

# Lab 8

## Stats 32: Introduction to R for Undergraduates

Harrison Li

04/25/2024

We return to the mpg dataset.

```
library(forcats)
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ----- tidyverse 2.0.0 --
## v dplyr      1.1.4      v readr      2.1.5
## v ggplot2    3.4.4      v stringr    1.5.1
## v lubridate  1.9.3      v tibble     3.2.1
## v purrr      1.0.2      v tidyr      1.3.1
## -- Conflicts ----- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()     masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
summary(mpg)
```

```
## manufacturer      model          displ          year
## Length:234        Length:234      Min.   :1.600   Min.   :1999
## Class :character  Class :character 1st Qu.:2.400   1st Qu.:1999
## Mode  :character  Mode  :character Median :3.300   Median :2004
##                                     Mean  :3.472   Mean  :2004
##                                     3rd Qu.:4.600 3rd Qu.:2008
##                                     Max.  :7.000   Max.  :2008
##      cyl          trans          drv          cty
## Min.   :4.000    Length:234      Length:234      Min.   : 9.00
## 1st Qu.:4.000    Class :character Class :character 1st Qu.:14.00
## Median :6.000    Mode  :character Mode  :character Median :17.00
## Mean   :5.889                                     Mean  :16.86
## 3rd Qu.:8.000                                     3rd Qu.:19.00
## Max.   :8.000                                     Max.  :35.00
##      hwy          fl          class
## Min.   :12.00    Length:234      Length:234
## 1st Qu.:18.00    Class :character Class :character
## Median :24.00    Mode  :character Mode  :character
## Mean   :23.44
## 3rd Qu.:27.00
## Max.   :44.00
```

## Multiple linear regression

Recall multiple linear regression is the extension of simple linear regression to the case of more than 1 predictor variable:

$$y = \beta_0 + \beta_1x_1 + \dots + \beta_px_p + e$$

where  $p > 1$  is the total number of predictors. The formula you have to pass into `lm()` takes the form `y ~ x_1 + ... + x_p`.

Last class, we tried to predict `hwy` from `cty`. Now let's add one additional predictor, `year`:

```
car_model_mult <- lm(formula=hwy ~ cty + year, data=mpg)
car_model_mult$coefficients
```

```
## (Intercept)      cty      year
## -99.14247615  1.33942478  0.04991331
```

The regression slopes are 1.339 for `cty` and 0.050 for `year`. This means that, **\*keeping year constant**, each 1 unit increase in a car's city MPG corresponds to a 1.339 increase in a car's highway MPG. It's important in your interpretation of each regression coefficient to note that you are keeping all other predictors constant.

