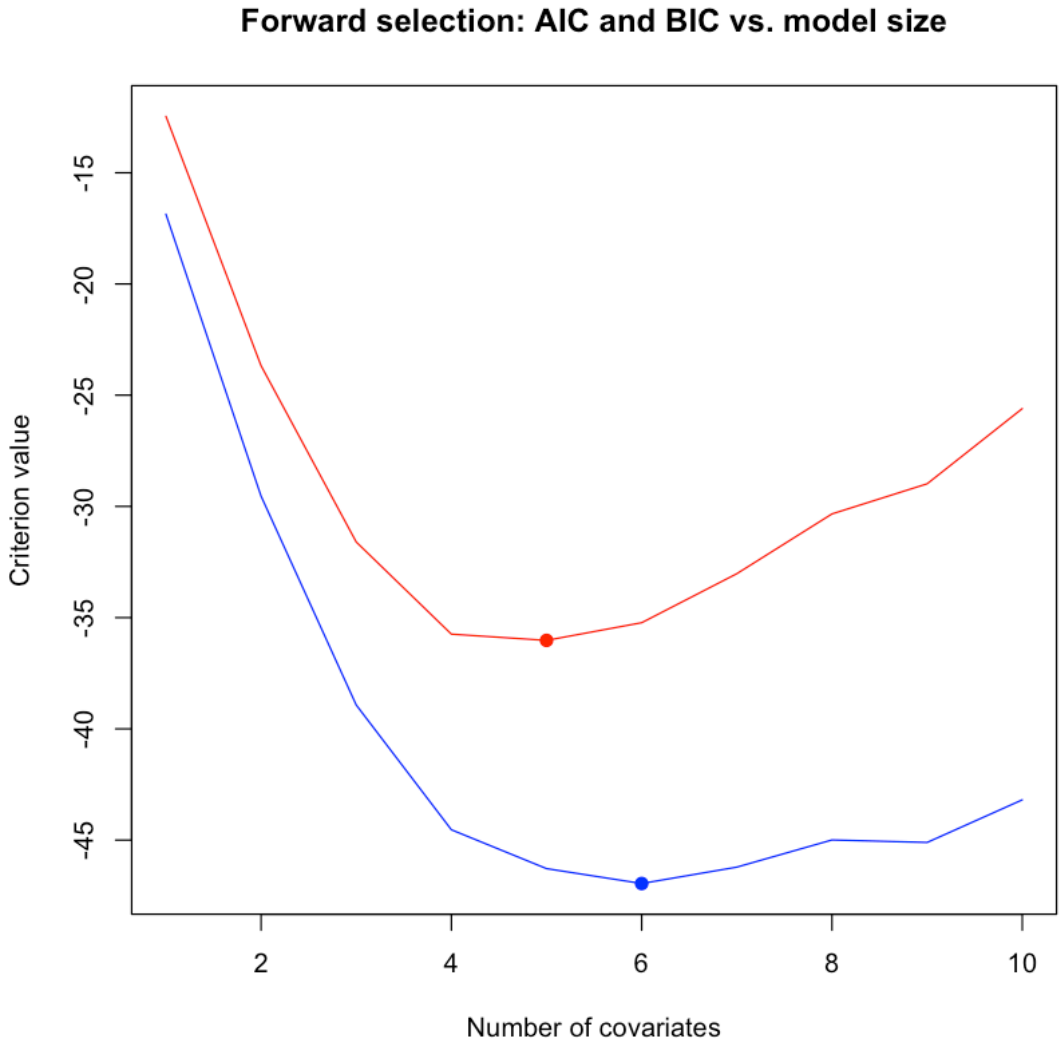
We now want to use **Bayesian Information Criterion (BIC)** and **Akaike Information Criterion (AIC)** to determine the best models. BIC and AIC are obtained with the formulas

$$BIC = -2l(\hat{\theta}) + \log(N)d \qquad AIC = -2l(\hat{\theta}) + 2d$$

where $l(\hat{\theta})$ is the Maximum Likelihood Estimate of θ, N is the number of observations (rows) and d is the number of estimated parameters, including the intercept. BIC can be found in the `summary()`, and we can use this to determine AIC.

