Model accuracy and fit

In this lab, you will learn how to plot a linear regression with confidence and prediction intervals, and various tools to assess model fit: calculating the MSE, making train-test splits, and writing a function for cross validation. You can download the student zip including all needed files for practical 3 here (https://surfdrive.surf.nl/files/index.php/s/BcQtVPSdH7bCYZA).

Note: the completed homework (the whole practical) has to be **handed in** on Black Board and will be **graded** (pass/fail, counting towards your grade for the individual assignment). **The deadline Monday 13th of May, end of day**. Hand-in should be a **PDF** file. If you know how to knit pdf files, you can hand in the knitted pdf file. However, if you have not done this before, you are advised to knit to a html file as specified below, and within the html browser, 'print' your file as a pdf file.

Please note that not all questions are dependent to each other; if you do not know the answer of one question, you can still solve part of the remaining lab.

We will use the `Boston` dataset, which is in the `MASS` package that comes with `R`. In addition, we will make use of the `caret` package to divide the `Boston` dataset into a training, test, and validation set.

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

Introduction

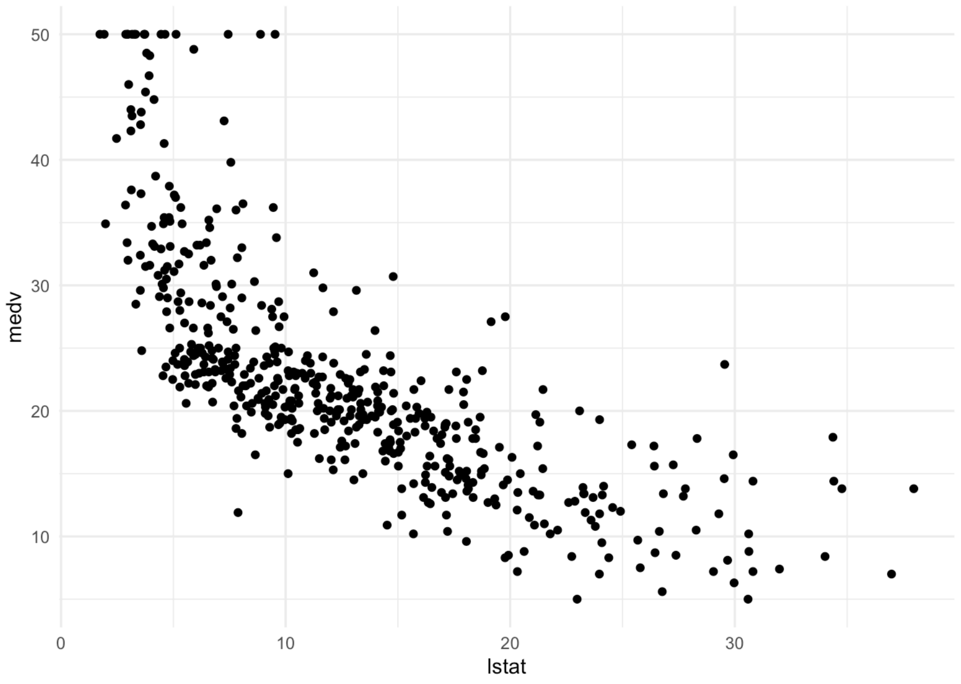
As always, the first thing we want to do is to inspect the Boston dataset using the `View()` function:

```
view(Boston)
```

The `Boston` dataset contains the housing values and other information about Boston suburbs. We will use the dataset to predict housing value (the variable `medv` - Median value of owner-occupied homes in $1000's - is the outcome/dependent variable) by socio-economic status (the variable `lstat` - % lower status of the population - is the predictor / independent variable).

Let's explore socio-economic status and housing value in the dataset using visualization. Use the following code to create a scatter plot from the `Boston` dataset with `lstat` mapped to the x position and `medv` mapped to the y position. We can store the plot in an object called `p_scatter`.

```
p_scatter <-
  Boston %>%
  ggplot(aes(x = lstat, y = medv)) +
  geom_point() +
  theme_minimal()

p_scatter
```



In the scatter plot, we can see that that the median value of owner-occupied homes in $1000's (medv) is going down as the percentage of the lower status of the population (lstat) increases.

Plotting observed data including a prediction line

We'll start with making and visualizing the linear model. As you know, a linear model is fitted in `R` using the function `lm()`, which then returns a `lm` object. We are going to walk through the construction of a plot with a fit line. Then we will add confidence intervals from an `lm` object to this plot.

