Practical 7

Emilia Löscher

2022-10-14

First, load the packages:

```
library(MASS)
library(splines)
library(ISLR)
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
```

```
library(cowplot)
```

Set a seed

```
set.seed(45)
```

1.

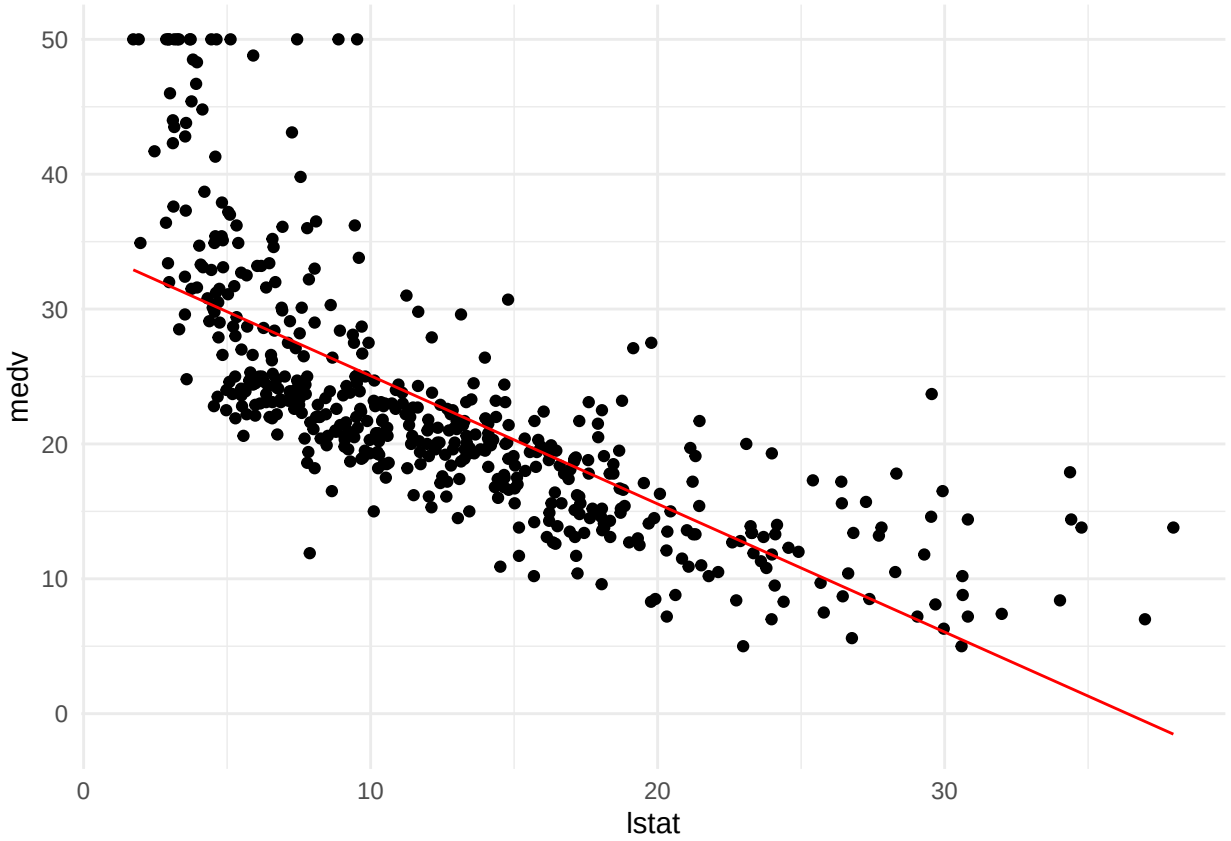
Create a function called pred_plot() that takes as input an lm object, which outputs the above plot but with a prediction line generated from the model object using the predict() method.

```
pred_plot <- function(lm_ob , data = Boston){
  #sequence for which predictions are made to create a line
  sequence <- as.data.frame(seq(min(data$lstat), max(data$lstat), length.out = 1000 ))
  colnames(sequence) <- c("lstat")

  #make predictions for sequence
  lm_pred <- predict(lm_ob, type = "response", newdata = sequence)

  #plot
  data %>%
  ggplot(aes(x = lstat, y = medv)) +
  geom_point() +
  geom_line(data = tibble(lstat= sequence$lstat, medv = lm_pred), col= "red")+
  theme_minimal()
}
```

2.

Create a linear regression object called lin_mod which models medv as a function of lstat. Check if your prediction plot works by running pred_plot(lin_mod). Do you see anything out of the ordinary with the predictions?

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

pred_plot(lm_ob = lin_mod)
```

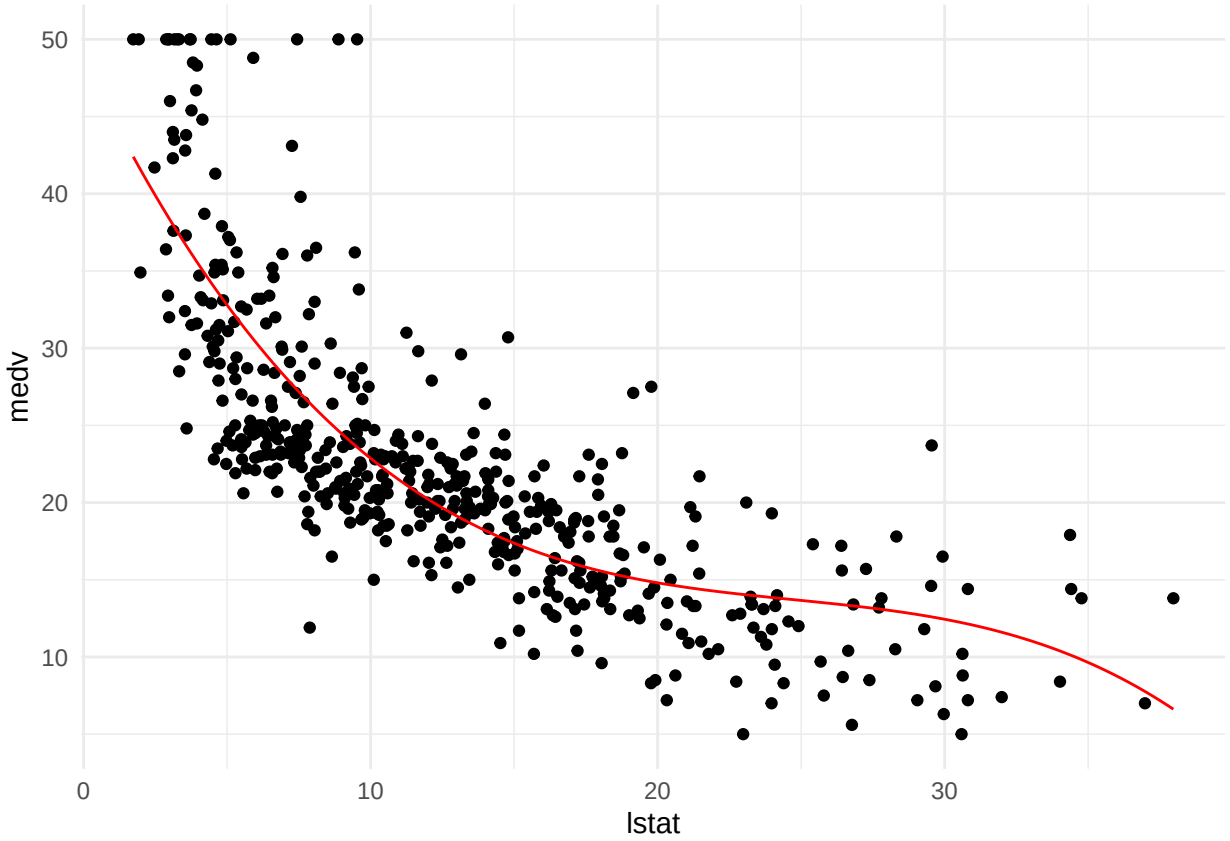


There are negative predictions for lstat > 36 which does not make sense for a median value of owner-occupied homes in \$1000s.

3.

Create another linear model pn3_mod, where you add the second and third-degree polynomial terms I(lstat^2) and I(lstat^3) to the formula. Create a pred_plot() with this model.

