Practical 4

Alex Carriero (9028757)

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

```
# Libraries
library(ISLR)
library(MASS)
library(tidyverse)
```

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

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
```

```
# b0: socio-economic status is zero, the median house value is 35.553
# b1: each one unit increase in lstat is associated with a 0.95 unit drop in median house value.
```

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 = Boston)
##
## 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 ***
```

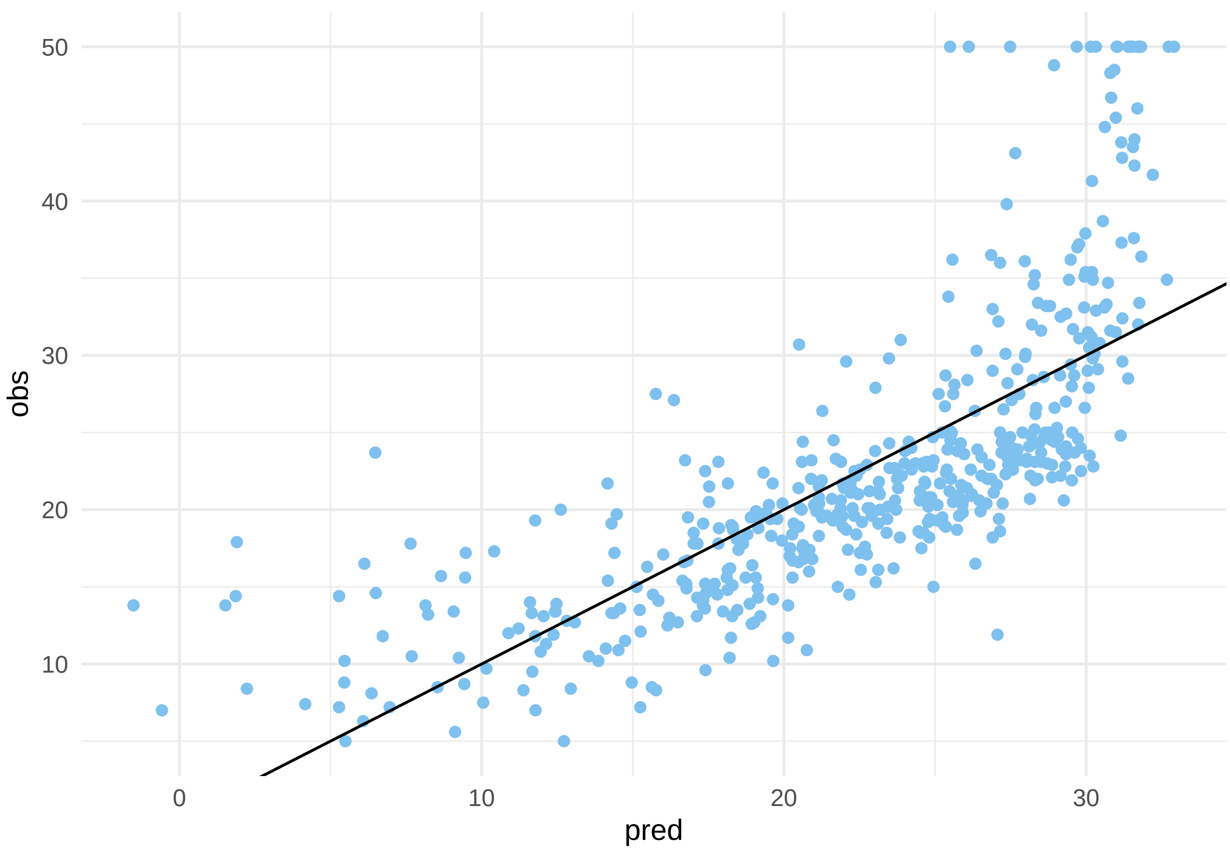
```
## lstat       -0.95005    0.03873  -24.53   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.216 on 504 degrees of freedom
## Multiple R-squared:  0.5441, Adjusted R-squared:  0.5432
## F-statistic: 601.6 on 1 and 504 DF,  p-value: < 2.2e-16
```

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?

```
tibble(pred = y_pred,
       obs  = Boston$medv) %>%
       ggplot(aes( x = pred, y = obs)) +
       geom_point( color = "skyblue2")+
       theme_minimal() +
       geom_abline(slope = 1)
```



If the predicted values mapped exactly to the observed values, I would expect to see most of the points fall along the like y = x.