First, we will create the linear model. This model will be used to predict outcomes for the current data set, and - further along in this lab - to create new data to enable adding the prediction line to the scatter plot.

---

1. **Use the following code to create a linear model object called `lm_ses` using the formula `medv ~ lstat` and the `Boston` dataset.**

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

You have now trained a regression model with `medv` (housing value) as the outcome/dependent variable and `lstat` (socio-economic status) as the predictor / independent variable.

Remember that a regression estimates β_0 (the intercept) and β_1 (the slope) in the following equation:

$$y = \beta_0 + \beta_1 \cdot x_1 + \epsilon$$

---

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

```
coefficients <- coef(lm_ses)
intercept <- coefficients[1]
slope <- coefficients[2]

print(intercept)
```

```
## (Intercept)
##    34.55384
```

```
print(slope)
```

```
##      lstat
## -0.9500494
```

```
#The slope coefficient (β1) = -0.9500494 represents the ch
ange in the median value of homes for each one-unit increa
se in the percentage of lower status population. A negativ
e slope would indicate that as the lstat increases, the me
dv tends to decrease, suggesting that higher socio-economi
c challenges are associated with lower housing values.
```

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

```
model_summary <- summary(lm_ses)

print(model_summary)
```

```
##
## 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 freedo
m
## Multiple R-squared:  0.5441, Adjusted R-squared:  0.543
2
## F-statistic: 601.6 on 1 and 504 DF,  p-value: < 2.2e-16
```

```
#The output from summary(lm_ses) that will help us underst
and the strength and significance of the relationship betw
een medv and lstat, and how well the model fits the data.
```

We now have a model object `lm_ses` that represents the formula

$$\text{medv}_i = 34.55 - 0.95 * \text{lstat}_i + \epsilon_i$$

With this object, we can predict a new `medv` value by inputting its `lstat` value. The `predict()` method enables us to do this for the `lstat` values in the original dataset.

4. **Save the predicted y values to a variable called `y_pred`. Use the `predict` function**

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

To generate a nice, smooth prediction line, we need a large range of (equally spaced) hypothetical `lstat` values. Next, we can use these hypothetical `lstat` values to generate predicted housing values with `predict()` method using the `newdat` argument.

One method to generate these hypothetical values is through using the function `seq()`. This function from `base R` generates a sequence of number using a standardized method. Typically length of the requested sequence divided by the range between `from` to `to`. For more information call `?seq`.

Use the following code and the `seq()` function to generate a sequence of 1000 equally spaced values from 0 to 40. We will store this vector in a data frame with (`data.frame()` or `tibble()`) as its column name `lstat`. We will name the data frame `pred_dat`.

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

Now we will use the newly created data frame as the `newdata` argument to a `predict()` call for `lm_ses` and we will store it in a variable named `y_pred_new`.

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

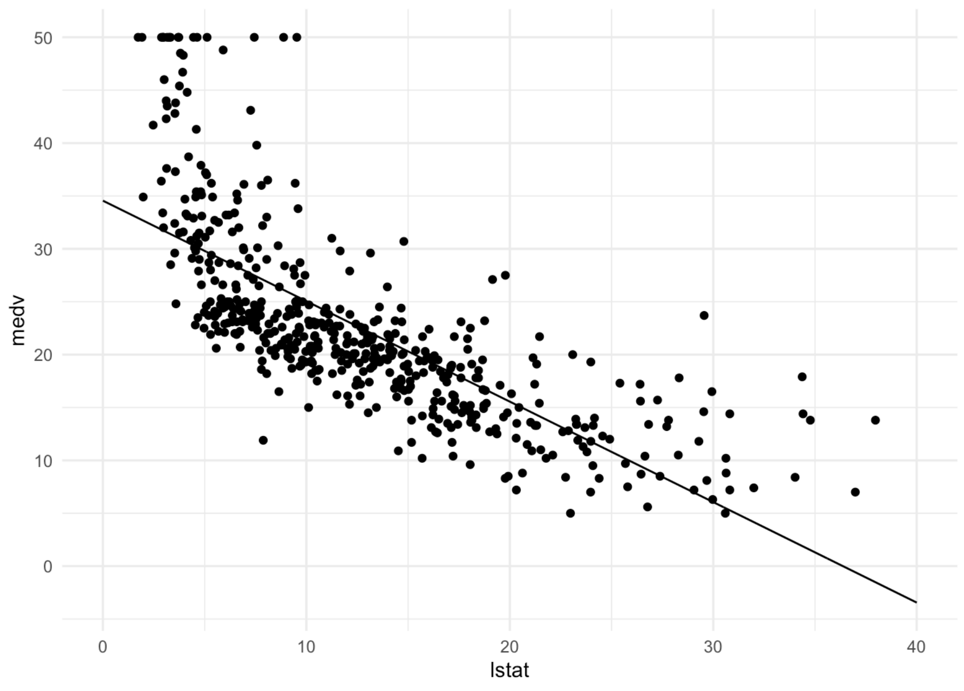
Now, we'll continue with the plotting part by adding a prediction line to the plot we previously constructed.

