Practical 4

Emilia Löscher

2022-09-26

First, load the packages:

```
library(ISLR)
library(MASS)
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
## v ggplot2 3.3.6      v purrr   0.3.5
## v tibble  3.1.8      v dplyr   1.0.10
## v tidyr   1.2.1      v stringr 1.4.1
## v readr   2.1.3      v forcats 0.5.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## x dplyr::select() masks MASS::select()
```

```
library(magrittr)
```

```
##
## Attache Paket: 'magrittr'
##
## Das folgende Objekt ist maskiert 'package:purrr':
##
##     set_names
##
## Das folgende Objekt ist maskiert 'package:tidyr':
##
##     extract
```

1.

Create a linear model object called lm_ses using the formula medv ~ lstat and the Boston dataset.

```
lm_ses <- Boston %>% lm(medv ~ lstat, data = .)
```

2.

Use the function coef() to extract the intercept and slope from the lm_ses object. Interpret the slope coefficient.

```
coef(lm_ses)
```

```
## (Intercept)       lstat
##  34.5538409  -0.9500494
```

With every percent point increase in lower status of the population, the median value of owner-occupied homes in $1000s decreases by 0.95.

3.

Use summary() to get a summary of the lm_ses object. What do you see? You can use the help file ?summary.lm.

```
summary(lm_ses)
```

```
##
## Call:
## lm(formula = medv ~ lstat, data = .)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -15.168  -3.990  -1.318   2.034  24.500
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 34.55384    0.56263   61.41   <2e-16 ***
## lstat       -0.95005    0.03873  -24.53   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.216 on 504 degrees of freedom
## Multiple R-squared:  0.5441, Adjusted R-squared:  0.5432
## F-statistic: 601.6 on 1 and 504 DF,  p-value: < 2.2e-16
```

The summary function shows the estimates of the parameters, their standard error and significance. Furthermore, the residual standard error, R^2, and F-statistic are given.

4.

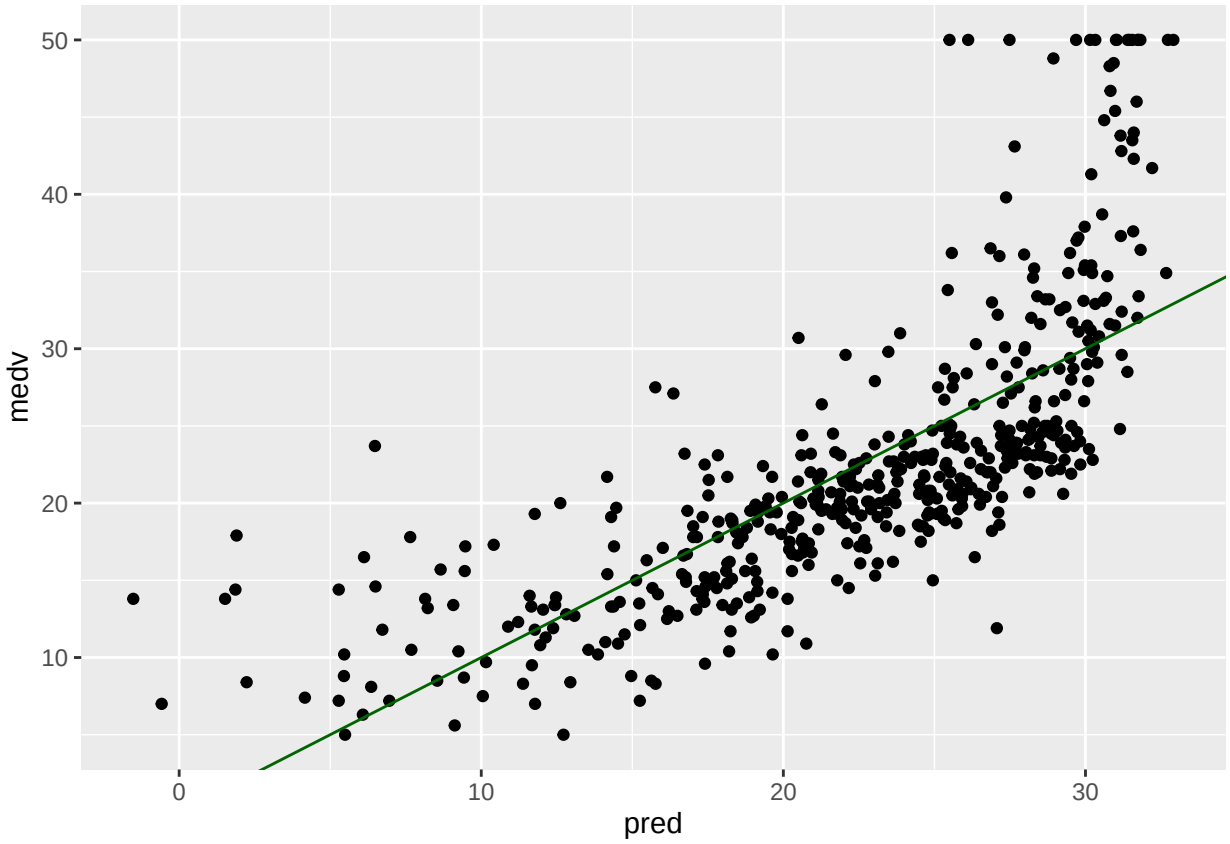
Save the predicted y values to a variable called y_pred

```
y_pred <- predict(lm_ses)
```