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 <- tibble(lstat = seq(0,40, length.out = 1000))
```

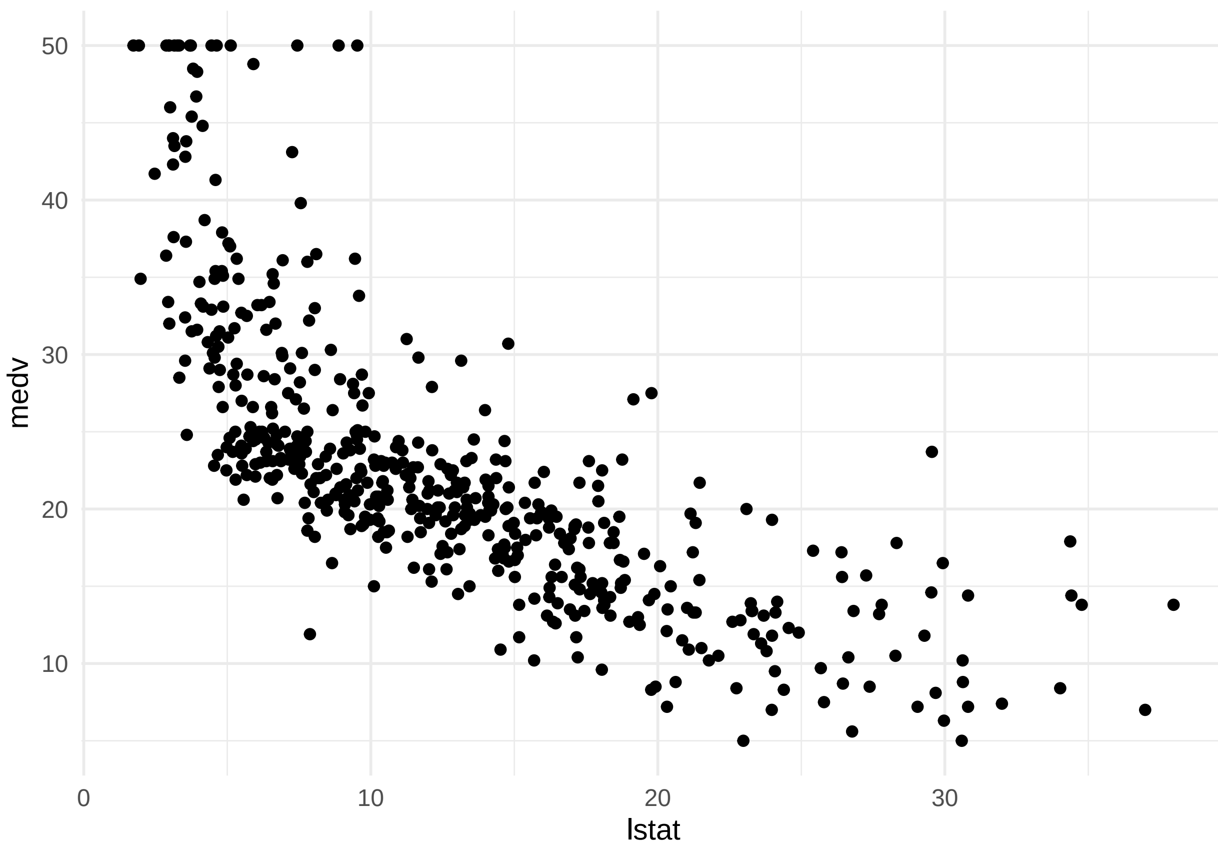
7. Use the newly created data frame as the newdata argument to a predict() call for lm_ses. Store it in a variable named y_pred_new.