Use the following code to add the vector `y_pred_new` to the `pred_dat` data frame with the name `medv`.

```
# this can be done in several ways. Here are two possibili
ties:
# pred_dat$medv <- y_pred_new
pred_dat <- pred_dat %>% mutate(medv = y_pred_new)
```

Finally, we will 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 va
lues of lstat
```

Plotting linear regression with confindence intervals

We will continue with the `Boston` dataset, the created model `lm_ses` that predicts `medv` (housing value) by `lstat` (socio-economic status), and the predicted housing values stored in `y_pred` . Now, we will - step by step - add confidence intervals to our plotted prediction line.

---

5. **The `interval` argument can be used to generate confidence or prediction intervals. Create a new object called `y_conf_95` using `predict()` (again with the `pred_dat` data) with the `interval` argument set to "confidence". What is in this object?**

```
y_conf_95 <- predict(lm_ses, newdata = pred_dat, interval
= "confidence")

head(y_conf_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
```

```
#The fitted values (predicted values of medv for each obse
rvation in the dataset).

#The lower and upper bounds of the 95% confidence interval
s for these predictions.
#This gives an estimate of where the true mean response is
expected to lie with 95% confidence.
```

---

6. **Using the data from Question 5 (`y_conf_95`), and the sequence created earlier; create a data frame with 4 columns: `medv` , `lstat` , `lower` , and `upper` . For the `lstat` use the `pred_dat$lstat` ; for `medv` , `lower` , and `upper` use data from `y_conf_95`**

```
results_df <- data.frame(
  lstat = pred_dat$lstat,                    # Use lstat fro
m the Boston dataset
  medv = y_conf_95[, "fit"],              # Predicted medv v
alues from y_conf_95
  lower = y_conf_95[, "lwr"],             # Lower bound of t
he confidence interval
  upper = y_conf_95[, "upr"]               # Upper bound of
the confidence interval
)



head(results_df)
```

```
##        lstat     medv    lower    upper
## 1 0.00000000 34.55384 33.44846 35.65922
## 2 0.04004004 34.51580 33.41307 35.61853
## 3 0.08008008 34.47776 33.37768 35.57784
## 4 0.12012012 34.43972 33.34229 35.53715
## 5 0.16016016 34.40168 33.30690 35.49646
## 6 0.20020020 34.36364 33.27150 35.45578
```

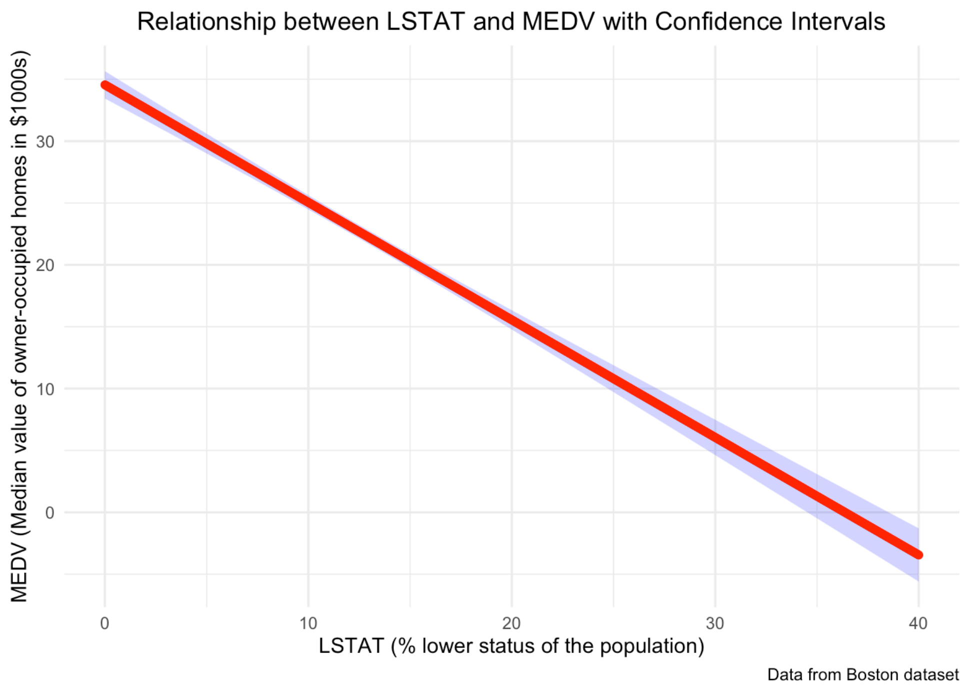
7. **Add a `geom_ribbon()` to the plot with the data frame you just made. The ribbon geom requires three aesthetics: `x` (`lstat` , already mapped), `ymin` (`lower`), and `ymax` (`upper`). Add the ribbon below the `geom_line()` and the `geom_points()` of before to make sure those remain visible. Give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