```
pn3_mod <- lm(medv~ lstat +I(lstat^2) + I(lstat^3), data = Boston)

pred_plot(pn3_mod)
```



Now, there are no negative values predicted anymore.

4.

Play around with the poly() function. What output does it generate with the arguments degree = 3 and raw = TRUE?

```
poly(3,degree =4 , raw =TRUE)
```

```
##      1 2  3  4
## [1,] 3 9 27 81
## attr(,"degree")
## [1] 1 2 3 4
## attr(,"class")
## [1] "poly"   "matrix"
```

```
poly(4,degree =2 , raw =TRUE)
```

```
##      1  2
## [1,] 4 16
## attr(,"degree")
## [1] 1 2
## attr(,"class")
## [1] "poly"   "matrix"
```

```
poly(2,degree =3 , raw =TRUE)
```

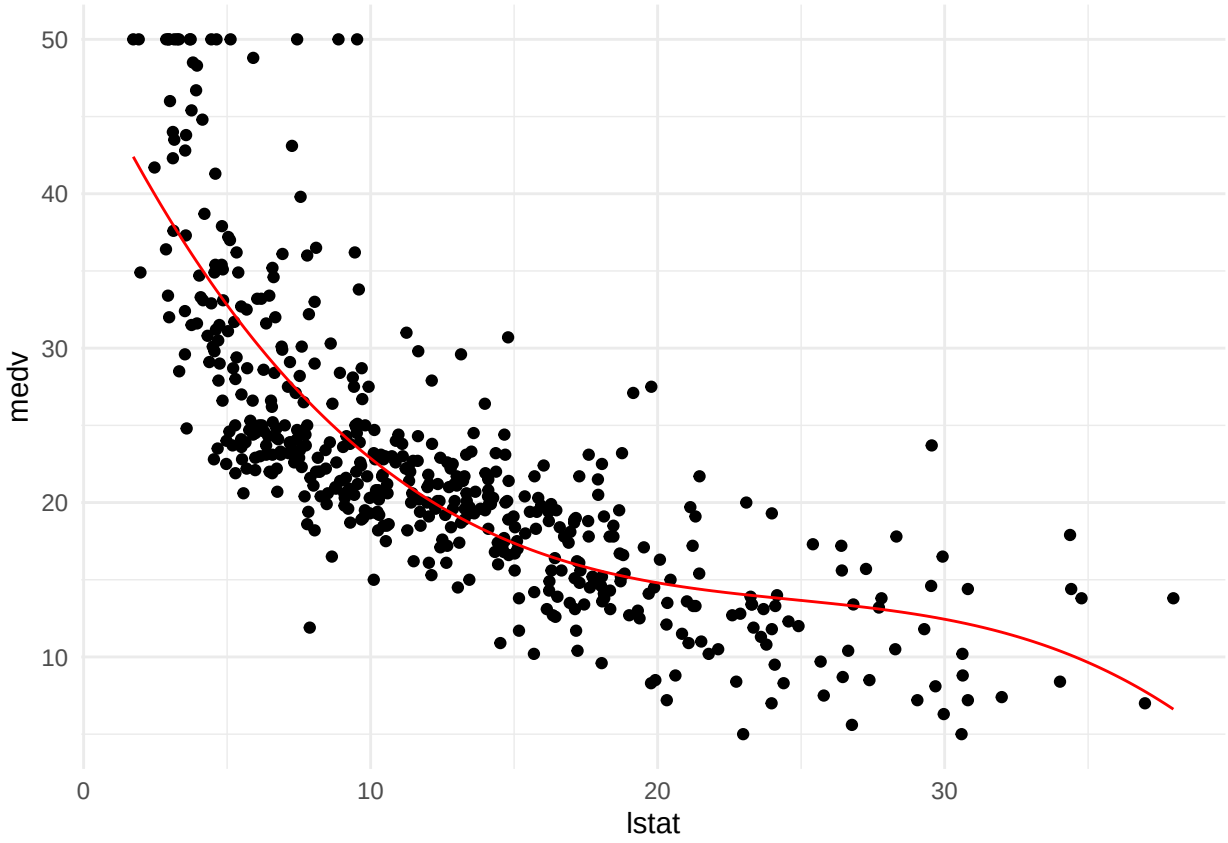
```
##      1 2 3
## [1,] 2 4 8
## attr(,"degree")
## [1] 1 2 3
## attr(,"class")
## [1] "poly"   "matrix"
```

It calculates x (first argument) to the power of 1, 2, ... up to the number defined by the argument "degree".

5.

Use the poly() function directly in the model formula to create a 3rd-degree polynomial regression predicting medv using lstat. Compare the prediction plot to the previous prediction plot you made. What happens if you change the poly() function to raw = FALSE?

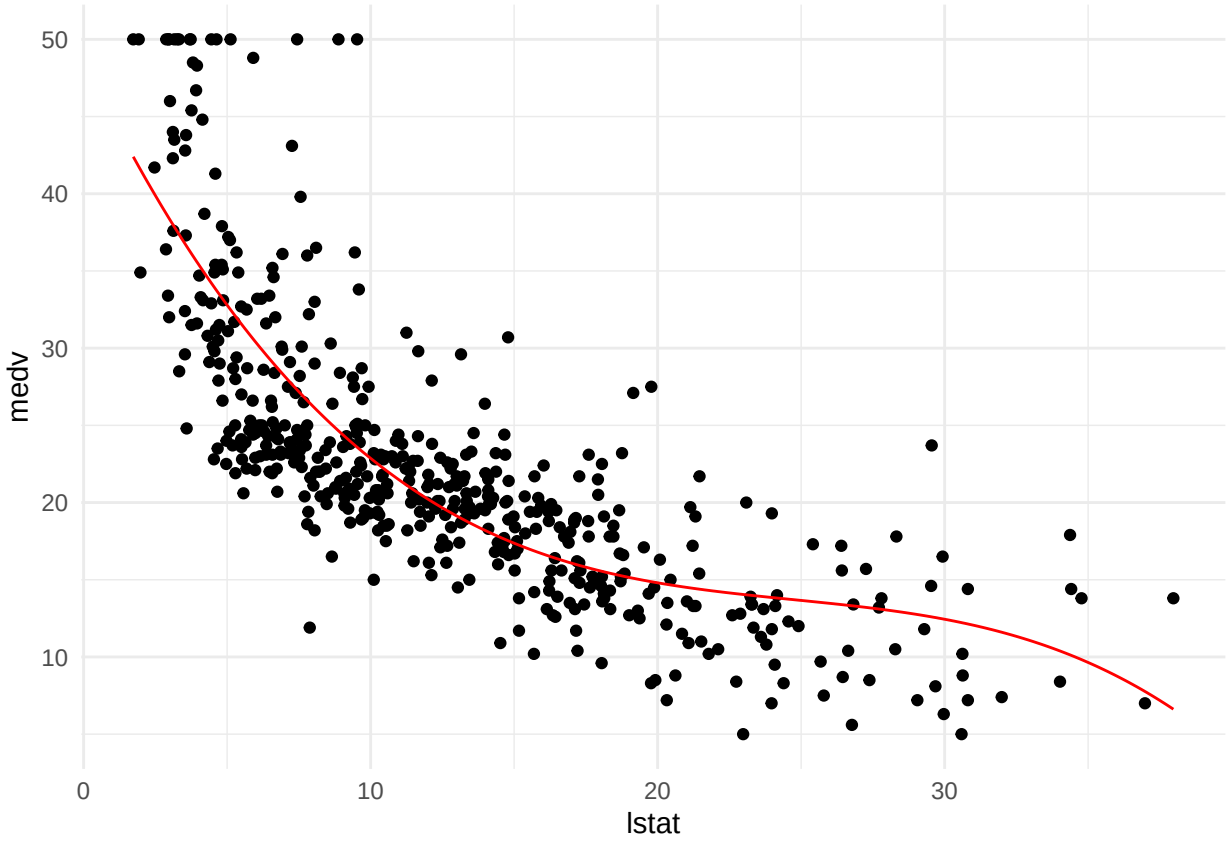
```
third_degree <- lm(medv~ poly(lstat, degree = 3, raw =TRUE), data = Boston)