1. How would you interpret the coefficient for `year`? How about the intercept in the model?

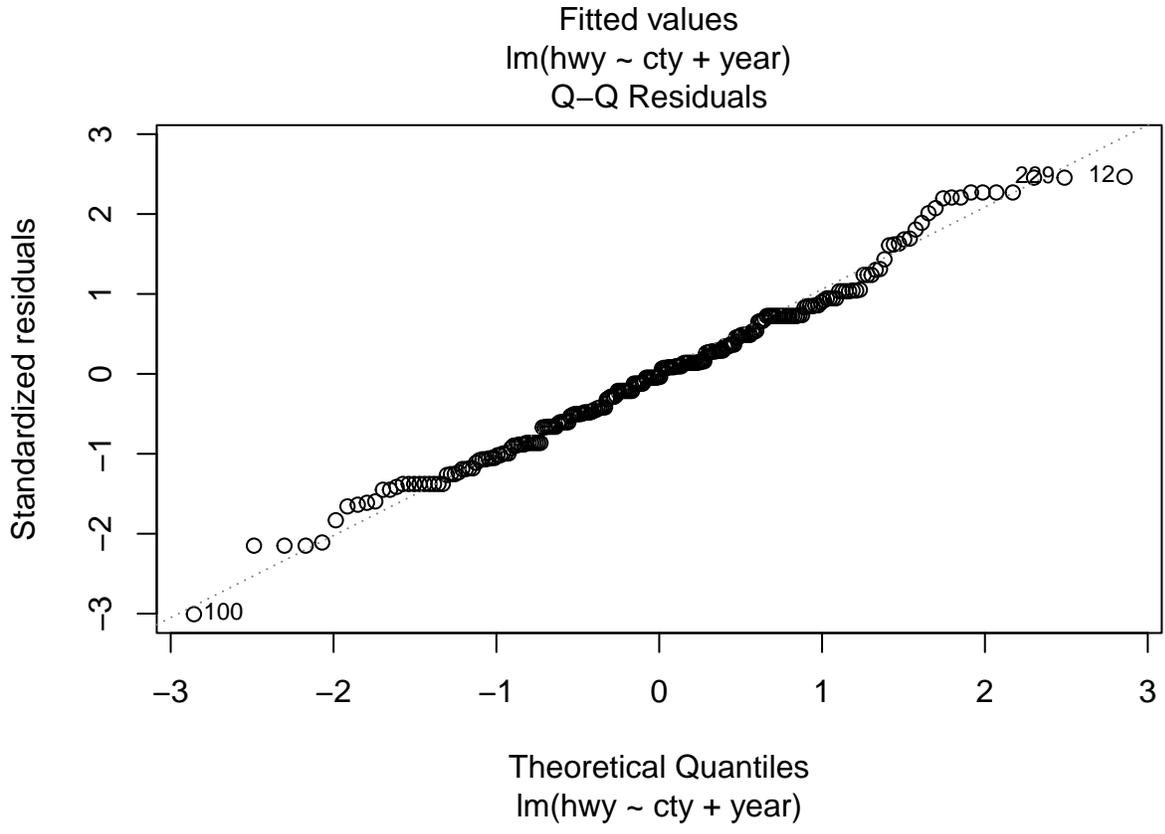
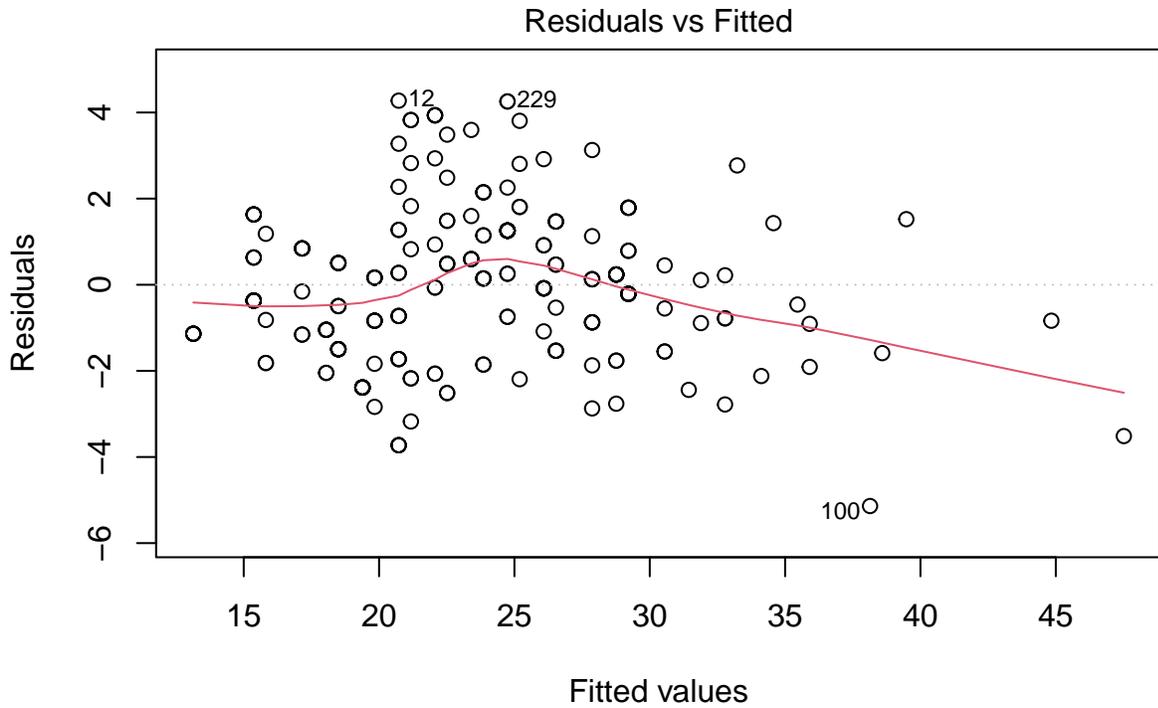
For prediction on new data, we now need a 2-column tibble: one column labeled `cty` and another labeled `year`. Let's get the highway mpg prediction for a car with 25 mpg manufactured in 2001:

