```
library(tidyverse)
library(faraway)
library(HH)
library(leaps)
library(caret)

states = state.x77 %>% # load data from faraway package
  as.data.frame() %>%
  janitor::clean_names()

# table of summary stats
states %>%
  skimr::skim_to_list() %>%
  as.data.frame %>%
  dplyr::select(1, 2, 5:11) %>%
  `colnames<-`(c(' ', 'NA', 'Mean', 'Std. Dev.', 'Min', '1st Q',
'Median', '3rd Q', 'Max')) %>%
  knitr::kable(caption = 'Summary statistics')

# scatterplot to assess correlation between vars
states %>% pairs

# correlation matrix to evaluate what is seen in scatterplots
# states %>%
#   cor

states_analysis = states %>%
  mutate(log_area = log(area),
         log_illiteracy = log(illiteracy),
         log_pop = log(population)) %>%
  dplyr::select(-area, -population, -illiteracy)

par(mfrow = c(2, 3))
hist(states$illiteracy)
hist(states$population)
hist(states$area)
hist(states_analysis$log_illiteracy)
hist(states_analysis$log_pop)
hist(states_analysis$log_area)

## backwards elimination
# summary(lm(life_exp ~ ., data = states_analysis))
# summary(lm(life_exp ~ murder + hs_grad + frost + log_area +
log_illiteracy + log_pop, data = states_analysis))
# summary(lm(life_exp ~ murder + hs_grad + frost + log_illiteracy +
log_pop, data = states_analysis))

b.fit = lm(life_exp ~ murder + hs_grad + frost + log_pop, data =
states_analysis)
summary(b.fit)
```

```
## forwards process
# summary(lm(life_exp ~ murder, data = states_analysis))
# summary(lm(life_exp ~ hs_grad, data = states_analysis))
# summary(lm(life_exp ~ frost, data = states_analysis))
# summary(lm(life_exp ~ log_area, data = states_analysis))
# summary(lm(life_exp ~ log_illiteracy, data = states_analysis))
# summary(lm(life_exp ~ log_pop, data = states_analysis))
#
# # murder has lowest p-val. Start adding secondary vars
# summary(lm(life_exp ~ murder + hs_grad, data = states_analysis))
# summary(lm(life_exp ~ murder + frost, data = states_analysis))
# summary(lm(life_exp ~ murder + log_area, data = states_analysis))
# summary(lm(life_exp ~ murder + log_illiteracy, data =
states_analysis))
# summary(lm(life_exp ~ murder + log_pop, data = states_analysis))
#
# # murder + hs_grad
# summary(lm(life_exp ~ murder + hs_grad + frost, data =
states_analysis))
# summary(lm(life_exp ~ murder + hs_grad + log_area, data =
states_analysis))
# summary(lm(life_exp ~ murder + hs_grad + log_illiteracy, data =
states_analysis))
# summary(lm(life_exp ~ murder + hs_grad + log_pop, data =
states_analysis))
#
# # murder + hs_grad + log_pop
# summary(lm(life_exp ~ murder + hs_grad + log_pop + frost, data =
states_analysis))
# summary(lm(life_exp ~ murder + hs_grad + log_pop + log_area, data =
states_analysis))
# summary(lm(life_exp ~ murder + hs_grad + log_pop + log_illiteracy,
data = states_analysis))
#
# # murder + hs_grad + log_pop + frost
# summary(lm(life_exp ~ murder + hs_grad + log_pop + frost + log_area,
data = states_analysis))
# summary(lm(life_exp ~ murder + hs_grad + log_pop + frost +
log_illiteracy, data = states_analysis))

f.fit = lm(life_exp ~ murder + hs_grad + log_pop + frost, data =
states_analysis)
summary(f.fit)

## Stepwise
step.fit = step(lm(life_exp ~ ., data = states_analysis))

# function to select the 'best' model
best <- function(model, ...)