```
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() +
             theme_minimal()

p_scatter
```

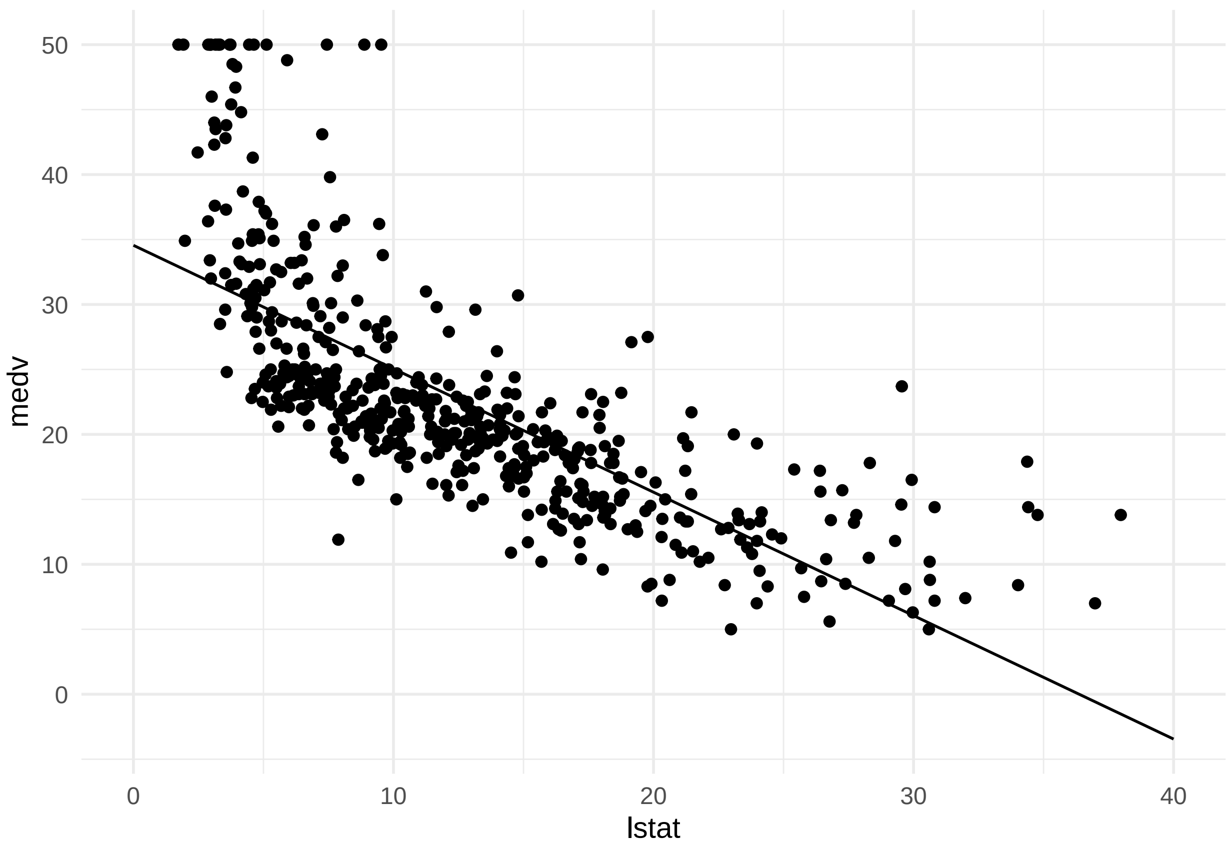


9. Add the vector y_pred_new to the pred_dat data frame with the name medv.

```
pred_dat <- 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)
```




```
# This line represents predicted values of medv for the values of lstat
```

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")
head(y_pred_95)
```

```
##        fit      lwr      upr
## 1 34.55384 33.44846 35.65922
```

```
## 2 34.51580 33.41307 35.61853
## 3 34.47776 33.37768 35.57784
## 4 34.43972 33.34229 35.53715
## 5 34.40168 33.30690 35.49646
## 6 34.36364 33.27150 35.45578
```

```
# it's a matrix with an estimate and a lower and an upper confidence interval.
```