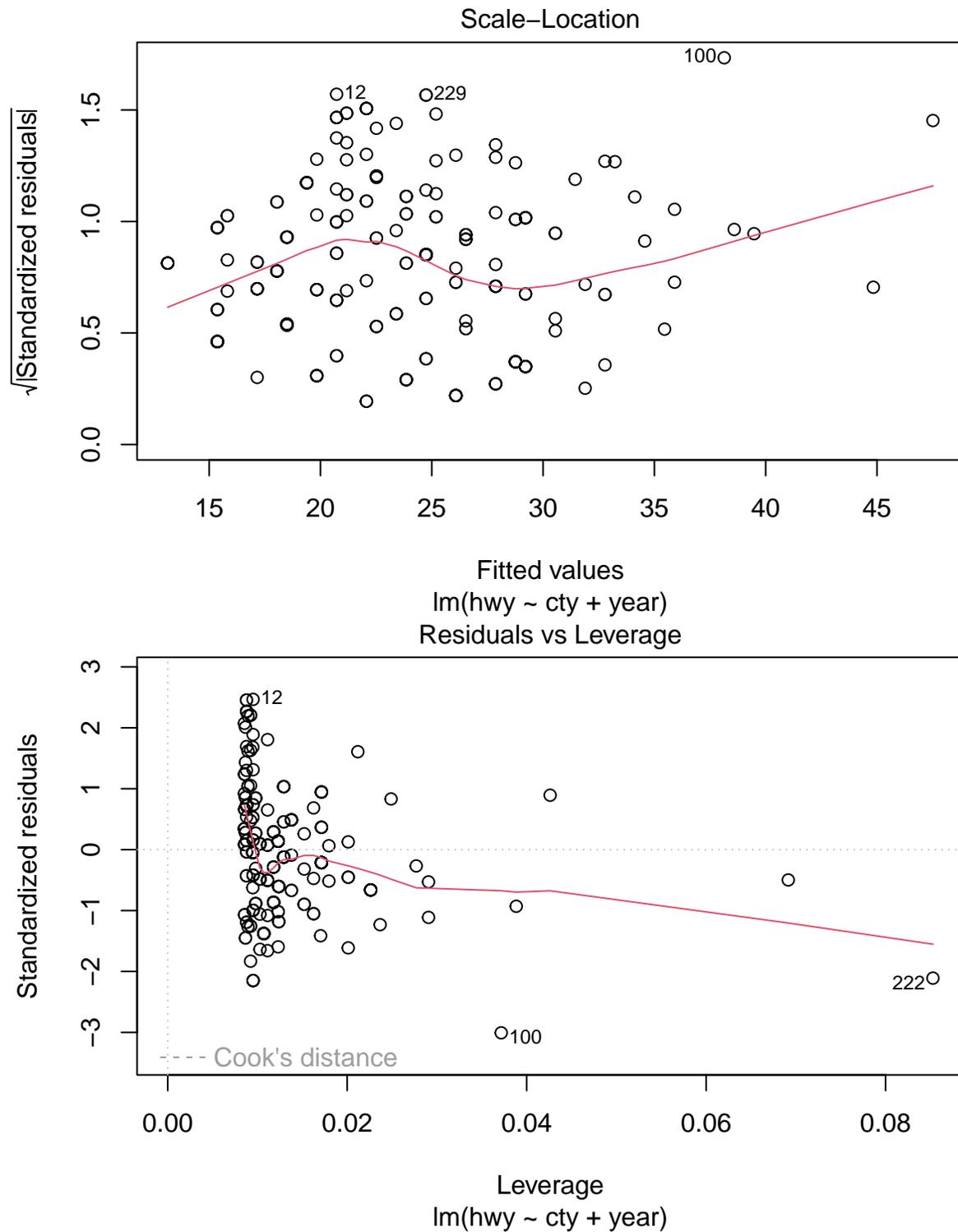
```
new_car <- tibble(cty=25, year=2001)
predict(car_model_mult, newdata=new_car)
```

```
##      1
## 34.21968
```