```
library(ggplot2)
library(gridExtra)
```

```
##
## Attaching package: 'gridExtra'
```

```
## The following object is masked from 'package:dplyr':
##
##     combine
```

```
library(dplyr)

# Assuming results_df is derived correctly and includes al
l necessary columns
p <- ggplot(data = results_df, aes(x = lstat, y = medv)) +
  geom_ribbon(aes(ymin = lower, ymax = upper), fill = "blu
e", alpha = 0.2) +  # Add geom_ribbon for confidence inter
vals
  geom_line(color = "darkblue") +  # Add the line for the
predictions
  geom_point(aes(y = medv), color = "red") +  # Add actual
data points using the same medv from results_df
  labs(
    title = "Relationship between LSTAT and MEDV with Conf
idence Intervals",
    x = "LSTAT (% lower status of the population)",
    y = "MEDV (Median value of owner-occupied homes in $10
00s)",
    caption = "Data from Boston dataset"
  ) +
  theme_minimal() +  # Use a minimal theme
  theme(
    plot.title = element_text(hjust = 0.5),  # Center the
plot title
    legend.position = "none"  # Hide the legend, as it's c
lear from the context
  )

# Print the plot
print(p)
```

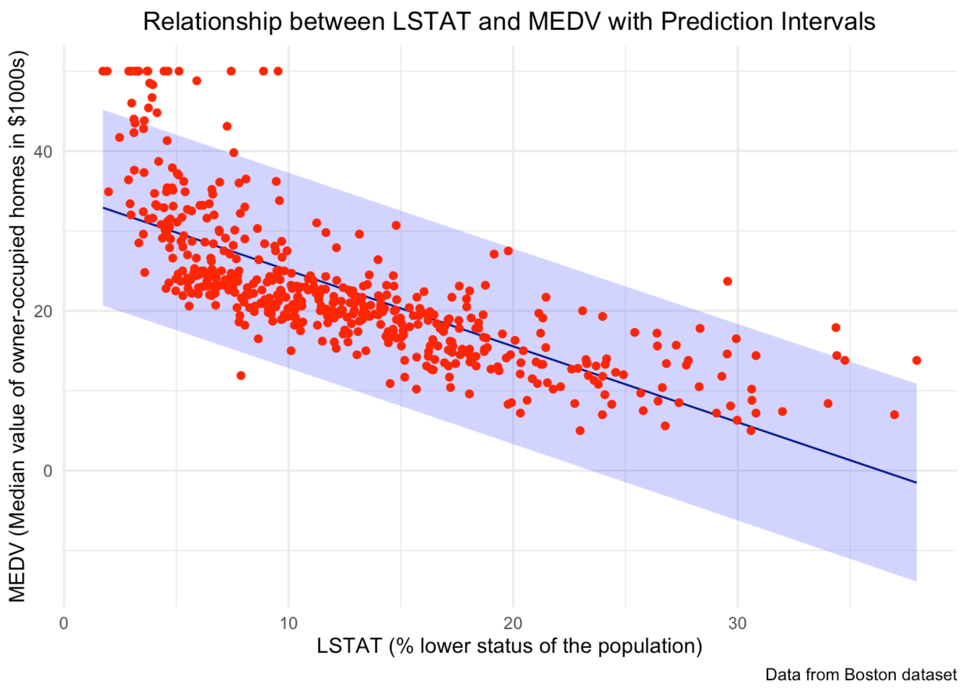


8. **Explain in your own words what the ribbon represents.**

```
# Confidence Intervals: This shaded ribbon shows where we
expect the actual values to lie, with a certain level of c
onfidence, usually 95%. It means we are 95% sure that the
true values are within this ribbon.