[11]:
```
N <- nrow(nuclear.data) # number of observations

# estimating AIC from BIC, where AIC = BIC - log(N)d + 2d
d = (c(1 : nvmax) + 2)
summary.nuclear.fwd$aic = summary.nuclear.fwd$bic - log(N) * d + 2 * d

par(bg="white") # white background when using dark mode ;)
```

```r
matplot(
  cbind(summary.nuclear.fwd$aic, summary.nuclear.fwd$bic),
  type = "l",
  lty = 1,
  col = c("blue", "red"),
  xlab = "Number of covariates",
  ylab = "Criterion value",
  main = "Forward selection: AIC and BIC vs. model size"
)

p.aic = which.min(summary.nuclear.fwd$aic)
p.bic = which.min(summary.nuclear.fwd$bic)

points(c(p.aic, p.bic),
       c(summary.nuclear.fwd$aic[p.aic], summary.nuclear.fwd$bic[p.bic]),
       col=c("blue", "red"), pch=19)
```



[12]:
```
# covariates chosen by AIC and BIC
best_model_fwd_aic <- which.min(summary.nuclear.fwd$aic)
best_model_fwd_bic <- which.min(summary.nuclear.fwd$bic)

# getting variable names
selected_vars_aic <- names(which(summary.nuclear.fwd$which[best_model_fwd_aic,␣
↪]))[-1] # no intercept
selected_vars_bic <- names(which(summary.nuclear.fwd$which[best_model_fwd_bic,␣
↪]))[-1]

cat("Best model in forward selection, AIC:", selected_vars_aic, "\n")
cat("Best model in forward selection, BIC:", selected_vars_bic, "\n")
```

```
Best model in forward selection, AIC: date cap ne ct cum.n pt
Best model in forward selection, BIC: date cap ne ct pt
```

We see here that both AIC and BIC have chosen the covariates `date, cap, ne, pt`, and AIC has furthermore included `cum.n` This is very much in line with the significance star codes in the original `summary()`. Since BIC tends to lean towards fewer covariates, it seems logical that it has omitted `cum.n`, which according to the mentioned summary is the least significant covariate now.