pred_plot(third_degree)
```



```r
summary(third_degree)
```

```
##
## Call:
## lm(formula = medv ~ poly(lstat, degree = 3, raw = TRUE), data = Boston)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -14.5441  -3.7122  -0.5145   2.4846  26.4153
##
## Coefficients:
##                                      Estimate Std. Error t value Pr(>|t|)
## (Intercept)                         48.6496253  1.4347240  33.909  < 2e-16 ***
## poly(lstat, degree = 3, raw = TRUE)1 -3.8655928  0.3287861 -11.757  < 2e-16 ***
## poly(lstat, degree = 3, raw = TRUE)2  0.1487385  0.0212987   6.983 9.18e-12 ***
## poly(lstat, degree = 3, raw = TRUE)3 -0.0020039  0.0003997  -5.013 7.43e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.396 on 502 degrees of freedom
## Multiple R-squared:  0.6578, Adjusted R-squared:  0.6558
## F-statistic: 321.7 on 3 and 502 DF,  p-value: < 2.2e-16
```

```r
third_degree_false <- lm(medv~ poly(lstat, degree = 3, raw =FALSE), data = Boston)

pred_plot(third_degree_false)
```

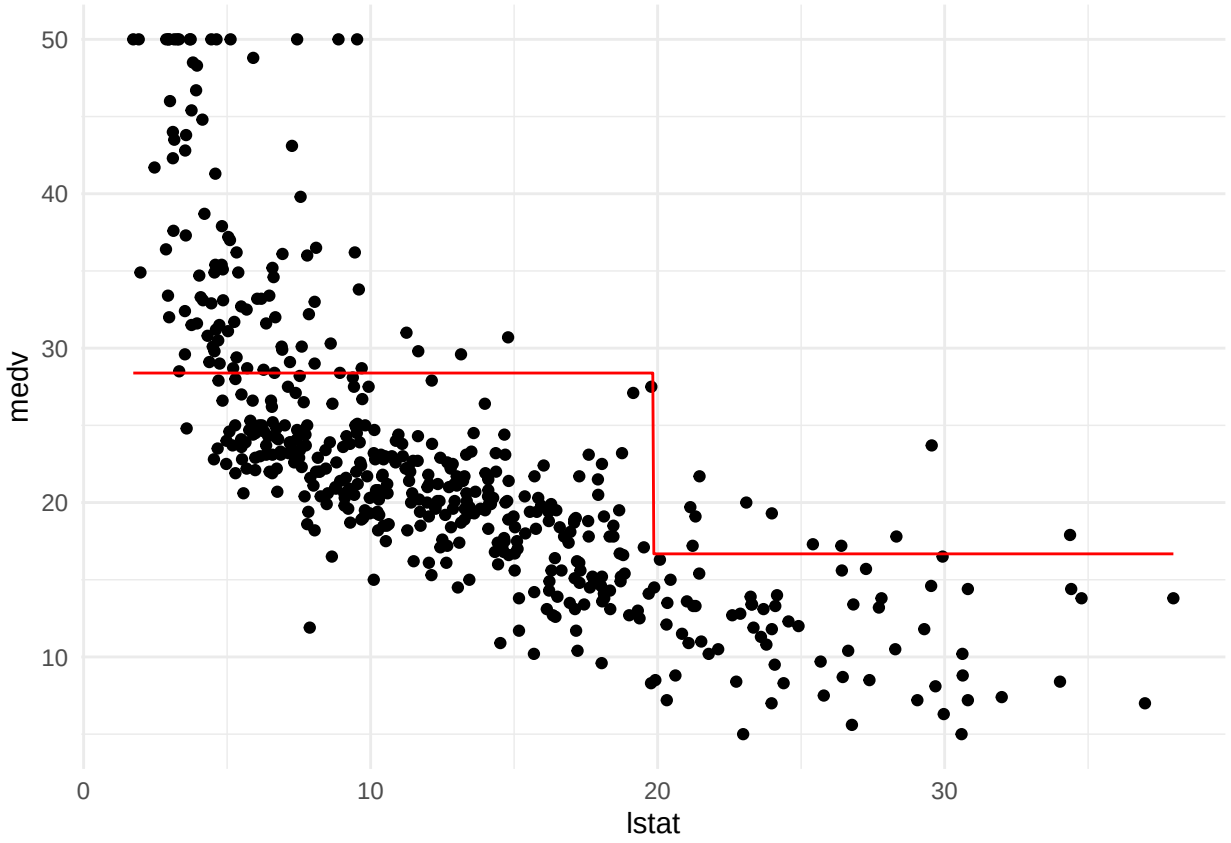


Same plot as before in 3 no matter whether raw is TRUE or FALSE. Hence, the predicted values for raw = TRUE and raw = FALSE are the same. With raw=FALSE we can see if ordering the polynomial in a specific way significantly improves the regression in comparison to lower orders.

6.

Create a model called pw2_mod with one predictor: I(lstat <= median(lstat)). Create a pred_plot with this model. Use the coefficients in coef(pw2_mod) to find out what the predicted value for a low-lstat neighbourhood is.

```
pw2_mod <- lm(medv ~ I(lstat <= median(lstat)), data = Boston)
pred_plot(pw2_mod)
```




```
coef(pw2_mod)
```

```
##                   (Intercept) I(lstat <= median(lstat))TRUE
##                      16.67747                       11.71067
```