5.

Create a scatter plot with y_pred mapped to the x position and the true y value (Boston$medv) mapped to the y value. What do you see? What would this plot look like if the fit were perfect?

```
Boston %>% mutate(pred=  y_pred) %>%
  ggplot(aes(x = pred, y= medv)) +
  geom_point()+
  geom_abline(slope = 1, col = "darkgreen")
```



If the predictions matched the observed values perfectly, they would lie on the line (bisecting angle). As they do not, we can conclude that there are some deviations of the predicted values from the observed values.

6.

Use the seq() function to generate a sequence of 1000 equally spaced values from 0 to 40. Store this vector in a data frame with (data.frame() or tibble()) as its column name lstat. Name the data frame pred_dat.

```
pred_dat <- data.frame(lstat = seq(0, 40, length.out= 1000))
```

7.

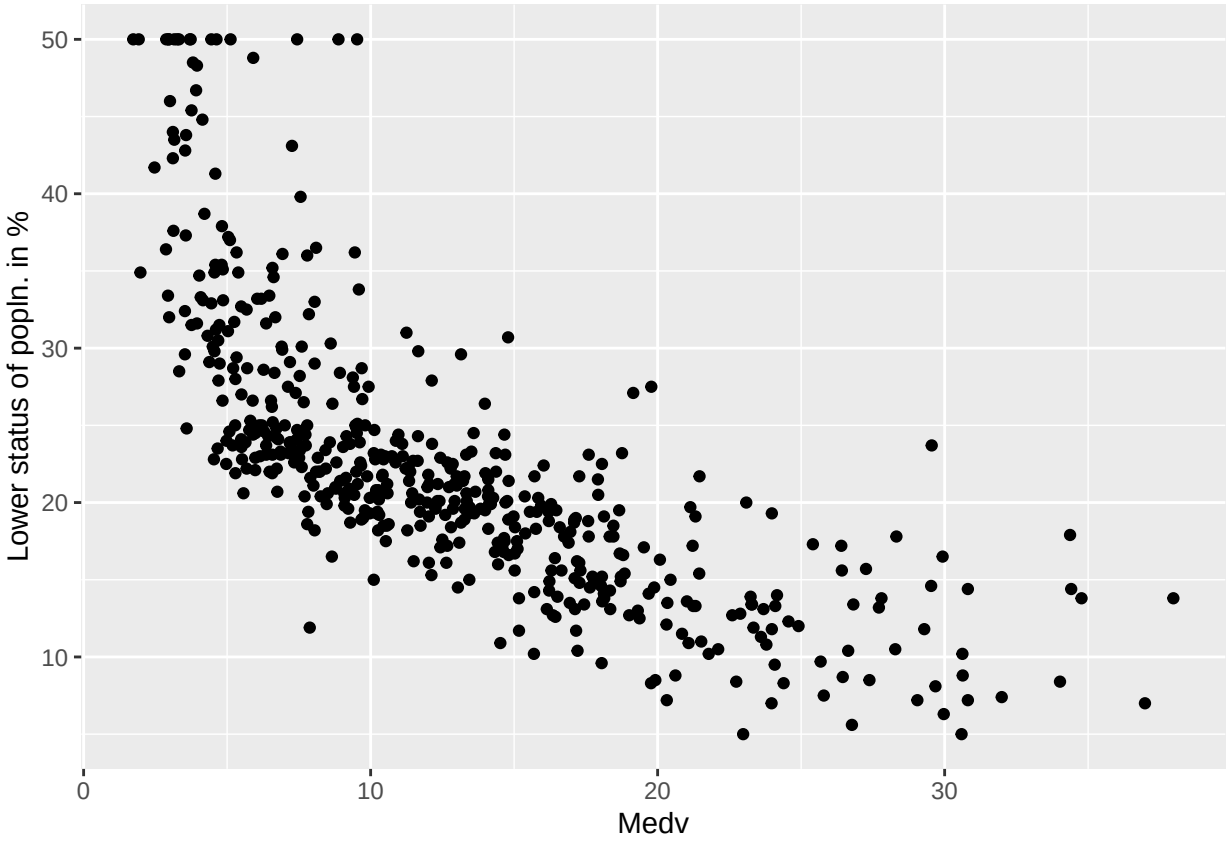
Use the newly created data frame as the newdata argument to a predict() call for lm_ses. Store it in a variable named y_pred_new.

```
y_pred_new <- predict(lm_ses, newdata = pred_dat)
```