#Width of the Ribbon: Where the ribbon is wider, our predi
ctions are less certain, and where it's narrower, our pred
ictions are more confident.

#Understanding Uncertainty: The ribbon helps us see how su
re we can be about the predictions at different levels of
lstat, the socio-economic status indicator. If the ribbon
is broad at certain points, it means there's more uncertai
nty in predicting house values at those points.
```

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

```
y_pred_int <- predict(lm_ses, newdata = Boston, interval =
"prediction")



pred_results_df <- data.frame(
  lstat = Boston$lstat,                   # Predictor
  medv = y_pred_int[, "fit"],             # Predicted valu
es
  lower = y_pred_int[, "lwr"],            # Lower bound of
the prediction interval
  upper = y_pred_int[, "upr"]             # Upper bound of
the prediction interval
)



p_pred <- ggplot(pred_results_df, aes(x = lstat, y = med
v)) +
  geom_ribbon(aes(ymin = lower, ymax = upper), fill = "blu
e", alpha = 0.2) +  # Add geom_ribbon for prediction inter
vals
  geom_line(color = "darkblue") +  # Add the line for the
predictions
  geom_point(aes(y = Boston$medv), color = "red") +  # Add
actual data points
  labs(
    title = "Relationship between LSTAT and MEDV with Pred
iction Intervals",
    x = "LSTAT (% lower status of the population)",
    y = "MEDV (Median value of owner-occupied homes in $10
00s)",
    caption = "Data from Boston dataset"
  ) +
  theme_minimal() +  # Use a minimal theme
  theme(
    plot.title = element_text(hjust = 0.5),  # Center the
plot title
    legend.position = "none"  # Hide the legend
  )

# Print the plot
print(p_pred)
```



Model fit using the mean square error

Next, we will write a function to assess the model fit using the mean square error: the square of how much our predictions on average differ from the observed values. Functions are “self contained” modules of code that accomplish a specific task. Functions usually “take in” data, process it, and “return” a result. If you are not familiar with functions, please read more on chapter 19 of the book “R for Data Science” here (https://r4ds.had.co.nz/functions.html)

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

Start like so:

```
mse <- function(y_true, y_pred) {
  # your formula here
}
```

Wikipedia (https://en.wikipedia.org/w/index.php?title=Mean_squared_error&oldid=857685443) may help for the formula.

```
mse <- function(y_true, y_pred) {
  # Calculate the differences between actual and predicted
values
  residuals <- y_true - y_pred

  # Square the residuals
  squared_residuals <- residuals^2

  # Calculate the mean of the squared residuals
  mean_squared_error <- mean(squared_residuals)

  # Return the mean squared error
  return(mean_squared_error)
}
```

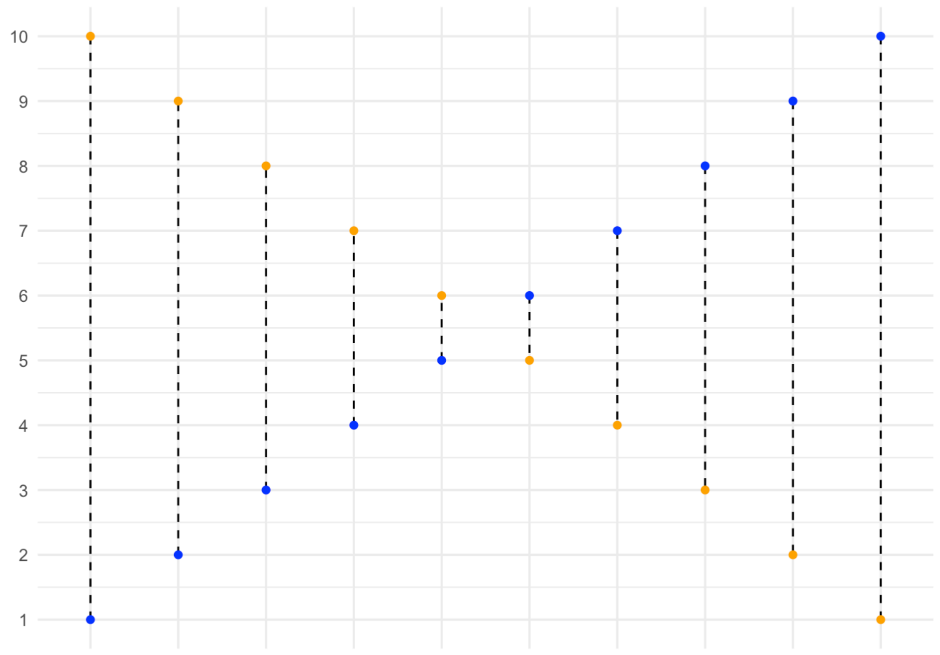
Make sure your `mse()` function works correctly by running the following code.