```
sum(coef(pw2_mod))
```

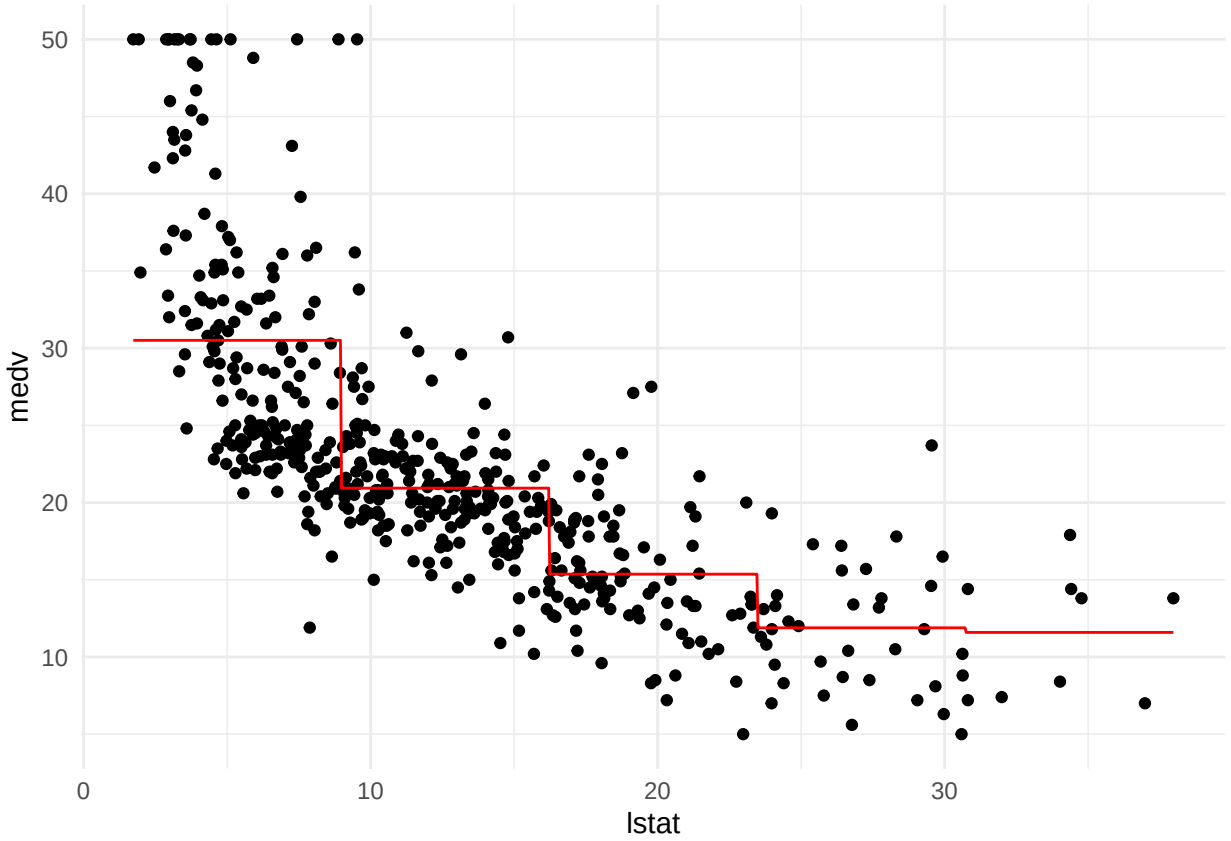
```
## [1] 28.38814
```

The predicted value for a low-stat neighborhood is 28.39.

7.

Use the cut() function in the formula to generate a piecewise regression model called pw5_mod that contains 5 equally spaced sections. Again, plot the result using pred_plot.

```
pw5_mod <- lm(medv~ cut(lstat, 5), data = Boston)
pred_plot(pw5_mod)
```

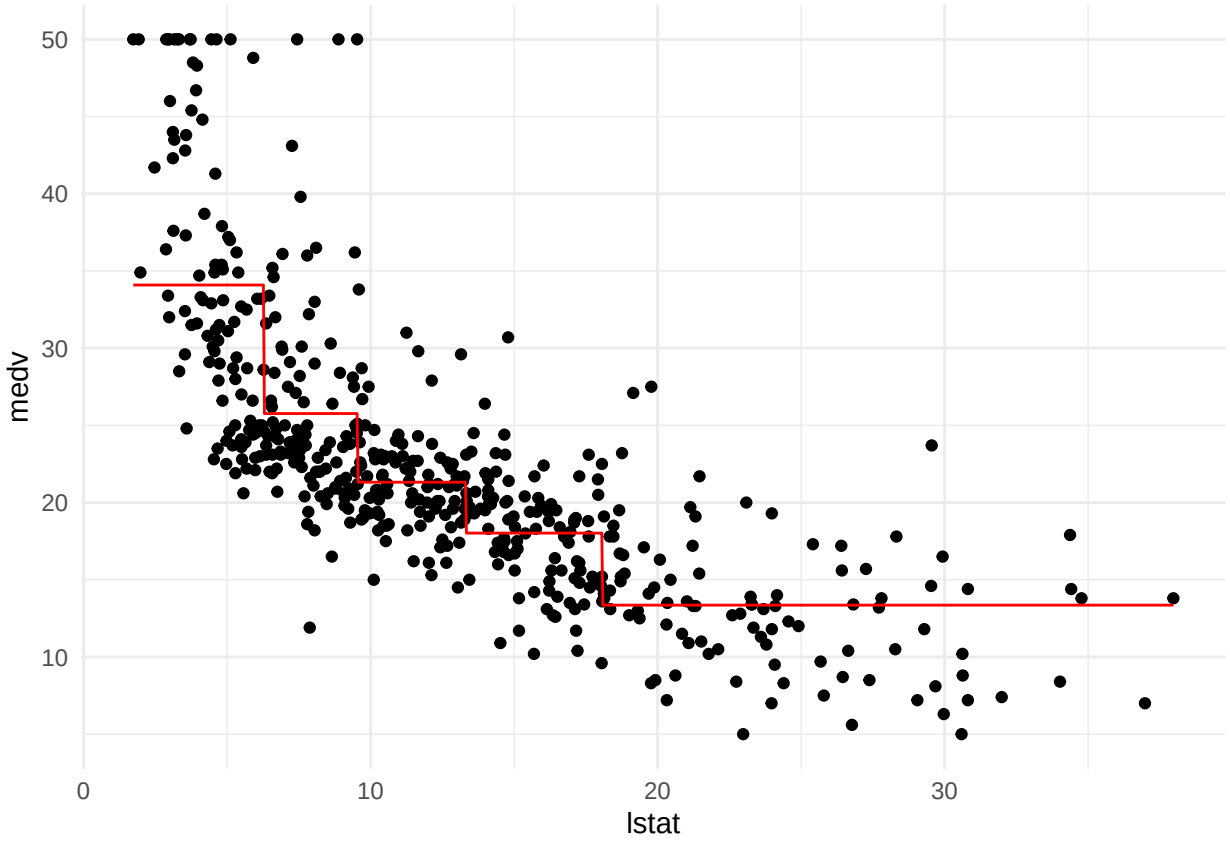


We can see that it is now divided up into 5 parts and for each part a linear model is fitted. Not that the slope for all parts is 0 (constant).

8.

Optional: Create a piecewise regression model pwq_mod where the sections are not equally spaced, but have equal amounts of training data. Hint: use the quantile() function.

```
spacing <- c(-Inf, quantile(Boston$lstat,  probs = c(.2, .4, .6, .8)), Inf)
pwq_model <- lm(medv ~ cut(lstat, spacing), data = Boston)