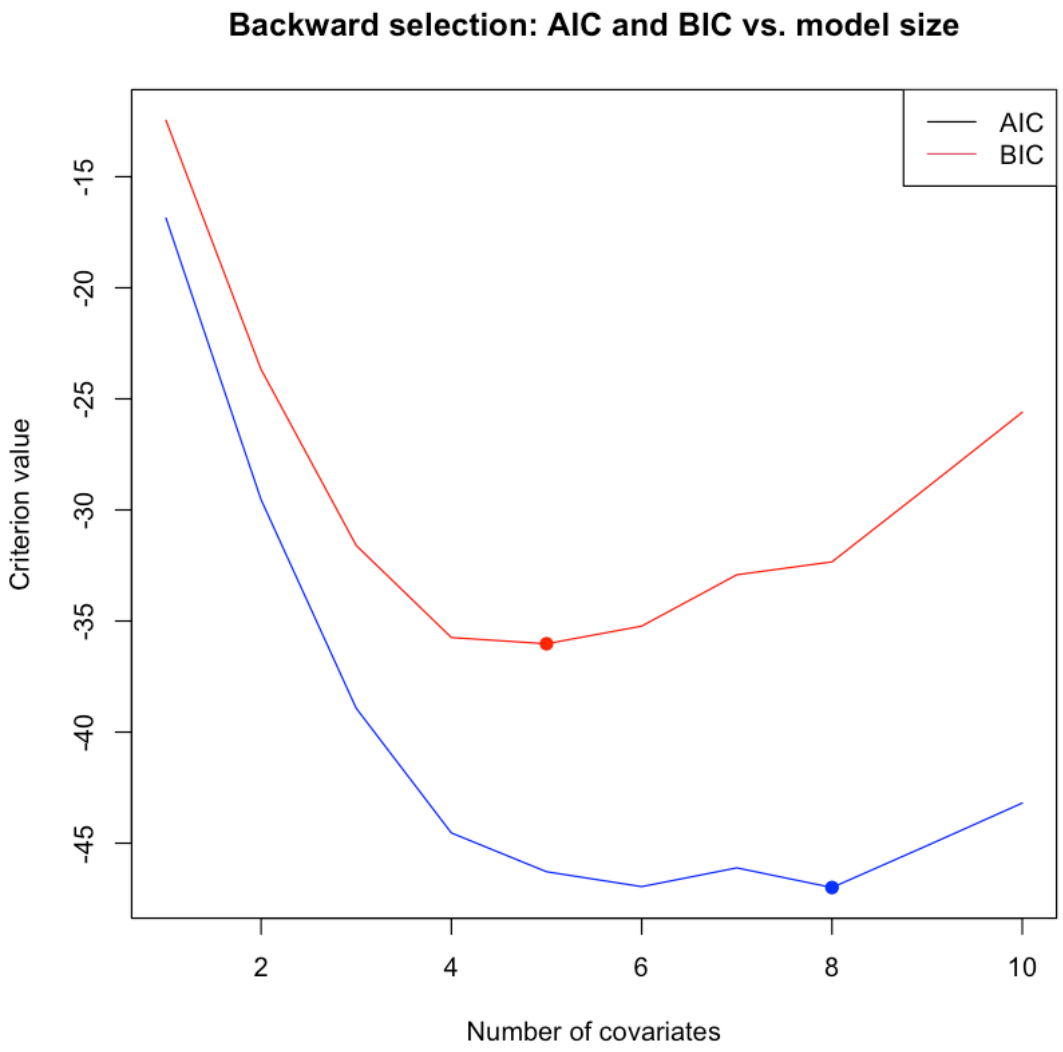
e) Backward selection

[13]:
```
nvmax = ncol(nuclear.data) - 1
nuclear.bwd <- regsubsets(log(cost) ~ ., data=nuclear.data, nvmax=nvmax,␣
 ↪method="backward")
summary.nuclear.bwd <- summary(nuclear.bwd)
summary.nuclear.bwd
```

```
Subset selection object
Call: regsubsets.formula(log(cost) ~ ., data = nuclear.data, nvmax = nvmax,
    method = "backward")
10 Variables  (and intercept)
      Forced in Forced out
date      FALSE      FALSE
t1        FALSE      FALSE
t2        FALSE      FALSE
cap       FALSE      FALSE
pr        FALSE      FALSE
ne        FALSE      FALSE
ct        FALSE      FALSE
bw        FALSE      FALSE
cum.n     FALSE      FALSE
pt        FALSE      FALSE
1 subsets of each size up to 10
Selection Algorithm: backward
          date t1  t2  cap pr  ne  ct  bw  cum.n pt
1  ( 1 )  " "  " " " " " " " " " " " " " " " "   "*"
2  ( 1 )  " "  " " " " " " "*" " " " " " " " " "   "*"
3  ( 1 )  "*"  " " " " " " "*" " " " " " " " " "   "*"
4  ( 1 )  "*"  " " " " " " "*" " " "*" " " " " "   "*"
5  ( 1 )  "*"  " " " " " " "*" " " "*" "*" " " "   "*"
6  ( 1 )  "*"  " " " " " " "*" " " "*" "*" " " "*"   "*"
7  ( 1 )  "*"  " " "*" "*" " " " " "*" "*" " " "*"   "*"
8  ( 1 )  "*"  " " "*" "*" "*" "*" "*" " " " " "*"   "*"
9  ( 1 )  "*"  " " "*" "*" "*" "*" "*" "*" "*" "*"   "*"
10  ( 1 ) "*"  "*" "*" "*" "*" "*" "*" "*" "*" "*"   "*"
```

Backward selection starts with all the covariates, then remove them one by one in the following order: `t1, bw, pr, t2, cum.n, ct, ne, date, cap, pt`