Since we have more than 1 predictor, we can no longer look at a scatterplot to visually judge whether a linear regression is appropriate. However, the residual plots examined at the end of Lab 7 still work. Recall these plots place the residuals on the y axis and the regression line predictions on the x axis, and any noticeable patterns in the residual plot suggest poor fit of the model.

```
plot(car_model_mult)
```





Now let's add another predictor, `drv`. Recall `drv` is categorical with 3 possible values: 4, f, and r. Last time, we made this a binary predictor; however, we now know how to deal with general categorical predictors, using dummy variables.

```
car_model_cat <- lm(hwy ~ cty + year + drv, data=mpg)
car_model_cat$coefficients
```

```
## (Intercept)          cty          year          drvf          drvr
## -110.39952150    1.19763344    0.05610173    2.26035868    2.10672765
```

Notice we have two new coefficients: `drvf` and `drvr`. This indicates 4 is the control level. Then we can interpret the coefficient of `drvf` as follows: *fixing year of manufacture and city mpg*, front wheel drive cars tend to have 2.260 greater highway mpg than 4wd cars.

2. Get the predicted highway mpg for a 4wd car with 25 city mpg manufactured in 2001. Is this higher, lower, or the same as the prediction under our previous model, which did not account for `drvf`?
3. Use `fct_relevel()` to change the control level to `r`. Then refit the model `car_model_cat`, and repeat question 2. Verify that you get the same prediction.

## Nonlinear regression

We will now take a look at the famous Boston housing dataset, in the built-in MASS package.

```
library(MASS)
```

```
##
## Attaching package: 'MASS'
## The following object is masked from 'package:dplyr':
##
##   select
```

```
head(Boston)
```

```
##      crim zn indus chas   nox   rm age   dis rad tax ptratio  black lstat
## 1 0.00632 18  2.31    0 0.538 6.575 65.2 4.0900   1 296    15.3 396.90  4.98
## 2 0.02731  0  7.07    0 0.469 6.421 78.9 4.9671   2 242    17.8 396.90  9.14
## 3 0.02729  0  7.07    0 0.469 7.185 61.1 4.9671   2 242    17.8 392.83  4.03
## 4 0.03237  0  2.18    0 0.458 6.998 45.8 6.0622   3 222    18.7 394.63  2.94
## 5 0.06905  0  2.18    0 0.458 7.147 54.2 6.0622   3 222    18.7 396.90  5.33
## 6 0.02985  0  2.18    0 0.458 6.430 58.7 6.0622   3 222    18.7 394.12  5.21
##   medv
## 1 24.0
## 2 21.6
## 3 34.7
## 4 33.4
## 5 36.2
## 6 28.7
```