```
#remove the comment to run the function
mse(1:10, 10:1)
```

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

In the code, we state that our observed values correspond to $1, 2, \ldots, 9, 10$, while our predicted values correspond to $10, 9, \ldots, 2, 1$. This is graphed below, where the blue dots correspond to the observed values, and the yellow dots correspond to the predicted values. Using your function, you have now calculated the mean squared length of the dashed lines depicted in the graph below.

If your function works correctly, the value returned should equal 33.



11. **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`.**

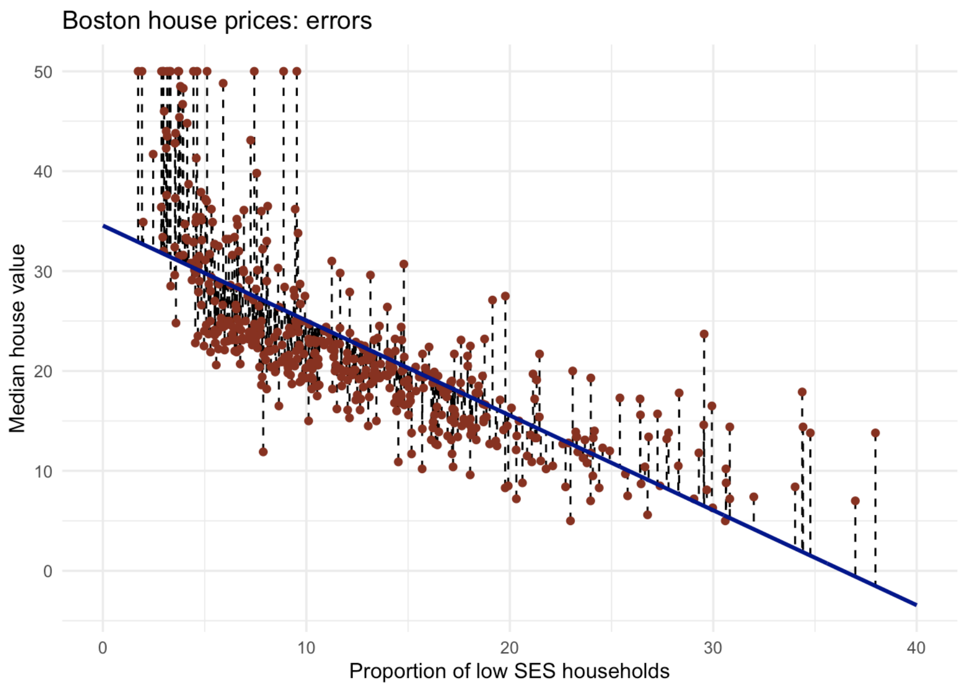
```
y_pred <- predict(lm_ses, newdata = Boston)


y_true <- Boston$medv


model_mse <- mse(y_true, y_pred)


print(model_mse)
```

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

You have calculated the mean squared length of the dashed lines in the plot below. As the MSE is computed using the data that was used to fit the model, we actually obtained the training MSE.

Below we continue with splitting our data in a training, test and validation set such that we can calculate the out-of sample prediction error during model building using the validation set, and estimate the true out-of-sample MSE using the test set.

```
## Warning: Using `size` aesthetic for lines was deprecate
d in ggplot2 3.4.0.
## ℹ Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see wher
e this warning was
## generated.
```



Note that you can also easily obtain how much the predictions on average differ from the observed values in the original scale of the outcome variable. To obtain this, you take the root of the mean square error. This is called the Root Mean Square Error, abbreviated as RMSE.

Train-validation-test splits

Next, we will use the `caret` package and the function `createDataPartition()` to obtain a training, test, and validation set from the `Boston` dataset. For more information on this package, see the caret website (https://topepo.github.io/caret/index.html). The training set will be used to fit our model, the validation set will be used to calculate the out-of sample prediction error during model building, and the test set will be used to estimate the true out-of-sample MSE.

Use the code given below to obtain training, test, and validation set from the `Boston` dataset.