8.

Create a scatter plot from the Boston dataset with lstat mapped to the x position and medv mapped to the y position. Store the plot in an object called p_scatter.

```
p_scatter <- Boston %>%
  ggplot(aes(x= lstat, y = medv))+
  geom_point()+
  labs(x= "Medv", y= "Lower status of popln. in %")
p_scatter
```



9.

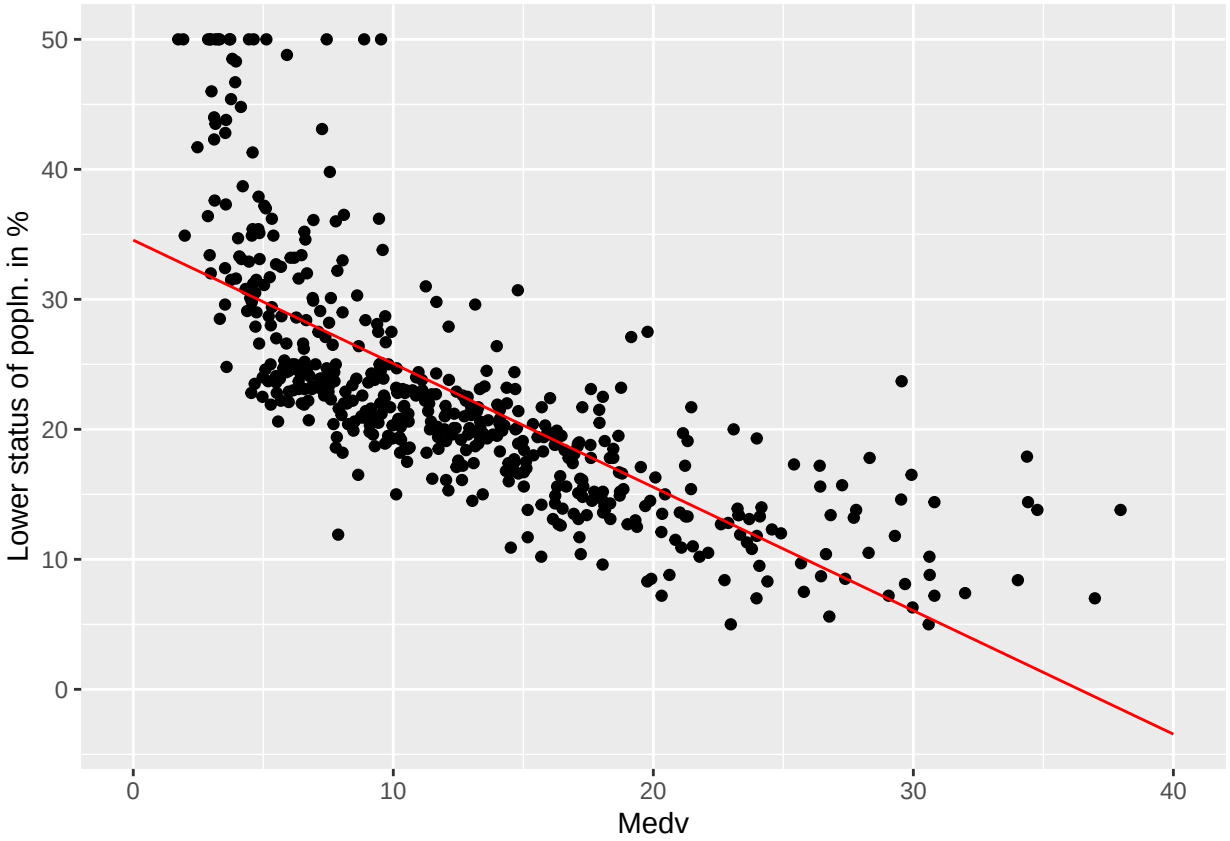
Add the vector y_pred_new to the pred_dat data frame with the name medv.

```
pred_dat %<>%  mutate(medv = y_pred_new)
```