pred_plot(pwq_model)
```



Now the data set is split up in a way that each part contains about the same amount of data points.

9.

This function does not have comments. Copy - paste the function and add comments to each line. To figure out what each line does, you can first create "fake" vec and knots variables, for example vec <- 1:20 and knots <- 2 and try out the lines separately.

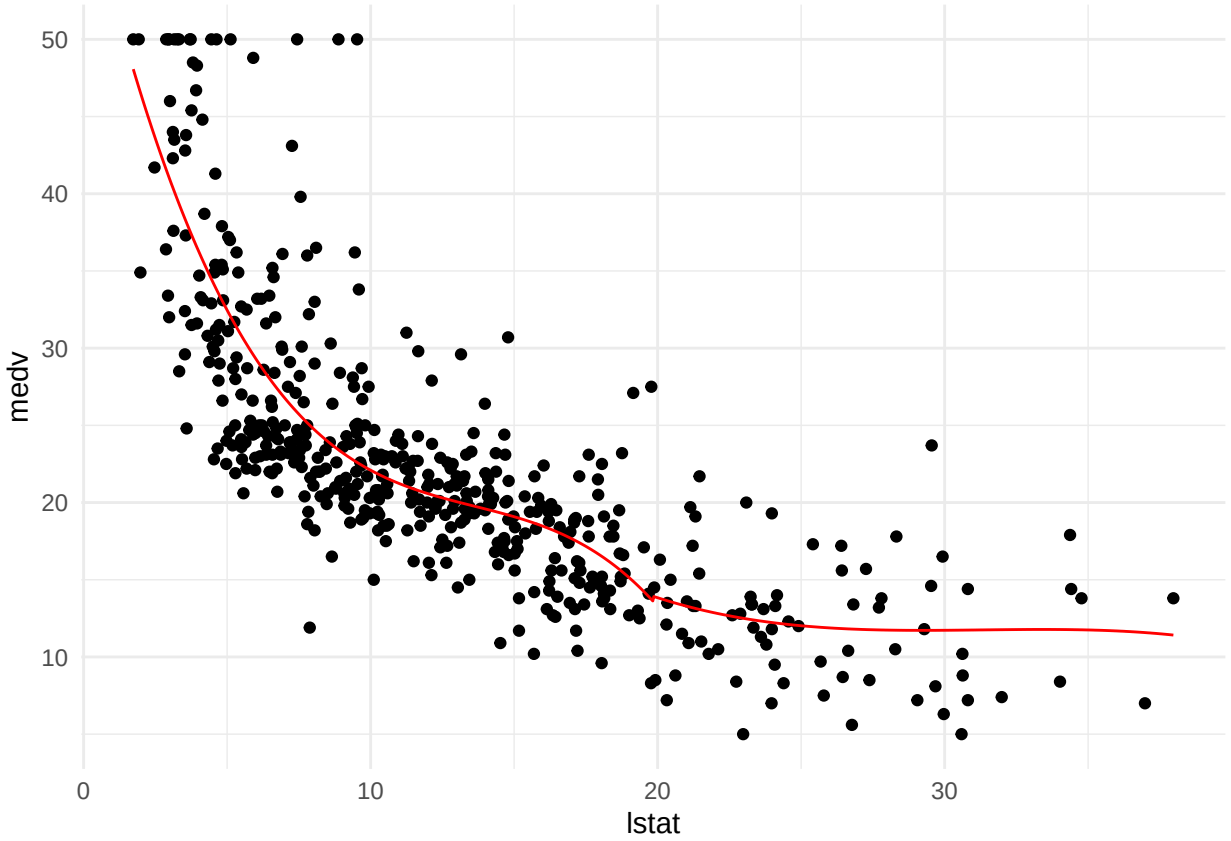
```
piecewise_cubic_basis <- function(vec, knots = 1) {
  #if the number of knots is equal
  #to 0, the function returns the polynomials of the given vector up to the third polynomial
  if (knots == 0) return(poly(vec, degree = 3, raw = TRUE))

  #if the number of knots is unequal 0 the following things happen:

  # the vector is cut into number of knots + 1 parts
  cut_vec <- cut(vec, breaks = knots + 1)

  # an empty matrix named "out" is created
  # the number of rows equals the length of the vector
  #because the number of columns is equal to 0, it technically just empty.
  out <- matrix(nrow = length(vec), ncol = 0)
```

```
  #for each of the parts of the cut vector is split up into, the following steps are taken
  for (lvl in levels(cut_vec)) {
    #save the original vector as tmp
    tmp <- vec

    #set each entry that does not fall into the part of interest to 0
    tmp[cut_vec != lvl] <- 0

    # compute the polynomials up to the third order
    # for the entries of the vector that fall into the
    # part of interest, and add them in the result matrix "out"
    out <- cbind(out, poly(tmp, degree = 3, raw = TRUE))
  }
  #output the result matrix "out"
  out
}
```

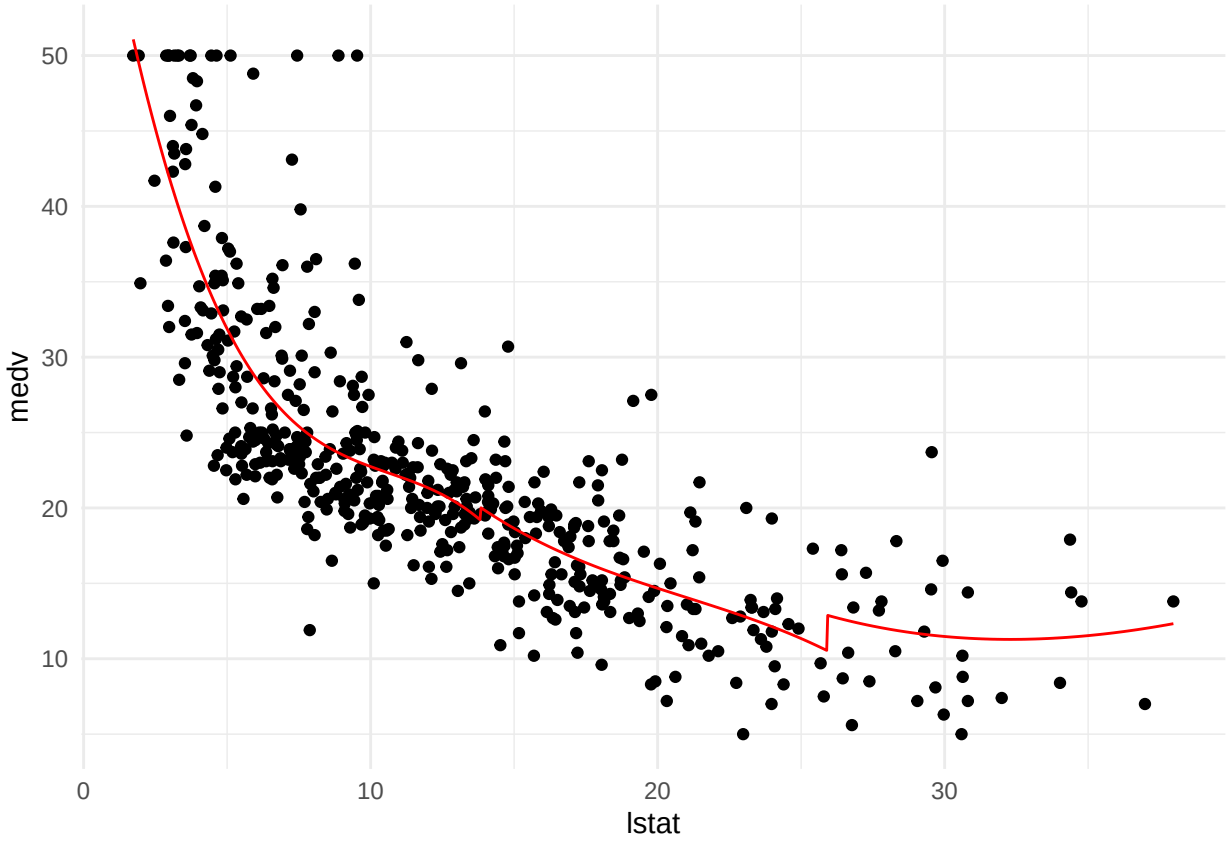
10.

Create piecewise cubic models with 1, 2, and 3 knots (pc1_mod - pc3_mod) using this piecewise cubic basis function. Compare them using the pred_plot() function.

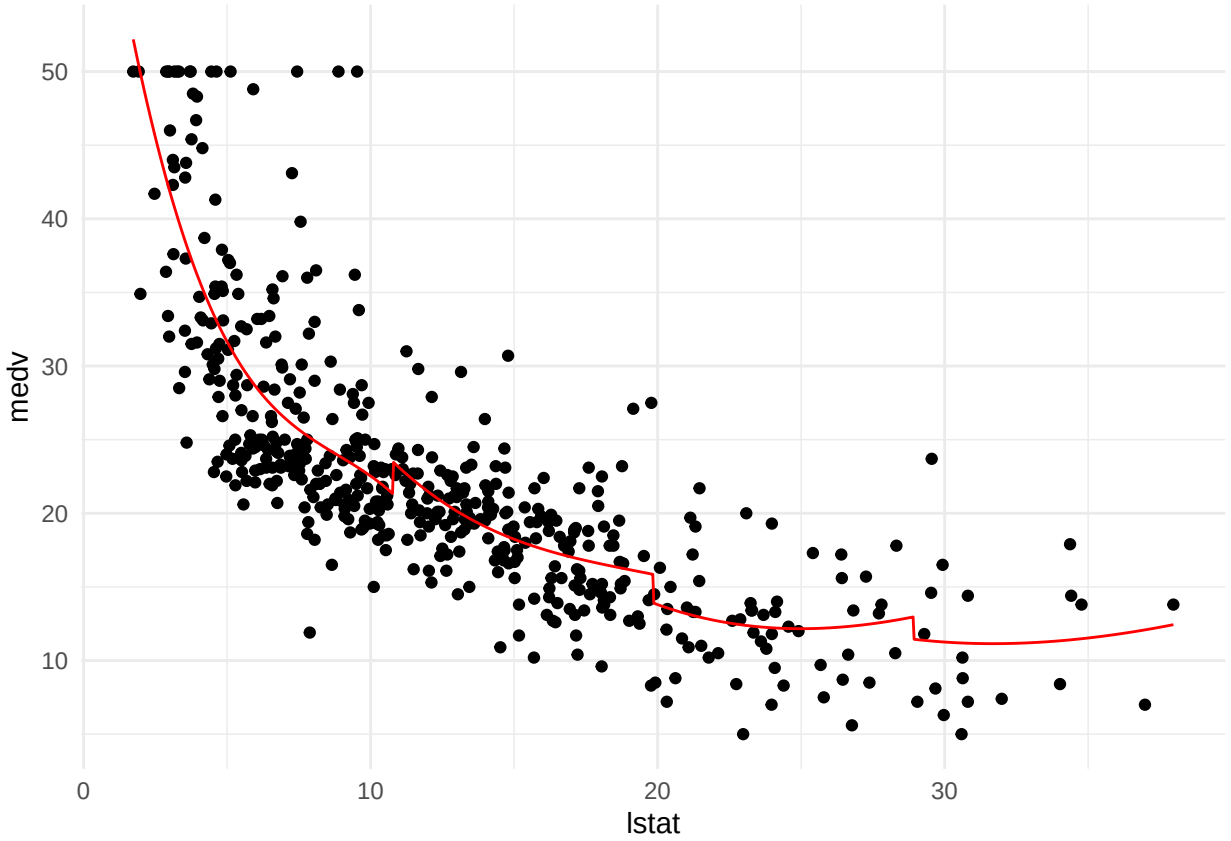
```
pc1_mod <- lm(medv~ piecewise_cubic_basis(lstat , knots = 1), data = Boston)
pc2_mod <- lm(medv~ piecewise_cubic_basis(lstat , knots = 2), data = Boston)
pc3_mod <- lm(medv~ piecewise_cubic_basis(lstat , knots = 3), data = Boston)