```

```
{
  subsets <- regsubsets(formula(model), model.frame(model), ...)
  subsets <- with(summary(subsets),
                  cbind(p = as.numeric(rownames(which)), which, rss,
rsq, adjr2, cp, bic))

  return(subsets)
}

best(lm(life_exp ~ ., data = states_analysis)) %>%
  knitr::kable(., 'latex', caption = 'Criterion-based model building')
%>%
  kableExtra::kable_styling(latex_options = c("hold_position")) %>%
  kableExtra::landscape()

# leaps::leaps(x = states_analysis[, c(1, 3:8)], y =
states_analysis$life_exp, nbest = 2, method = "Cp")

# leaps::leaps(x = states_analysis[, c(1, 3:8)], y =
states_analysis$life_exp, nbest = 2, method = "adjr2")

# Summary of models for each size (one model per size)
b = leaps::regsubsets(life_exp ~ ., data = states_analysis)
rs = summary(b)

# Plots of Cp and Adj-R2 as functions of parameters
par(mar = c(4, 4, 1, 1))
par(mfrow = c(1, 2))

plot(1:7, rs$cp, xlab = "No of parameters", ylab = "Cp Statistic")
abline(0, 1)
plot(1:7, rs$adjr2, xlab = "No of parameters", ylab = "Adj R2")

life_exp_fit = b.fit

# rstandard function gives the INTERNALLY studentized residuals

stu_res = rstandard(life_exp_fit)
outliers_y = stu_res[abs(stu_res) > 2.5]

# Measures of influence:
# Gives DFFITS, Cook's Distance, Hat diagonal elements, and others.

# influence.measures(life_exp_fit)

# Look at the Cook's distance lines / influential point output and
notice obs 11 as potential Y outlier / influential point

par(mfrow = c(2, 2))
plot(life_exp_fit)
```

```
# Examine results with and without observations 5 and 28 that have
very high survivals (>2000)
fit_nooutlier = lm(life_exp ~ murder + hs_grad + log_pop + frost, data
= states_analysis[-11, ])
summary(fit_nooutlier) # look at the results of the fitted model
without the influential point

plot(fit_nooutlier)

## 10-fold CV
kfold_cv = lapply(1:10, function(i){
  # create 10-fold training datasets
  data_train <- trainControl(method = "cv", number = 10)

  # Fit the model used above
  model_caret <- train((life_exp ~ murder + hs_grad + log_pop +
frost),
                       data = states_analysis,
                       trControl = data_train,
                       method = 'lm',
                       na.action = na.pass)

  #return(list(model_caret$results, model_caret$resample))
  return(model_caret$results)
})

do.call("rbind", kfold_cv) %>%
  dplyr::select(RMSE, RMSESD) %>% # summarise(mse = mean(RMSE))
  mutate(MSE = RMSE^2,
         std.error = RMSESD / 9) %>%
  dplyr::select(1, 3, 2, 4) # %>% summarise(se = mean(std.error))

do.call("rbind", kfold_cv) %>%
  dplyr::select(RMSE, RMSESD) %>% # summarise(mse = mean(RMSE))
  mutate(MSE = RMSE^2,
         std.error = RMSESD / 9) %>%
  dplyr::select(1, 3, 2, 4) %>% summarise(mse = mean(MSE),
                                          se = mean(std.error))

## Bootstrap
set.seed(1)

# Perform a regression model with the original sample; calculate
predicted values and residuals.
states_analysis = states_analysis %>%
  modelr::add_predictions(life_exp_fit) %>% # add predicted
birthweight
  modelr::add_residuals(life_exp_fit) %>% # residual of observed bwt -
predicted bwt
```

```
  rename('pred1' = pred)


# function to bootstrap residuals and regress new predictions
boot.res <- function(data, index){
  data = data %>%
    rowwise %>%
    mutate(rand_res = sample(resid, replace = T, size = 1), # Randomly
resample the residuals (with replacement), but leave the X values and
predicted values unchanged.
           boot_y = pred1 + rand_res) %>% # New observations by adding
the original predicted values to the bootstrap residuals
    modelr::add_predictions(lm(boot_y ~ murder + hs_grad + log_pop +
frost, data = .)) %>%
    mutate(sq = (boot_y - pred)^2)

  mse = (1/(nrow(data)) * sum(data$sq)) # calculate mse
  root.mse = sqrt(mse) # rmse
  return(root.mse)
}

broom::tidy(boot::boot(states_analysis, boot.res, 10)) %>%
  rename('RMSE' = statistic) %>%
  mutate(MSE = RMSE^2)
broom::tidy(boot::boot(states_analysis, boot.res, 1000)) %>%
  rename('RMSE' = statistic) %>%
  mutate(MSE = RMSE^2)
```