```r
library(caret)
# define the training partition
train_index <- createDataPartition(Boston$medv, p = .7,
                                  list = FALSE,
                                  times = 1)

# split the data using the training partition to obtain tr
aining data
boston_train <- Boston[train_index,]

# remainder of the split is the validation and test data (
still) combined
boston_val_and_test <- Boston[-train_index,]

# split the remaining 30% of the data in a validation and
test set
val_index <- createDataPartition(boston_val_and_test$medv,
p = .66,
                                  list = FALSE,
                                  times = 1)

boston_valid <- boston_val_and_test[val_index,]
boston_test  <- boston_val_and_test[-val_index,]



# Outcome of this section is that the data (100%) is split
into:
# training (~70%)
# validation (~20%)
# test (~10%)
```

Note that creating the partitions using the `y` argument (letting the function know what your dependent variable will be in the analysis), makes sure that when your dependent variable is a factor, the random sampling occurs within each class and should preserve the overall class distribution of the data.

We will set aside the `boston_test` dataset for now.

12. **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
## -9.902 -3.900 -1.528  1.964 24.553
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 34.31916    0.66505   51.60   <2e-16 ***
## lstat       -0.93094    0.04575  -20.35   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.
1 ' ' 1
##
## Residual standard error: 6.326 on 354 degrees of freedo
m
## Multiple R-squared:  0.5391, Adjusted R-squared:  0.537
8
## F-statistic:   414 on 1 and 354 DF,  p-value: < 2.2e-16
```

```
#lm(medv ~ lstat, data = boston_train): This line creates
a linear regression model where medv (median value of owne
r-occupied homes) is predicted from lstat (percentage of l
ower status of the population). The model is trained using
the boston_train dataset, which is assumed to be a subset
of the Boston data prepared for training purposes.

#summary(model_1): This function call will print a summary
of the linear model, providing detailed information about
the coefficients (including estimates and significance), t
he overall fit of the model (R-squared, F-statistic), and
other diagnostic measures. The summary helps in evaluating
how well lstat predicts medv and whether the relationship
is statistically significant.
```

13. **Calculate the MSE with this object. Save this value as `model_1_mse_train`.**

```
y_pred_train <- predict(model_1, newdata = boston_train)

# Actual values from the training dataset
y_true_train <- boston_train$medv

# Calculate the Mean Square Error using the mse function
model_1_mse_train <- mse(y_true_train, y_pred_train)

# Save this value
print(model_1_mse_train)
```

```
## [1] 39.79702
```

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

```
# Generate predicted values using the trained model on the
validation dataset
y_pred_valid <- predict(model_1, newdata = boston_test)

# Actual values from the validation dataset
y_true_valid <- boston_test$medv

# Calculate the Mean Square Error using the mse function f
or the validation data
model_1_mse_valid <- mse(y_true_valid, y_pred_valid)

# Save this value
print(model_1_mse_valid)
```

```
## [1] 42.25584
```

This is the estimated out-of-sample mean squared error.

15. **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_trai
n)



y_pred_train_2 <- predict(model_2, newdata = boston_train)

# Actual values from the training dataset
y_true_train_2 <- boston_train$medv

# Calculate the Mean Square Error using the mse function f
or the training data
model_2_mse_train <- mse(y_true_train_2, y_pred_train_2)

print(model_2_mse_train)
```

```
## [1] 37.75858
```

```
y_pred_valid_2 <- predict(model_2, newdata = boston_test)



y_true_valid_2 <- boston_test$medv

# Calculate the Mean Square Error using the mse function f
or the validation data
model_2_mse_valid <- mse(y_true_valid_2, y_pred_valid_2)



print(model_2_mse_valid)
```

```
## [1] 42.1523
```

16. **Compare model 1 and model 2 in terms of their training and**

validation MSE. Which would you choose and why?

```
#Training MSE Comparison:
#model_1_mse_train vs. model_2_mse_train: If model_2_mse_t
rain is lower than model_1_mse_train, it suggests that mod
el_2, with the additional predictors (age and tax), fits t
he training data better. A lower training MSE indicates th
at the model captures more of the variance in the data due
to more variables explaining the variability in medv.