pred_plot(pc1_mod)
```




```
pred_plot(pc2_mod)
```




```
pred_plot(pc3_mod)
```



For knots = 1 we obtain a cubic fit for the data set being split into two pieces. To each of the two parts, a cubic model is fit. For knots = 2 and knots = 3 the same happens with the data set being split into three and four pieces, respectively.

11.

Create a data frame called boston_tpb with the columns medv and lstat from the Boston dataset.

```
boston_tpb <- data.frame(medv = Boston$medv, lstat = Boston$lstat)
```

12.

Now use mutate to add squared and cubed versions of the lstat variable to this dataset.

```
boston_tpb %<>% mutate(sq_lstat = lstat^2, cub_lstat = lstat^3)
```

13.

Use mutate to add a column lstat_tpb to this dataset which is 0 below the median and has value (lstat - median(lstat))^3 above the median. Tip: you may want to use ifelse() within your mutate() call.

```
boston_tpb %<>% mutate(lstat_tpb = ifelse(lstat < median(lstat), 0, (lstat - median(lstat))^3))
```

14.

Create a linear model tpb_mod using the lm() function. How many predictors are in the model? How many degrees of freedom does this model have?

```
tpb_mod <- lm(medv~ ., data = boston_tpb)
summary(tpb_mod)
```

```
##
## Call:
## lm(formula = medv ~ ., data = boston_tpb)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -13.5106  -3.0547  -0.7488   2.1178  27.1383
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  65.514246   3.054303  21.450  < 2e-16 ***
## lstat       -10.324275   1.089886  -9.473  < 2e-16 ***
## sq_lstat      0.856576   0.116108   7.377 6.75e-13 ***
## cub_lstat    -0.025484   0.003810  -6.688 6.05e-11 ***
## lstat_tpb     0.026582   0.004291   6.194 1.22e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.206 on 501 degrees of freedom
## Multiple R-squared:  0.6822, Adjusted R-squared:  0.6796
## F-statistic: 268.8 on 4 and 501 DF,  p-value: < 2.2e-16
```

```
tpb_mod$df.residual
```

```
## [1] 501
```

```
length(coef(tpb_mod))
```

```
## [1] 5
```

The model has 501 degrees of freedom and four predictors (+ intercept).

15.

Create a cubic spline model bs1_mod with a knot at the median using the bs() function. Compare its predictions to those of the tpb_mod using the predict() function on both models.

```
bs1_mod <- lm(medv ~ bs(lstat, knots = median(lstat)), data = Boston)
bs1_pred <- predict(bs1_mod, type= "response")

tpb_pred <- predict(tpb_mod)