10.

Add a geom_line() to p_scatter, with pred_dat as the data argument. What does this line represent?

```
p_scatter +
  geom_line(data = pred_dat, col = "red")
```



The line represents the predictions for the sequence of 1000 values between 0 and 40 as saved in pred_dat.

11.

The interval argument can be used to generate confidence or prediction intervals. Create a new object called y_pred_95 using predict() (again with the pred_dat data) with the interval argument set to "confidence". What is in this object?

```
y_pred_95 <- predict(lm_ses, newdata = pred_dat, interval = "confidence")
```

The object contains 3 columns. The first is the prediction, the second contains the lower confidence interval and the third column contains the upper confidence interval. By default it is a 95%-confidence interval.

12.

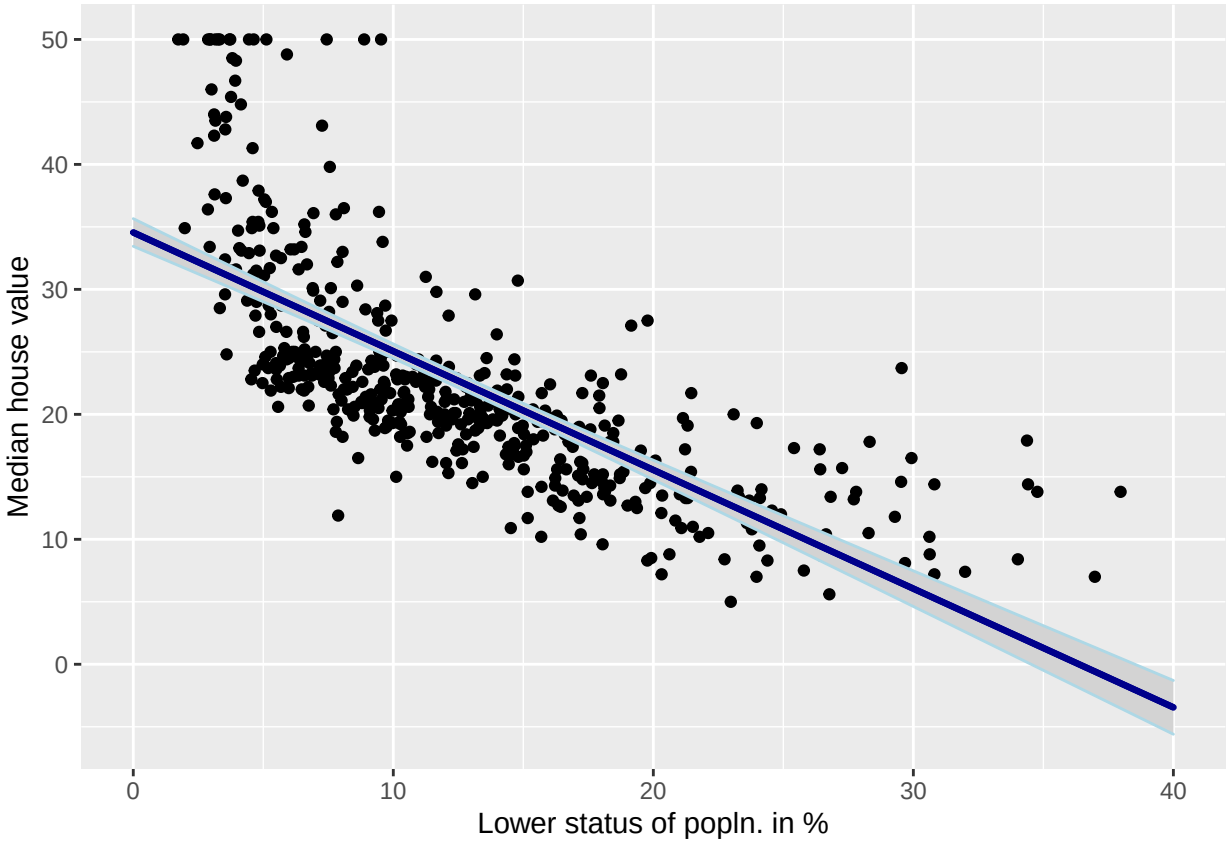
Create a data frame with 4 columns: medv, lstat, lower, and upper.

```
pred_dat %<>% mutate(lower= y_pred_95[,2], upper= y_pred_95[,3])
```

13.

Add a geom_ribbon() to the plot with the data frame you just made. The ribbon geom requires three aesthetics: x (lstat, already mapped), ymin (lower), and ymax (upper). Add the ribbon below the geom_line() and the geom_points() of before to make sure those remain visible. Give it a nice colour and clean up the plot, too!

```
p_intv <-
  p_scatter +
  geom_ribbon(aes(ymin = lower, ymax = upper), col = "lightblue", outline.type = "both", fill = "lightg
  geom_point(size= 0.5, col ="darkblue", data = pred_dat)+
  labs(y= "Median house value", x= "Lower status of popln. in %")
p_intv
```



14.

Explain in your own words what the ribbon represents.

The ribbon represents the predictions between 0 and 40. the lightblue lines represent the lower and upper 95% confidence interval boundaries. All points in the lightgrey area fall into that 95% confidence interval.

15.