[14]:
```r
N <- nrow(nuclear.data) # number of observations

d = (c(1 : nvmax) + 2)
summary.nuclear.bwd$aic = summary.nuclear.bwd$bic - log(N) * d + 2 * d

par(bg="white")

matplot(
  cbind(summary.nuclear.bwd$aic, summary.nuclear.bwd$bic),
  type = "l",
  lty = 1,
  col = c("blue", "red"),
  xlab = "Number of covariates",
  ylab = "Criterion value",
  main = "Backward selection: AIC and BIC vs. model size"
)

p.aic = which.min(summary.nuclear.bwd$aic)
p.bic = which.min(summary.nuclear.bwd$bic)

points(c(p.aic, p.bic),
       c(summary.nuclear.bwd$aic[p.aic], summary.nuclear.bwd$bic[p.bic]),
       col=c("blue", "red"), pch=19)

legend("topright",c("AIC","BIC"),lty=1,col=1:3,)
```



[15]:
```
best_model_bwd_aic <- which.min(summary.nuclear.bwd$aic)
best_model_bwd_bic <- which.min(summary.nuclear.bwd$bic)

selected_vars_aic <- names(which(summary.nuclear.bwd$which[best_model_bwd_aic,␣
↪]))[-1]
selected_vars_bic <- names(which(summary.nuclear.bwd$which[best_model_bwd_bic,␣
↪]))[-1]

cat("Best model in backward selection, AIC:", selected_vars_aic, "\n")
cat("Best model in backward selection, BIC:", selected_vars_bic, "\n")
```

```
Best model in backward selection, AIC: date t2 cap pr ne ct cum.n pt
Best model in backward selection, BIC: date cap ne ct pt
```

After both forward and backward selection has been performed, and since BIC rendered 5 covariates in both tests (see the summary after each test for covariates), we are left with three "best" models. Interestingly, AIC choose to include `t2` in the backward selection. As discussed in **c)** this *could* be logical, as `t2` has a higher significance than `t1, bw` according to the results from the hypothesis testing, and by visual inspection of the confidence intervals.

f) K-Fold Cross Validation

[16]:
```
Err.Kcv <- rep(0, 4) # number of "best of" models
K <- 10
N <- nrow(nuclear.data) # observations

# we need the covariates from our "best of models"
covariates_fwd_aic <- names(which(summary.nuclear.fwd$which[best_model_fwd_aic,␣
 ↪]))[-1]
covariates_fwd_bic <- names(which(summary.nuclear.fwd$which[best_model_fwd_bic,␣
 ↪]))[-1]
covariates_bwd_aic <- names(which(summary.nuclear.bwd$which[best_model_bwd_aic,␣
 ↪]))[-1]
covariates_bwd_bic <- names(which(summary.nuclear.bwd$which[best_model_bwd_bic,␣
 ↪]))[-1]

models <- list(
  "AIC (forward)" = covariates_fwd_aic,
  "BIC (forward)" = covariates_fwd_bic,
  "AIC (backward)" = covariates_bwd_aic,
  "BIC (backward)" = covariates_bwd_bic
)

index <- sample(rep(1:K, ceiling(N/K))[1:N])

for (p in 1:4) {
  # setting a formula-string
  formula.str <- paste("log(cost) ~", paste(models[[p]], collapse = " + "))

  # accumulating squared errors
  SSE <- 0

  for (k in 1:K) {
    # training the model on everything but the fold k
    nuclear.Kcv <- lm(
      as.formula(formula.str),
      data = nuclear.data,
      subset = (1:N)[-which(index == k)]
    )
    test_inds <- which(index == k)

    for (i in test_inds) {
```

```
      # log scaling
      y_true_log <- log(nuclear.data$cost[i])
      y_pred_log <- predict(nuclear.Kcv, newdata = nuclear.data[i, models[[p]],␣
 ↪drop = FALSE])

      SSE <- SSE + (y_true_log - y_pred_log)^2
    }
  }

  # finding MSE as 1/N*SSE
  Err.Kcv[p] <- SSE / N
}

names(Err.Kcv) <- c("AIC (forward)", "BIC (forward)", "AIC (backward)", "BIC␣
 ↪(backward)")
print(data.frame(Method = names(Err.Kcv), MSE = Err.Kcv))
```

```
                      Method        MSE
AIC (forward)   AIC (forward) 0.03822791
BIC (forward)   BIC (forward) 0.03821806
AIC (backward) AIC (backward) 0.03923557
BIC (backward) BIC (backward) 0.03821806
```