mean(abs(tpb_pred - bs1_pred))
```

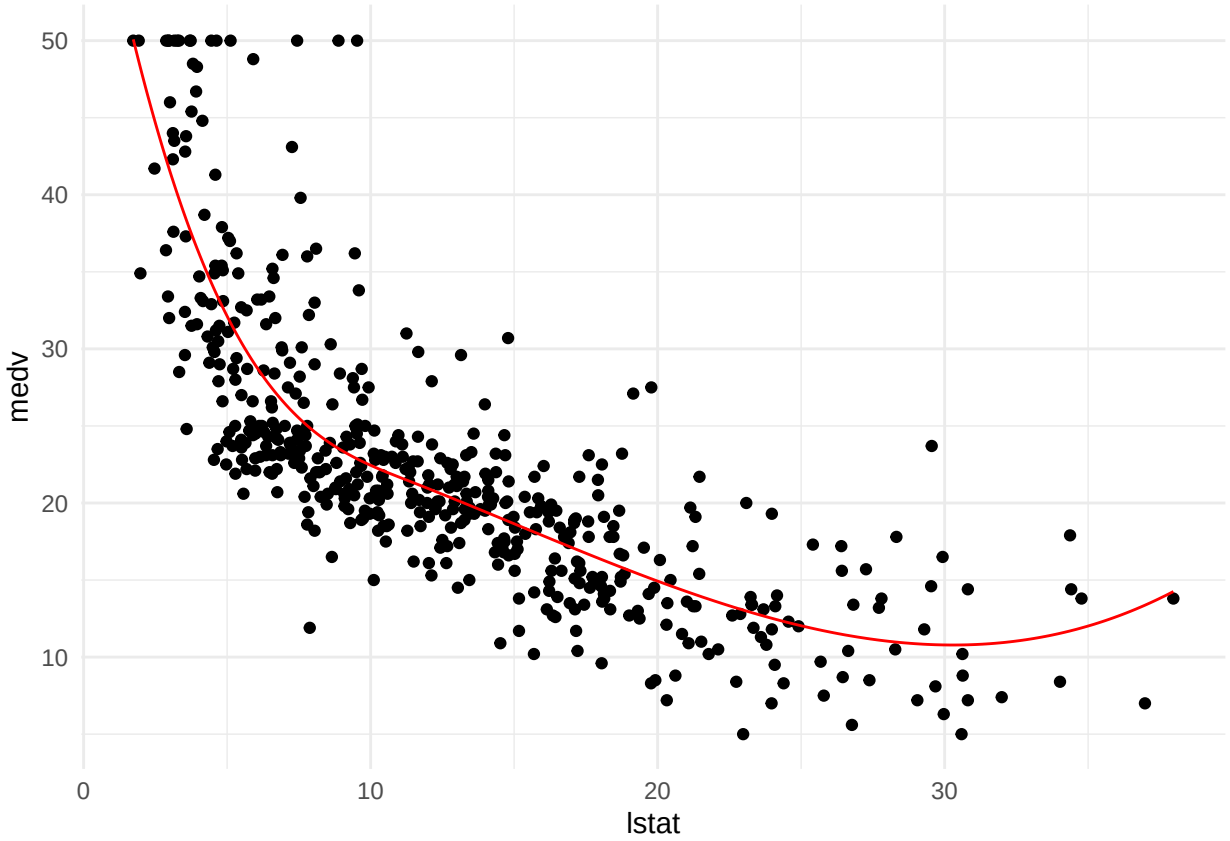
```
## [1] 3.449994e-13
```

There is almost no difference between the predictions from the tpb and the bs1 model.

16.

Create a prediction plot from the bs1_mod object using the plot_pred() function.

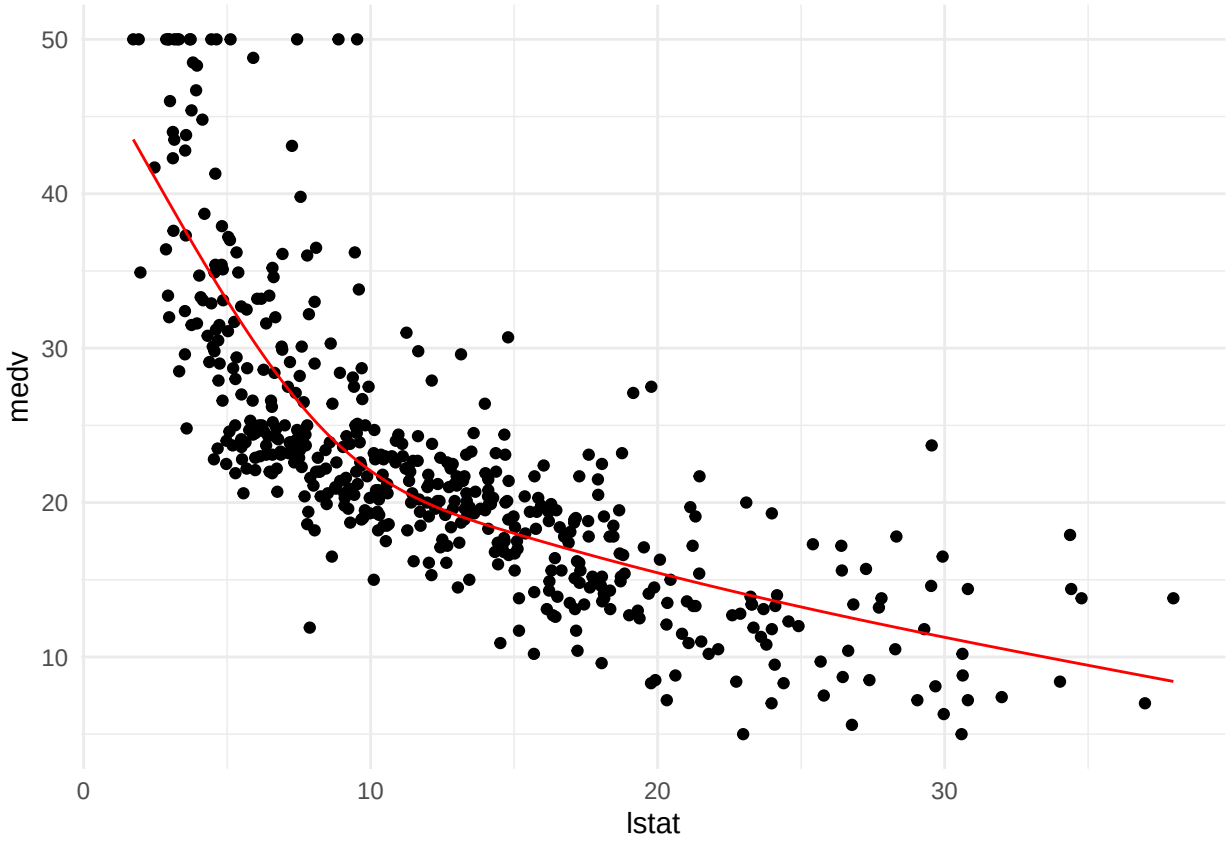
```
pred_plot(bs1_mod)
```



17.

##Create a natural cubic spline model (ns3_mod) with 3 degrees of freedom using the ns() function. Plot it, and compare it to the bs1_mod.

```
ns3_mod <- lm(medv ~ ns(lstat, df =3), data = Boston)

pred_plot(ns3_mod)
```

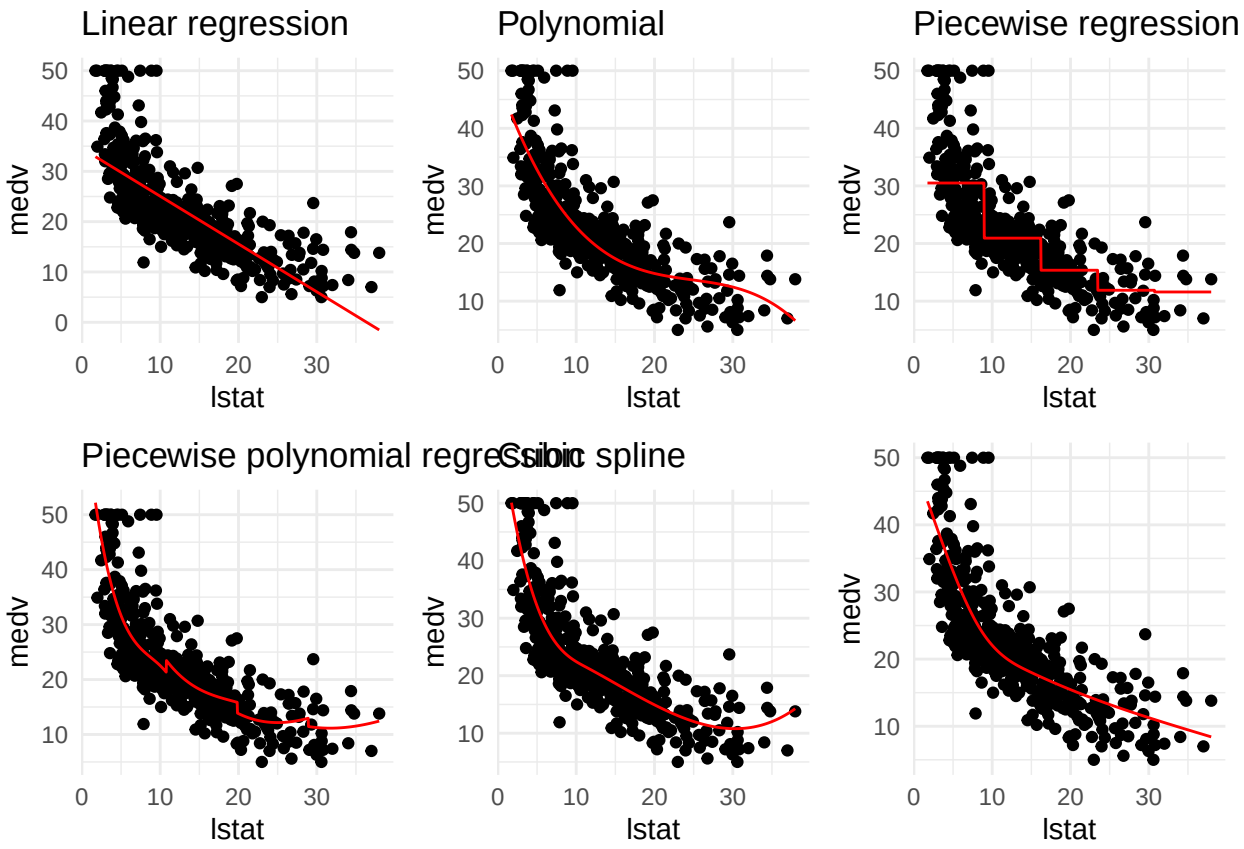


It has a similar fit as the bs1 model plot, but in the beginning it is not as steeply decreasing and the last part is still decreasing and not increasing again (as for bs1).

18.

Plot lin_mod, pn3_mod, pw5_mod, pc3_mod, bs1_mod, and ns3_mod and give them nice titles by adding + ggtitle("My title") to the plot. You may use the function plot_grid() from the package cowplot to put your plots in a grid.