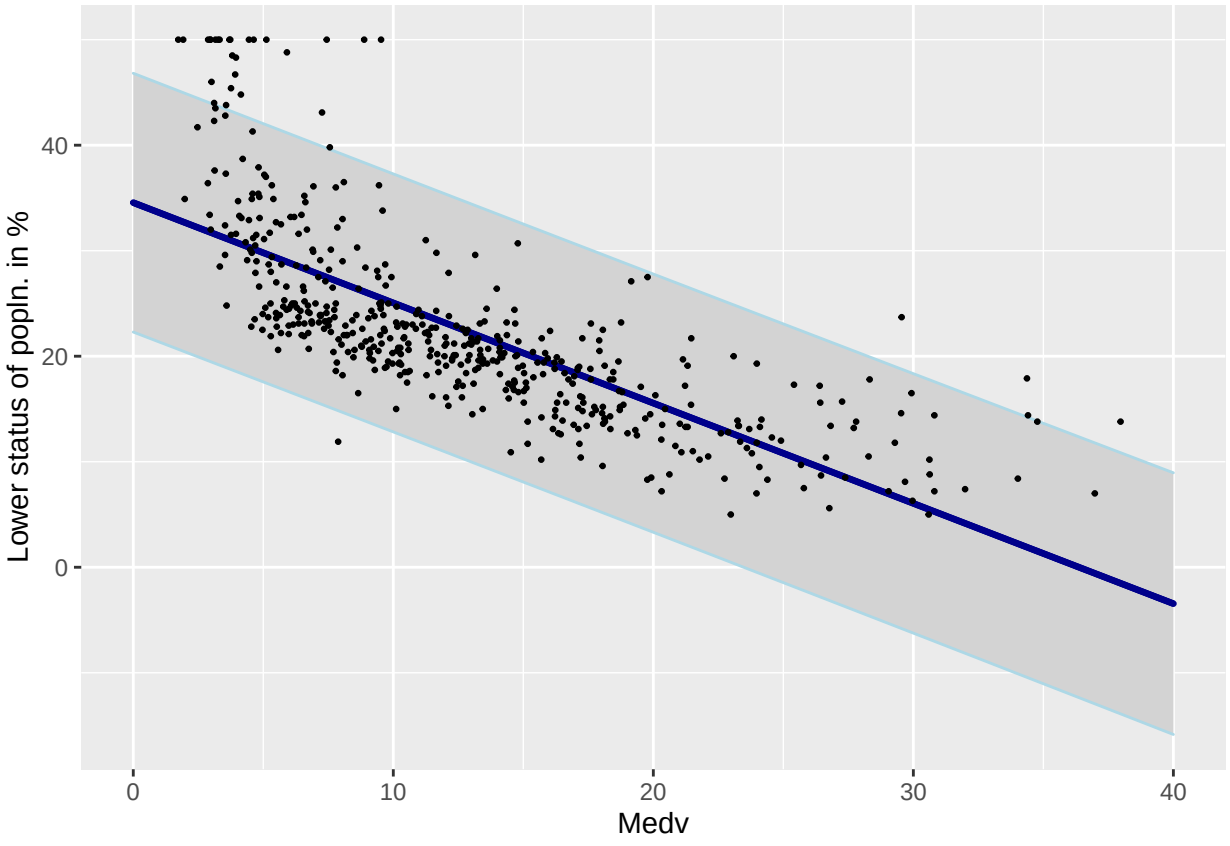
Do the same thing, but now with the prediction interval instead of the confidence interval.

```
y_pred_pI <- predict(lm_ses, newdata = pred_dat, interval = "predict")

pred_dat %<>% mutate(lowerP= y_pred_pI[,2], upperP= y_pred_pI[,3])

p_predInt <- Boston %>%
  ggplot(aes(x= lstat, y = medv))+
   geom_ribbon(aes(ymin = lowerP, ymax = upperP), col = "lightblue", outline.type = "both", fill = "lig
  geom_point(size= 0.5, col ="darkblue", data = pred_dat)+
  geom_point(size= 0.5)+
  labs(x= "Medv", y= "Lower status of popln. in %")

p_predInt
```



16.

Write a function called mse() that takes in two vectors: true y values and predicted y values, and which outputs the mean square error.

```
mse <- function(y_true, y_pred) {
  MSE <- sum((y_true - y_pred)^2)/length(y_true)
  return(MSE)
}
```

17.

Make sure your mse() function works correctly by running the following code.

```
mse(1:10, 10:1)
```

```
## [1] 33
```

18.

Calculate the mean square error of the lm_ses model. Use the medv column as y_true and use the predict() method to generate y_pred.

```
#We already obtained y_pred above.
mse(Boston$medv, y_pred)
```

```
## [1] 38.48297
```

19.

The Boston dataset has 506 observations. Use c() and rep() to create a vector with 253 times the word "train", 152 times the word "validation", and 101 times the word "test". Call this vector splits.

```
splits <- c(rep("train", 253), rep("validation", 152), rep("test", 101))
```

20.

Use the function sample() to randomly order this vector and add it to the Boston dataset using mutate(). Assign the newly created dataset to a variable called boston_master.

```
set.seed(0310)
Boston %<>% mutate(boston_master = sample(splits, size = length(splits), replace = FALSE))
```

21.