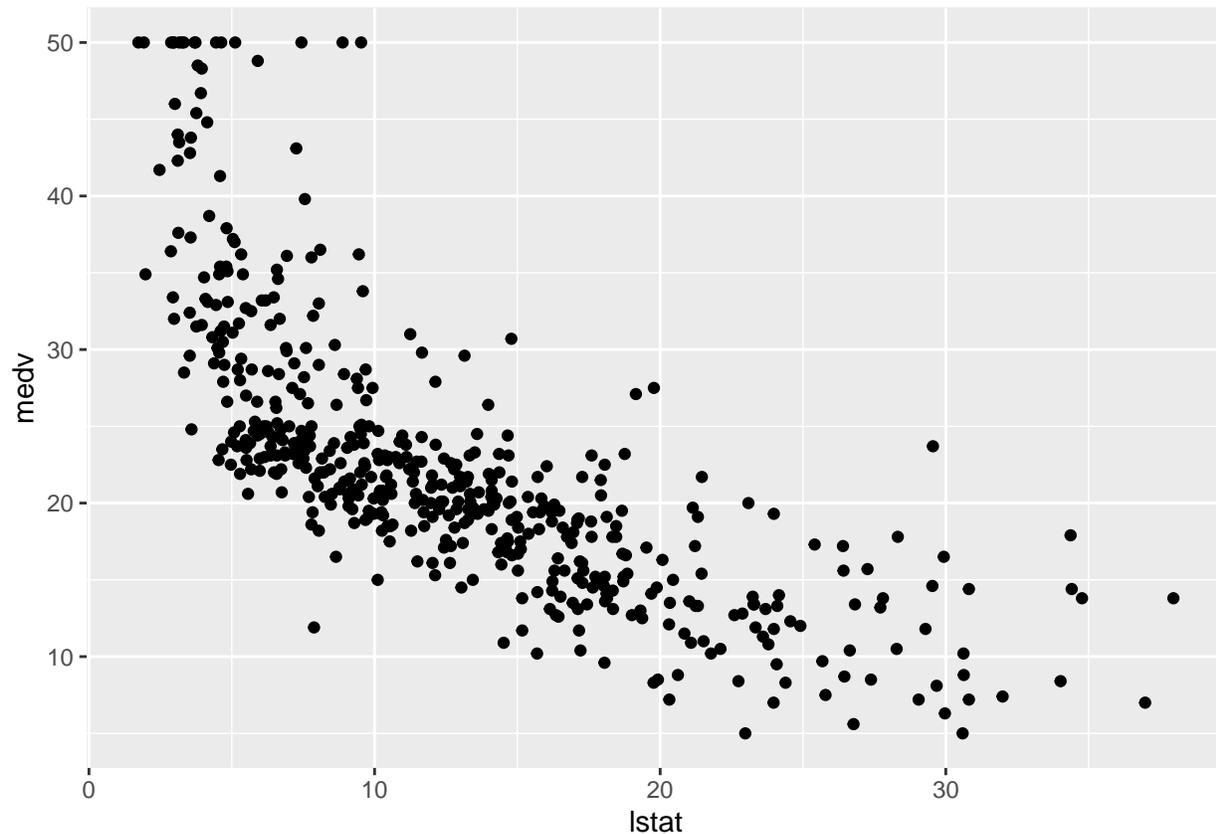
```
str(Boston)
```

```
## 'data.frame':   506 obs. of  14 variables:
##  $ crim   : num  0.00632 0.02731 0.02729 0.03237 0.06905 ...
##  $ zn     : num  18 0 0 0 0 0 12.5 12.5 12.5 12.5 ...
##  $ indus  : num  2.31 7.07 7.07 2.18 2.18 2.18 7.87 7.87 7.87 7.87 ...
##  $ chas   : int   0 0 0 0 0 0 0 0 0 0 ...
##  $ nox    : num  0.538 0.469 0.469 0.458 0.458 0.458 0.524 0.524 0.524 0.524 ...
##  $ rm     : num  6.58 6.42 7.18 7 7.15 ...
##  $ age    : num  65.2 78.9 61.1 45.8 54.2 58.7 66.6 96.1 100 85.9 ...
##  $ dis    : num  4.09 4.97 4.97 6.06 6.06 ...
##  $ rad    : int   1 2 2 3 3 3 5 5 5 5 ...
##  $ tax    : num  296 242 242 222 222 222 311 311 311 311 ...
##  $ ptratio: num  15.3 17.8 17.8 18.7 18.7 18.7 15.2 15.2 15.2 15.2 ...
##  $ black  : num  397 397 393 395 397 ...
```

```
## $ lstat : num 4.98 9.14 4.03 2.94 5.33 ...
## $ medv  : num 24 21.6 34.7 33.4 36.2 28.7 22.9 27.1 16.5 18.9 ...
```