```
plot_grid(pred_plot(lin_mod) + ggtitle("Linear regression"),
          pred_plot(pn3_mod) + ggtitle("Polynomial"),
          pred_plot(pw5_mod) + ggtitle("Piecewise regression"),
          pred_plot(pc3_mod) + ggtitle("Piecewise polynomial regression"),
          pred_plot(bs1_mod) + ggtitle("Cubic spline"),
          pred_plot(ns3_mod)) + ggtitle("Natural spline")
```



Nice overview to directly compare the different models.

19.

Use 12-fold cross validation to determine which of the 6 methods (lin, pn3, pw5, pc3, bs1, and ns3) has the lowest out-of-sample MSE.

```
#MSE function
mse <- function(y_true, y_pred) mean((y_true - y_pred)^2)



set.seed(1810)
myCV <- function(k = 12, data = Boston){
   mse_mat <- matrix(NA,  ncol = 12, nrow = 6)

  split <- c(rep(1:k, floor(nrow(data)/k)), 1:(nrow(data)%%k))
  split_shuff <- sample(split, length(split) )
  #adding column to randomly split the data set
  data$split <- split_shuff
  for(i in 1:k){
    #splitting the data set for each k into train and test
    data_train <- data[which(data$split != i),]
    data_test <- data[which(data$split == i),]
```

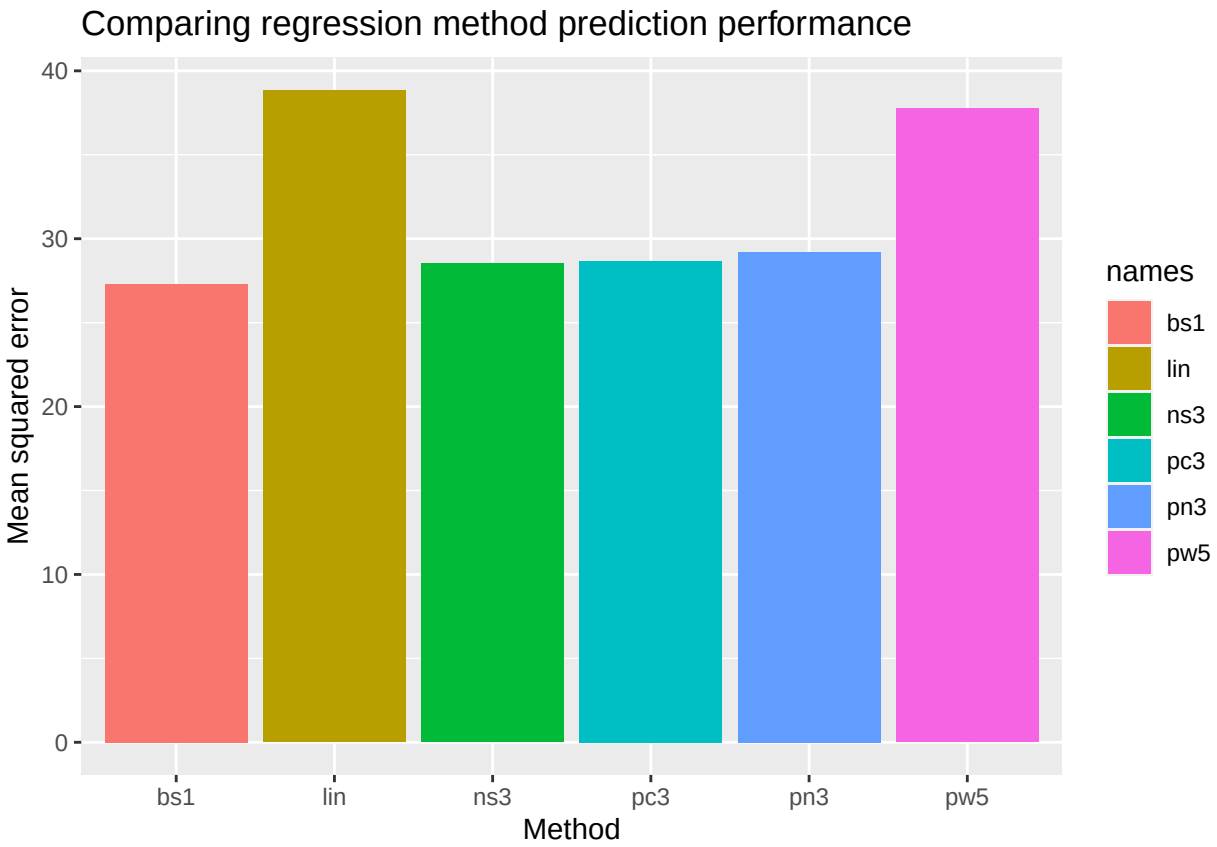
```
  #fitting the different models
    #linear
    lin <- lm(medv~ lstat, data = data_train)

    #polynomial
    pn3 <- lm(medv~ lstat +I(lstat^2) + I(lstat^3), data = data_train)

    #piecewise
    brks <- c(-Inf, 7, 15, 22, Inf)
    pw5 <- lm(medv~ cut(lstat, brks), data = data_train)

    #piecewise polynomial
    pc3 <- lm(medv~ piecewise_cubic_basis(lstat , knots = 3), data = data_train)

    #cubic spline
    bs1 <- lm(medv ~ bs(lstat, knots = median(lstat)), data = data_train)

    #natural spline
    ns3 <- lm(medv ~ ns(lstat, df =3), data = data_train)

    model_list <- list(lin, pn3, pw5, pc3, bs1, ns3)
    pred_list <- lapply(model_list, function(x) predict(x, newdata = data_test))

    for(j in 1:6){
     mse_mat[j,i] <- mse(data_test$medv, pred_list[[j]])
    }
  }
  return(mse_mat)
}
mse_res <- myCV()
```

```
## Warning in bs(lstat, degree = 3L, knots = 11.48, Boundary.knots = c(1.92, :
## einige 'x' Werte ausserhalb der Begrenzungsknoten erzeugen eventuell schlecht
## konditionierte Basis

## Warning in bs(lstat, degree = 3L, knots = 11.395, Boundary.knots = c(1.73, :
## einige 'x' Werte ausserhalb der Begrenzungsknoten erzeugen eventuell schlecht
## konditionierte Basis
```

```
rownames(mse_res) <- c("lin", "pn3", "pw5", "pc3", "bs1", "ns3")

rowMeans(mse_res)
```

```
##      lin      pn3      pw5      pc3      bs1      ns3
## 38.83971 29.20300 37.76762 28.68667 27.28753 28.51361
```

```
results <- data.frame("names"= rownames(mse_res), "mse" = rowMeans(mse_res))
ggplot(results, aes(x = names, y = mse, fill= names))+
  geom_bar(stat ="identity")+
   labs(
    x     = "Method",
    y     = "Mean squared error",
    title = "Comparing regression method prediction performance")
```



It can be seen that the MSE for the Natural spline (bs1) performs best as it has the lowest MSE (27.37). The simple linear model (lin) performs worst with the highest MSE (38.74). The pw5 performs similarly bad as the linear model. The remaining three models have similar performance with an MSE around 28.