Now use filter() to create a training, validation, and test set from the boston_master data. Call these datasets boston_train, boston_valid, and boston_test.

```
boston_train <- Boston %>% filter(boston_master == "train")
boston_valid <- Boston %>% filter(boston_master == "validation")
boston_test <- Boston %>% filter(boston_master == "test")
```

22.

Train a linear regression model called model_1 using the training dataset. Use the formula medv ~ lstat like in the first lm() exercise. Use summary() to check that this object is as you expect.

```
model_1 <- boston_train %>% lm(medv ~ lstat, dat = .)
summary(model_1)
```

```
##
## Call:
## lm(formula = medv ~ lstat, data = .)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -10.447  -4.353  -1.615   2.290  23.416
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 34.83000    0.81758   42.60   <2e-16 ***
## lstat       -0.92859    0.05611  -16.55   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.583 on 251 degrees of freedom
## Multiple R-squared:  0.5218, Adjusted R-squared:  0.5199
## F-statistic: 273.8 on 1 and 251 DF,  p-value: < 2.2e-16
```

23.

Calculate the MSE with this object. Save this value as model_1_mse_train.

```
model_1_mse <- mse(boston_train$medv, predict(model_1))
model_1_mse
```

```
## [1] 42.9883
```

24.

Now calculate the MSE on the validation set and assign it to variable model_1_mse_valid. Hint: use the newdata argument in predict().

```
model_1_mse_valid <- mse(boston_valid$medv, predict(model_1, newdata = boston_valid))
model_1_mse_valid
```

```
## [1] 43.70795
```

25.

Create a second model model_2 for the train data which includes age and tax as predictors. Calculate the train and validation MSE.

```
model_2 <- boston_train %>% lm(medv ~ lstat + age + tax, .)

model_2_mse <- mse(boston_train$medv, predict(model_2))
model_2_mse
```

```
## [1] 41.30532
```

```
model_2_mse_valid <- mse(boston_valid$medv, predict(model_2, newdata = boston_valid))
model_2_mse_valid
```

```
## [1] 42.06791
```

26.

Compare model 1 and model 2 in terms of their training and validation MSE. Which would you choose and why?

I would choose the second model, as the training MSE as well as the validation MSE are slightly lower than the corresponding MSE for model 1.

27

Calculate the test MSE for the model of your choice in the previous question. What does this number tell you?

```
model_2_mse_test <- mse(boston_test$medv, predict(model_2, newdata = boston_test))
model_2_mse_test
```

```
## [1] 20.1568
```

When predicting `medv`on new data, 20.16 is the expected amount of error when using the second model.

28.

Create a function that performs k-fold cross-validation for linear models.

```
my_cross_Val <- function(formula = medv ~ lstat + age + tax, dataset = Boston, k = 9, seed= 0310){
  #setting a seed for sampling which observation falls into which fold
  set.seed(seed)
  #adding the respective fold to the dataset
  dataset %<>% mutate(fold = sample(rep(rep(1:k), length.out = nrow(dataset)), nrow(dataset), replace =

  #empty vector to save the mse for each fold
  mse_fold <- numeric(k)

  #fitting the model for every fold and calculating the mse and saving it in mse_fold
  for(i in 1:k){
   data_train <- dataset %>% filter(fold != i)
   data_val <- dataset %>% filter(fold == i)
   mod <- lm(formula, data = data_train)
   mse_fold[i] <- mse(data_val[[as.character(formula)[2]]], predict(mod, newdata = data_val))
  }
  av_mse <- mean(mse_fold)
  return(av_mse)
}
```

29.

Use your function to perform 9-fold cross validation with a linear model with as its formula medv ~ lstat + age + tax. Compare it to a model with as formulat medv ~ lstat + I(lstat^2) + age + tax.

```
my_cross_Val()
```

```
## [1] 37.4886
```

```
my_cross_Val(formula = medv ~ lstat + I(lstat^2) + age + tax)
```

```
## [1] 27.84697
```

The second model has a lower MSE, hence, I would opt for using the slightly more complex model.