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

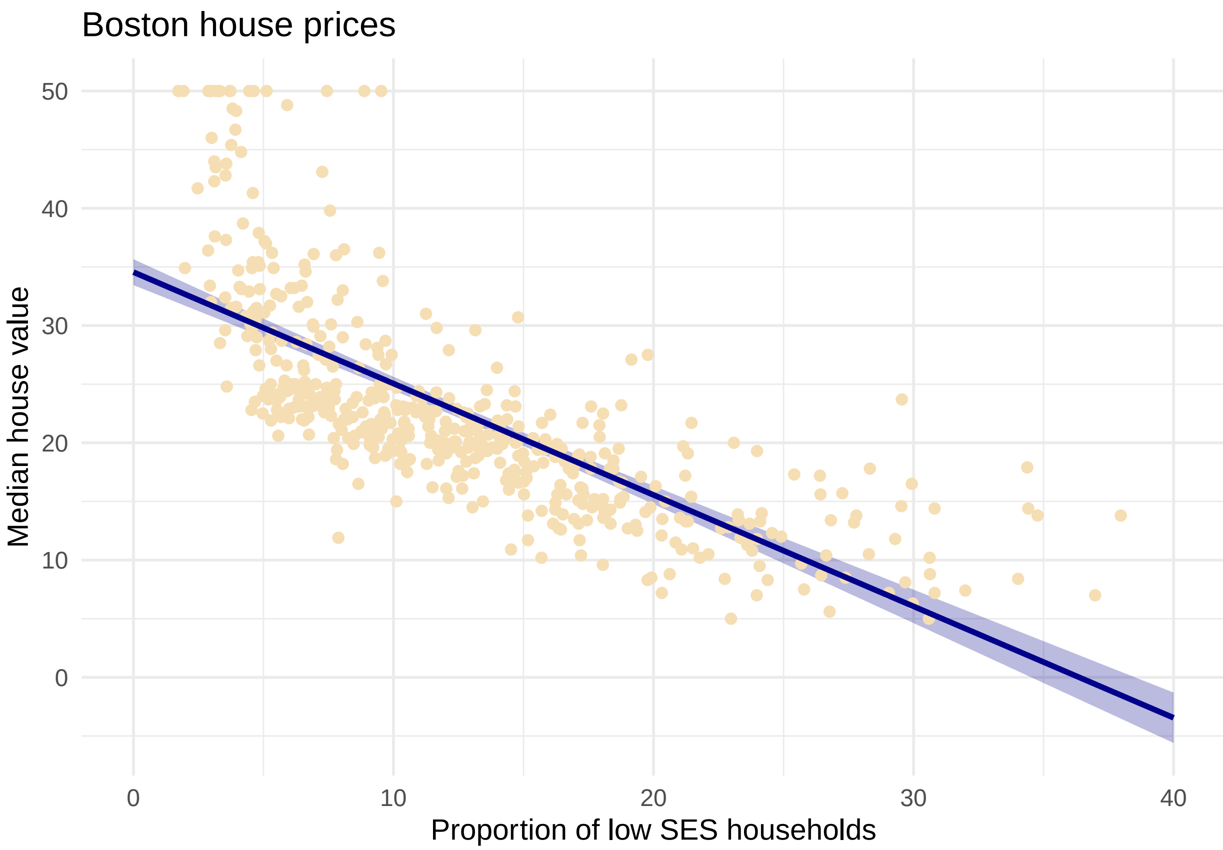
```
gg_pred <- tibble(
            lstat = pred_dat$lstat,
            medv  = y_pred_95[, 1],
            lower = y_pred_95[, 2],
            upper = y_pred_95[, 3])

gg_pred
```

```
## # A tibble: 1,000 x 4
##     lstat  medv lower upper
##     <dbl> <dbl> <dbl> <dbl>
##  1 0       34.6  33.4  35.7
##  2 0.0400  34.5  33.4  35.6
##  3 0.0801  34.5  33.4  35.6
##  4 0.120   34.4  33.3  35.5
##  5 0.160   34.4  33.3  35.5
##  6 0.200   34.4  33.3  35.5
##  7 0.240   34.3  33.2  35.4
##  8 0.280   34.3  33.2  35.4
##  9 0.320   34.2  33.2  35.3
## 10 0.360   34.2  33.1  35.3
## # ... with 990 more rows
```

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!

```
Boston %>%
  ggplot(aes(x = lstat, y = medv)) +
  geom_ribbon(aes(ymin = lower, ymax = upper), data = gg_pred, fill = "#00008b44")+
  geom_point(color = "wheat")+
  geom_line(data = pred_dat, color = "#00008b", size = 1)+
  theme_minimal() +
    labs(x     = "Proportion of low SES households",
         y     = "Median house value",
        title = "Boston house prices")
```