#Validation MSE Comparison:
#model_1_mse_valid vs. model_2_mse_valid: This comparison
is crucial because it evaluates how well each model genera
lizes to unseen data. If model_2_mse_valid is significantl
y lower than model_1_mse_valid, it suggests that the compl
exity added by including age and tax as predictors does no
t lead to overfitting and indeed helps in making more accu
rate predictions on new data.
```

In choosing the best model, you should base your answer on the validation MSE. Using the out of sample mean square error, we have made a model decision (which parameters to include, only `lstat`, or using `age` and `tax` in addition to `lstat` to predict housing value). Now we have selected a final model.

17. **For your final model, retrain the model one more time using both the training *and* the validation set. Then, calculate the test MSE based on the (retrained) final model. What does this number tell you?**

```r
combined_data <- rbind(boston_train, boston_test)

final_model <- lm(medv ~ lstat + age + tax, data = combine
d_data)

# Predict using the final model on the test dataset
y_pred_test <- predict(final_model, newdata = boston_test)
y_true_test <- boston_test$medv

# Calculate Mean Square Error on the test set
final_model_mse_test <- mse(y_true_test, y_pred_test)
print(final_model_mse_test)
```

```
## [1] 41.92099
```

As you will see during the remainder of the course, usually we set apart the test set at the beginning and on the remaining data perform the train-validation split multiple times. Performing the train-validation split multiple times is what we for example do in cross validation (see below). The validation sets are used for making model decisions, such as selecting predictors or tuning model parameters, so building the model. As the validation set is used to base model decisions on, we can not use this set to obtain a true out-of-sample MSE. That's where the test set comes in, it can be used to obtain the MSE of the final model that we choose when all model decisions have been made. As all model decisions have been made, we can use all data except for the test set to retrain our model one last time using as much data as possible to estimate the parameters for the final model.

Optional: cross-validation

This is an advanced exercise. Some components we have seen before in this lab, but some things will be completely new. Try to complete it by yourself, but don't worry if you get stuck. If you need to refresh your memory on `for loops` in `R`, have another look at lab 1 on the course website.

Use help in this order:

- R help files
- Internet search & stack exchange
- Your peers
- The answer, which shows one solution

You may also just read the answer when they have been made available and try to understand what happens in each step.

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

Inputs:

- `formula` : a formula just as in the `lm()` function
- `dataset` : a data frame
- `k` : the number of folds for cross validation
- any other arguments you need necessary

Outputs:

- Mean square error averaged over folds

```
crossval <- function(formula, dataset, k) {
  set.seed(123)  # For reproducibility
  n <- nrow(dataset)
  indices <- sample(1:k, n, replace = TRUE)
  mse_values <- numeric(k)

  for (i in 1:k) {
    test_idx <- which(indices == i)
    train_idx <- setdiff(1:n, test_idx)

    train_data <- dataset[train_idx, ]
    test_data <- dataset[test_idx, ]

    model <- lm(formula, data = train_data)
    predictions <- predict(model, newdata = test_data)
    mse_values[i] <- mean((test_data$medv - predictions)^
2)
  }

  return(mean(mse_values))
}
```

19. **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 formula

`medv ~ lstat + I(lstat^2) + age + tax` .

```
dataset <- Boston  # Assuming 'Boston' is your dataset
k <- 9  # Number of folds

# Linear model with lstat, age, and tax
mse1 <- crossval(medv ~ lstat + age + tax, dataset, k)

# Linear model with lstat, lstat^2, age, and tax
mse2 <- crossval(medv ~ lstat + I(lstat^2) + age + tax, da
taset, k)

# Output the results
print(paste("MSE for model 1 (lstat + age + tax):", mse1))
```

```
## [1] "MSE for model 1 (lstat + age + tax): 38.3364006538
797"
```

```
print(paste("MSE for model 2 (lstat + I(lstat^2) + age + t
ax):", mse2))
```

```
## [1] "MSE for model 2 (lstat + I(lstat^2) + age + tax):
28.3918356637821"
```

```
# Comparing MSEs: The mean square errors (MSE) from the cr
oss-validation results
#can indicate which model performs better in terms of pred
iction accuracy.
#The model with the lower MSE is considered to have a bett
er fit because it means
#less deviation between the predicted and actual values.

# Model Complexity: The second model includes a quadratic
term for lstat, which
#might capture more complex relationships in the data but
also risks overfitting.
```