We see that MSE for `BIC (forward)` and `BIC (backward)` are the same, which is expected, since the models are identical.

g) Bootstrap 0.632-model

[17]:
```
B <- 1000
N <- nrow(nuclear.data)

SSE_Train <- rep(0, 4)
SSE_Test  <- rep(0, 4)
nTrain    <- rep(0, 4)
nTest     <- rep(0, 4)

for (b in 1:B) {
  index <- sample(N, replace = TRUE)

  # splitting into training and test data
  temp.train.data <- nuclear.data[index, , drop = FALSE]
  temp.test.data  <- nuclear.data[-index, , drop = FALSE]

  # adding a logCost response variable in training and test sets
  temp.train.data$logCost <- log(temp.train.data$cost)
  temp.test.data$logCost  <- log(temp.test.data$cost)

  for (m in 1:4) {
```

```
    formula.str <- paste("logCost ~", paste(models[[m]], collapse = " + "))

    # fitting model on training data (used suppressWarnings to turn off rank␣
 ↪deficiency warnings)
    model.boot <- lm(as.formula(formula.str), data = temp.train.data)

    y_pred_train <- predict(model.boot, newdata = temp.train.data)
    y_true_train <- temp.train.data$logCost
    SSE_Train[m] <- SSE_Train[m] + sum((y_true_train - y_pred_train)^2)
    nTrain[m]    <- nTrain[m] + nrow(temp.train.data)

    if (nrow(temp.test.data) > 0) {
      y_pred_test <- predict(model.boot, newdata = temp.test.data)
      y_true_test <- temp.test.data$logCost
      SSE_Test[m] <- SSE_Test[m] + sum((y_true_test - y_pred_test)^2)
      nTest[m]    <- nTest[m] + nrow(temp.test.data)
    }
  }
}

# getting MSE
MSE_Train <- SSE_Train / nTrain
MSE_Test  <- SSE_Test  / nTest

# 0.632 bootstrap
MSE_632 <- 0.368 * MSE_Train + 0.632 * MSE_Test

best.model <- which.min(MSE_632)

# printing results
results <- data.frame(
  Model    = names(models),
  MSE_632   = MSE_632
)

results
```

A data.frame: 4 × 2

Model	MSE_632
<chr>	<dbl>
AIC (forward)	0.03659303
BIC (forward)	0.03529443
AIC (backward)	0.03972387
BIC (backward)	0.03529443

Also after bootstrap with $B = 1000$ and using the "0.632-estimator", the model with the best result is the BIC forward or backward models, namely the model with the least covariates. With this method we obtained a slightly lower $MSE = 0.0353$ than with K-Fold Cross Validation ($MSE =$

0.0382).

h) Shrinkage with Ridge

[18]:
```r
# downloading packages
install.packages("glmnet")
library(glmnet)
```

```
The downloaded binary packages are in
/var/folders/dp/x5lf9lp142l0p7rp61v608h00000gn/T//Rtmpr7jwqa/downloaded_packages
```

```
Loading required package: Matrix
```

```
Loaded glmnet 4.1-8
```