14. Explain in your own words what the ribbon represents.

```
# The ribbon represent the 95% confidence interval associated with the fit line

# Upon repeated sampling of data from the same population, at least 95% of the ribbons with contain
```

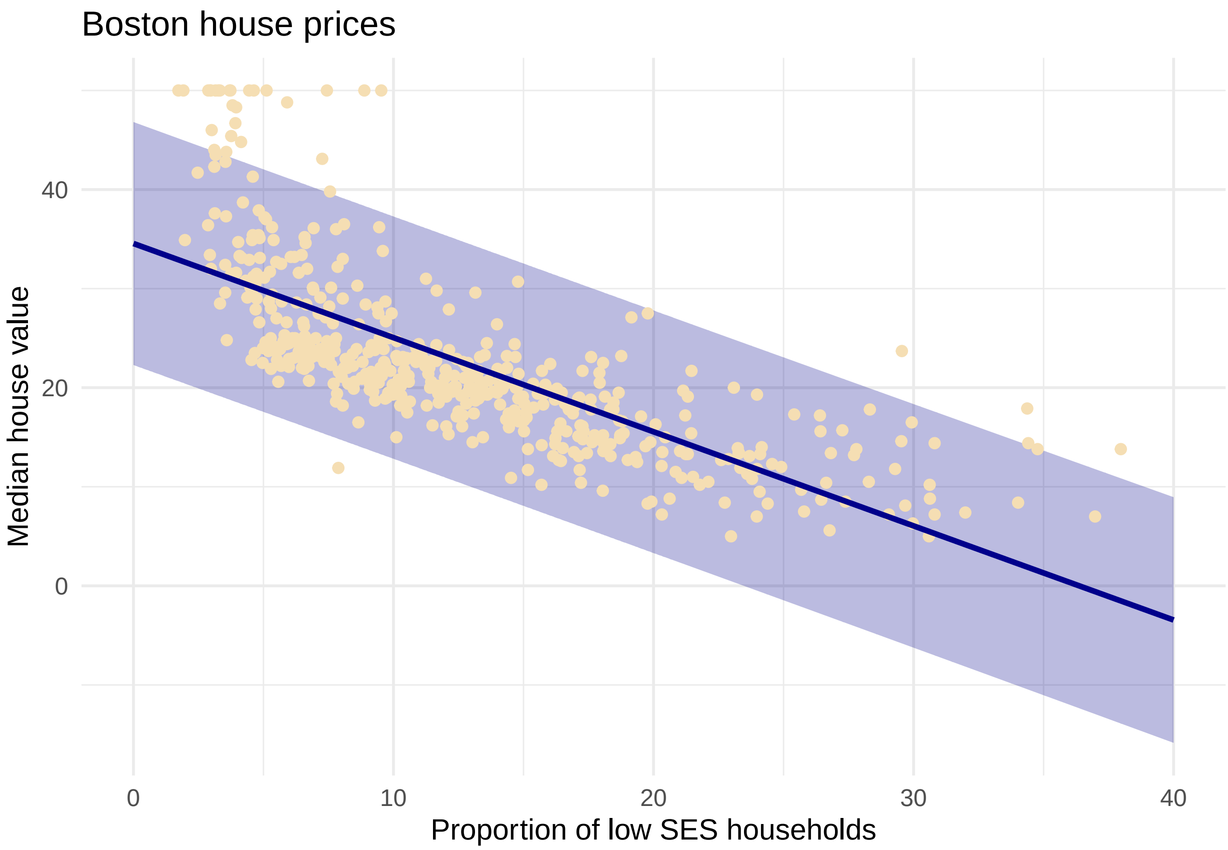
15. Do the same thing, but now with the prediction interval instead of the confidence interval.

```
# pred with pred interval
y_pred_95 <- predict(lm_ses, newdata = pred_dat, interval = "prediction")


# create the df
gg_pred <- tibble(
  lstat = pred_dat$lstat,
  medv  = y_pred_95[, 1],
  l95   = y_pred_95[, 2],
  u95   = y_pred_95[, 3]
)

# Create the plot
Boston %>%
  ggplot(aes(x = lstat, y = medv)) +
  geom_ribbon(aes(ymin = l95, ymax = u95), data = gg_pred, fill = "#00008b44") +
  geom_point(colour = "wheat") +
  geom_line(data = pred_dat, colour = "#00008b", size = 1) +
```

```
  theme_minimal() +
  labs(x     = "Proportion of low SES households",
       y     = "Median house value",
       title = "Boston house prices")
```



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){
  mean((y_true - y_pred)^2)
}
```

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.