[19]:
```r
# Preparing the design matrix (without intercept) and response for glmnet␣
 ↪function
X <- model.matrix(log(cost) ~ ., data = nuclear.data)[,-1]
y <- log(nuclear.data$cost)

K <- 10  # Number of folds
N <- nrow(nuclear.data)

index <- sample(rep(1:K, ceiling(N/K))[1:N])  # Cross-validation indices

# Step 1: Use 10-fold CV to select optimal lambda
lambda.grid <- 10^seq(5, -2, length = 100)
fit.ridge.cv <- cv.glmnet(X, y, lambda = lambda.grid, alpha = 0)

optimal_lambda_ridge <- fit.ridge.cv$lambda.min

cat("Optimal lambda chosen by 10-fold CV for Ridge:", optimal_lambda_ridge,␣
 ↪"\n")

# Step 2: Use 10-fold CV to compute MSE with chosen lambda
MSE_Ridge_Kcv <- 0

for (k in 1:K) {
  # Training and test split
  train_inds <- which(index != k)
  test_inds  <- which(index == k)

  X_train <- X[train_inds, ]
  y_train <- y[train_inds]
  X_test  <- X[test_inds, ]
  y_test  <- y[test_inds]
```

```
  # Fit Ridge model with the optimal lambda from step 1
  fit.ridge <- glmnet(X_train, y_train, lambda = optimal_lambda_ridge, alpha =␣
 ↪0)

  # Predict on test set
  y_pred_ridge <- predict(fit.ridge, newx = X_test)

  # Accumulate squared error
  MSE_Ridge_Kcv <- MSE_Ridge_Kcv + sum((y_test - y_pred_ridge)^2)
}

# Compute final MSE for Ridge
MSE_Ridge_Kcv <- MSE_Ridge_Kcv / N

# Print results
cat("MSE for Ridge Regression (10-fold CV):", MSE_Ridge_Kcv, "\n")
```

```
Optimal lambda chosen by 10-fold CV for Ridge: 0.07054802
MSE for Ridge Regression (10-fold CV): 0.04225129
```

We note that Ridge regression with 10-fold Cross Validation renders a best $MSE = 0.0423$ which is slightly higher than for with the bootstrap 0.632-method ($MSE = 0.0353$).

[20]:
```
# Preparing the design matrix (without intercept) and response for glmnet␣
 ↪function
X <- model.matrix(log(cost) ~ ., data = nuclear.data)[,-1]
y <- log(nuclear.data$cost)

K <- 10  # Number of folds
N <- nrow(nuclear.data)

index <- sample(rep(1:K, ceiling(N/K))[1:N])  # Cross-validation indices

# Step 1: Use 10-fold CV to select optimal lambda for Lasso
lambda.grid <- 10^seq(5, -2, length = 100)
fit.lasso.cv <- cv.glmnet(X, y, lambda = lambda.grid, alpha = 1)

optimal_lambda_lasso <- fit.lasso.cv$lambda.min

cat("Optimal lambda chosen by 10-fold CV for Lasso:", optimal_lambda_lasso,␣
 ↪"\n")

# Step 2: Use 10-fold CV to compute MSE with chosen lambda
MSE_Lasso_Kcv <- 0

for (k in 1:K) {
  # Training and test split
  train_inds <- which(index != k)
```

```r
  test_inds  <- which(index == k)

  X_train <- X[train_inds, ]
  y_train <- y[train_inds]
  X_test  <- X[test_inds, ]
  y_test  <- y[test_inds]

  # Fit Lasso model with the optimal lambda from step 1
  fit.lasso <- glmnet(X_train, y_train, lambda = optimal_lambda_lasso, alpha =␣
 ↪1)

  # Predict on test set
  y_pred_lasso <- predict(fit.lasso, newx = X_test)

  # Accumulate squared error
  MSE_Lasso_Kcv <- MSE_Lasso_Kcv + sum((y_test - y_pred_lasso)^2)
}

# Compute final MSE for Lasso
MSE_Lasso_Kcv <- MSE_Lasso_Kcv / N

# Print results
cat("MSE for Lasso Regression (10-fold CV evaluation):", MSE_Lasso_Kcv, "\n")
```

```
Optimal lambda chosen by 10-fold CV for Lasso: 0.02656088
MSE for Lasso Regression (10-fold CV evaluation): 0.03847974
```

Lasso regression with 10-fold Cross Validation leaves us with a best $MSE = 0.0385$ which is marginally lower than for Ridge, however also slightly higher than for with the bootstrap 0.632-method ($MSE = 0.0353$).