```
y_true <- Boston$medv
y_pred <- predict(lm_ses)

mse(y_true, 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.

```
boston_master <- Boston %>% mutate(splits = sample(splits))
```

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_master %>% filter(splits == "train")
boston_valid <- boston_master %>% filter(splits == "validation")
boston_test  <- boston_master %>% filter(splits == "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 <- lm(medv ~ lstat, data = boston_train)
summary(model_1)
```

```
##
## Call:
## lm(formula = medv ~ lstat, data = boston_train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -14.973  -3.837  -1.392   1.512  24.650
##
## Coefficients:
```

```
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  34.1468     0.8141   41.94   <2e-16 ***
## lstat        -0.9230     0.0568  -16.25   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.263 on 251 degrees of freedom
## Multiple R-squared:  0.5127, Adjusted R-squared:  0.5107
## F-statistic: 264.1 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_train <- mse(y_true = boston_train$medv, y_pred = predict(model_1))
```

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(y_true = boston_valid$medv,
                         y_pred = predict(model_1, newdata = boston_valid))
```

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 <- lm(medv ~ lstat + age + tax, data = boston_train)
model_2_mse_train <- mse(y_true = boston_train$medv, y_pred = predict(model_2))
model_2_mse_valid <- mse(y_true = boston_valid$medv,
                         y_pred = predict(model_2, newdata = boston_valid))
```

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

```
model_1_mse_train
```

```
## [1] 38.9139
```

```
model_1_mse_valid
```

```
## [1] 46.28173
```

```
model_2_mse_train
```

```
## [1] 37.79505
```

```
model_2_mse_valid
```

```
## [1] 43.72235
```

```
# If you are interested in out-of-sample prediction, the
# answer may depend on the random sampling of the rows in the
# dataset splitting: everyond has a different split. However, it
# is likely that model_2 has both lower training and validation MSE.
```

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(y_true = boston_test$medv,
                        y_pred = predict(model_2, newdata = boston_test))
```

```
# The estimate for the expected amount of error when predicting
# the median value of a not previously seen town in Boston when
# using this model is:

sqrt(model_2_mse_test)
```

```
## [1] 4.985098
```

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

```
# Just for reference, here is the mse() function once more
mse <- function(y_true, y_pred) mean((y_true - y_pred)^2)

cv_lm <- function(formula, dataset, k) {
  # We can do some error checking before starting the function
  stopifnot(is_formula(formula))       # formula must be a formula
  stopifnot(is.data.frame(dataset))    # dataset must be data frame
  stopifnot(is.integer(as.integer(k))) # k must be convertible to int

  # first, add a selection column to the dataset as before
  n_samples  <- nrow(dataset)
  select_vec <- rep(1:k, length.out = n_samples)
  data_split <- dataset %>% mutate(folds = sample(select_vec))

  # initialise an output vector of k mse values, which we
  # will fill by using a _for loop_ going over each fold
  mses <- rep(0, k)

  # start the for loop
  for (i in 1:k) {
    # split the data in train and validation set
    data_train <- data_split %>% filter(folds != i)
    data_valid <- data_split %>% filter(folds == i)
```

```
    # calculate the model on this data
    model_i <- lm(formula = formula, data = data_train)

    # Extract the y column name from the formula
    y_column_name <- as.character(formula)[2]

    # calculate the mean square error and assign it to mses
    mses[i] <- mse(y_true = data_valid[[y_column_name]],
                   y_pred = predict(model_i, newdata = data_valid))
  }

  # now we have a vector of k mse values. All we need is to
  # return the mean mse!
  mean(mses)
}
```

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.

```
cv_lm(formula = medv ~ lstat + age + tax, dataset = Boston, k = 9)
```

```
## [1] 38.25751
```

```
cv_lm(formula = medv ~ lstat + I(lstat^2) + age + tax, dataset = Boston, k = 9)
```

```
## [1] 28.1648
```