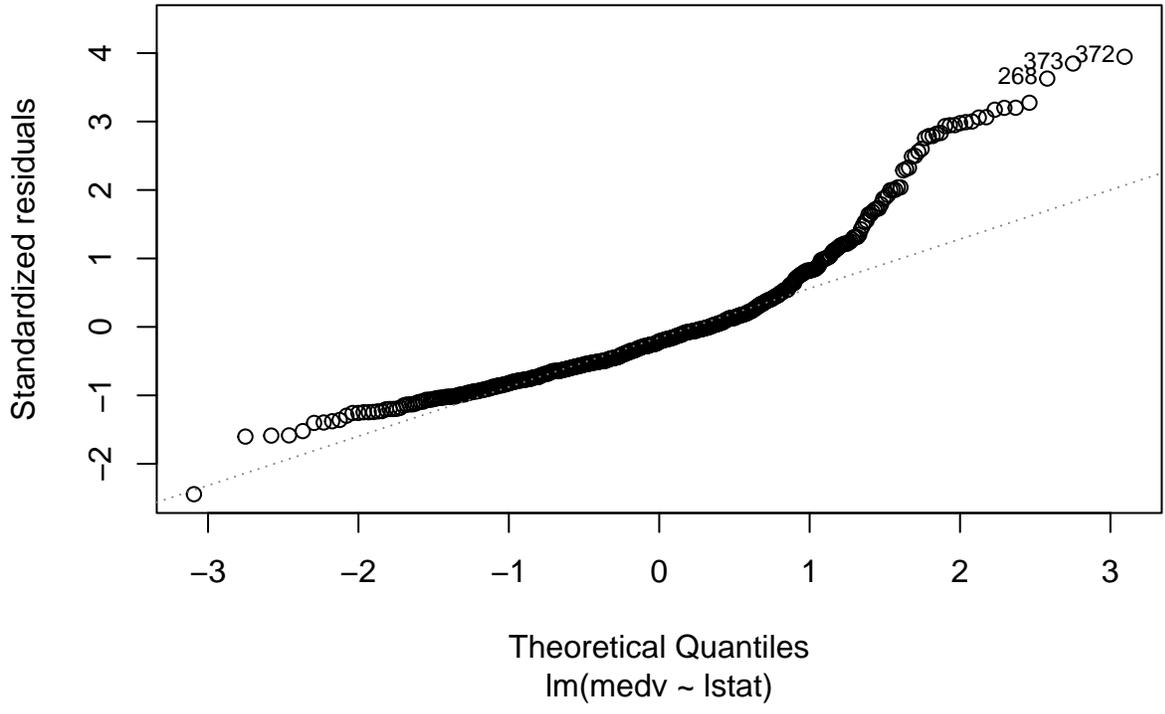
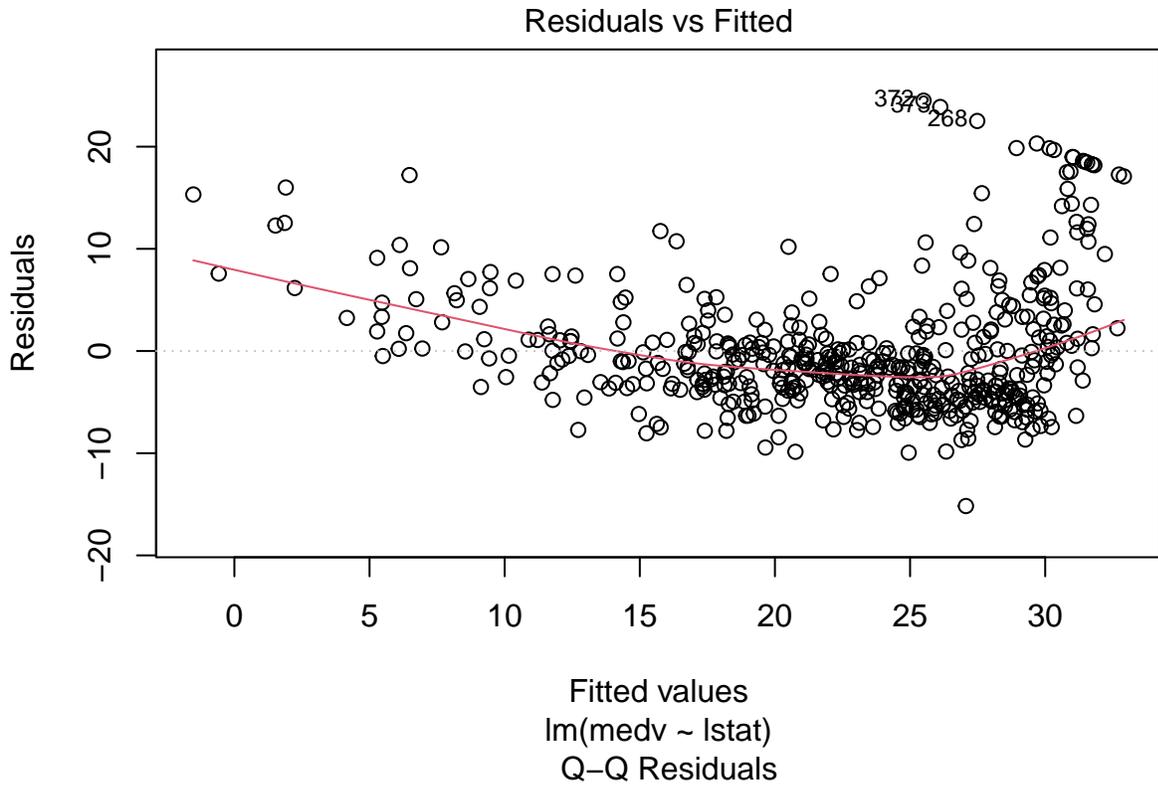
Let's say we want to predict a neighborhood's median home value `medv` with `lstat`, the proportion of lower income residents in that neighborhood. Here's a scatterplot:

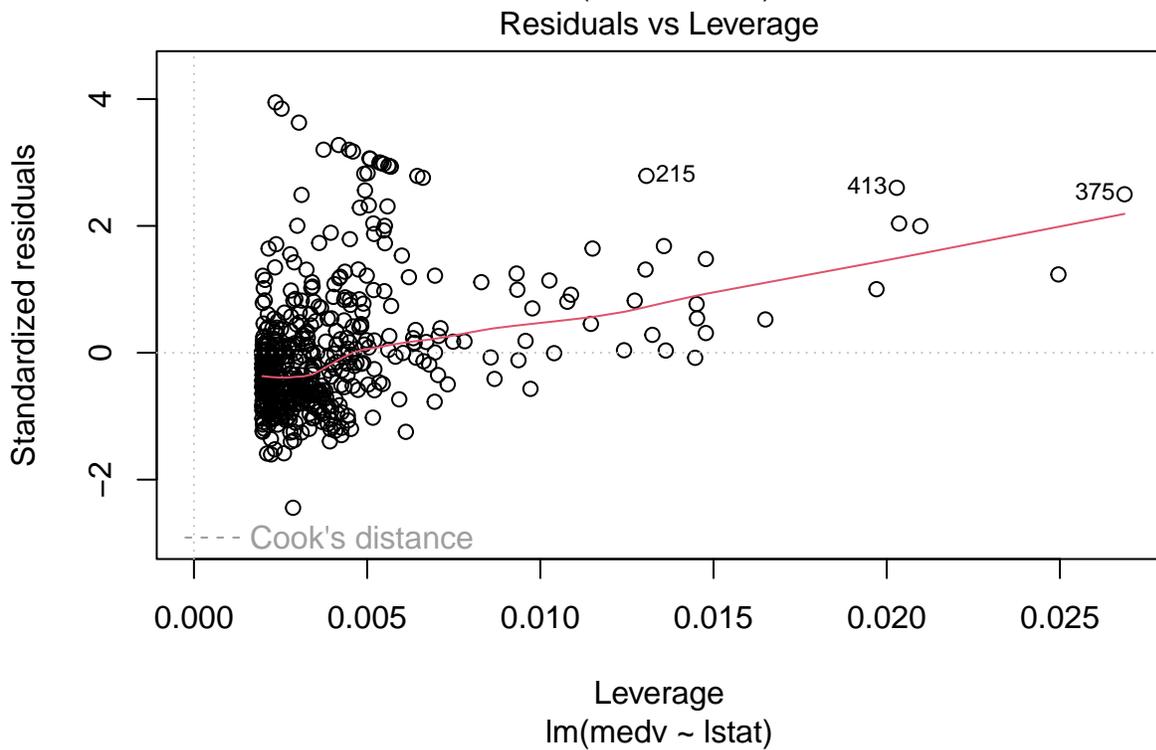
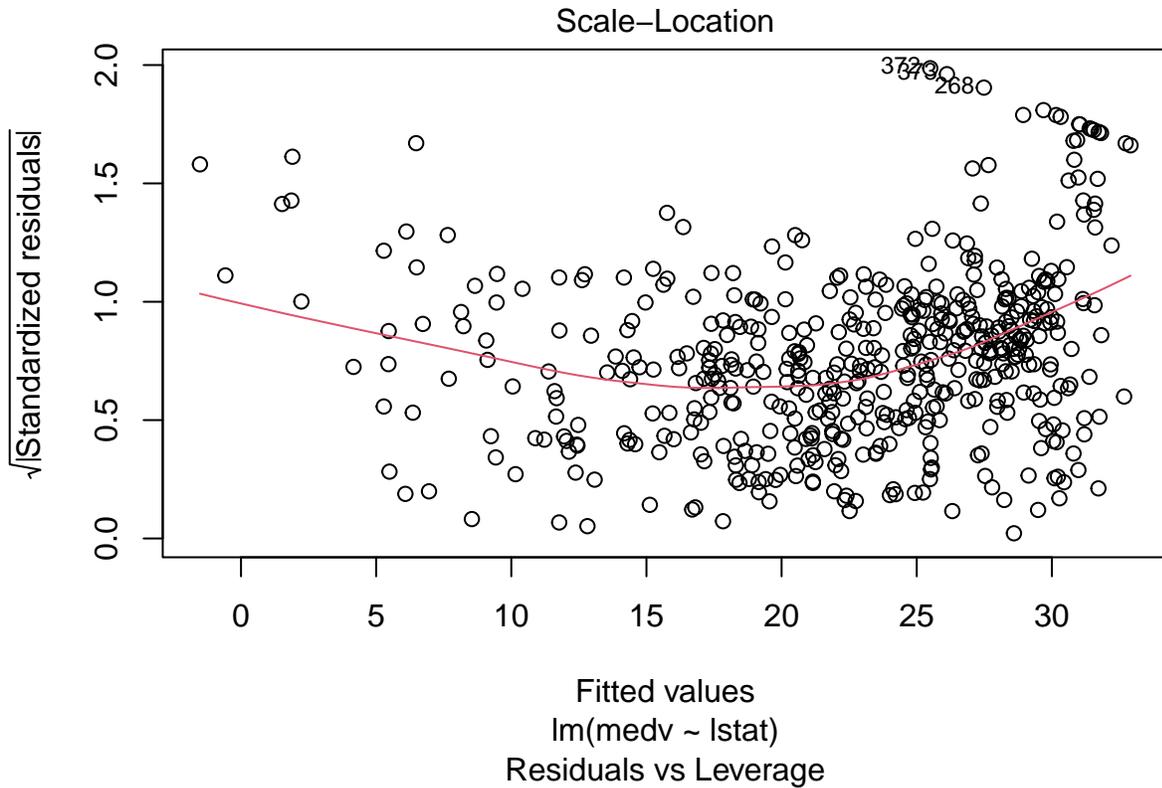
```
Boston %>%
  ggplot(aes(x=lstat, y=medv)) +
  geom_point()
```



It's pretty obvious a linear regression model won't do well here. Let's look at the residual plot to confirm this:

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





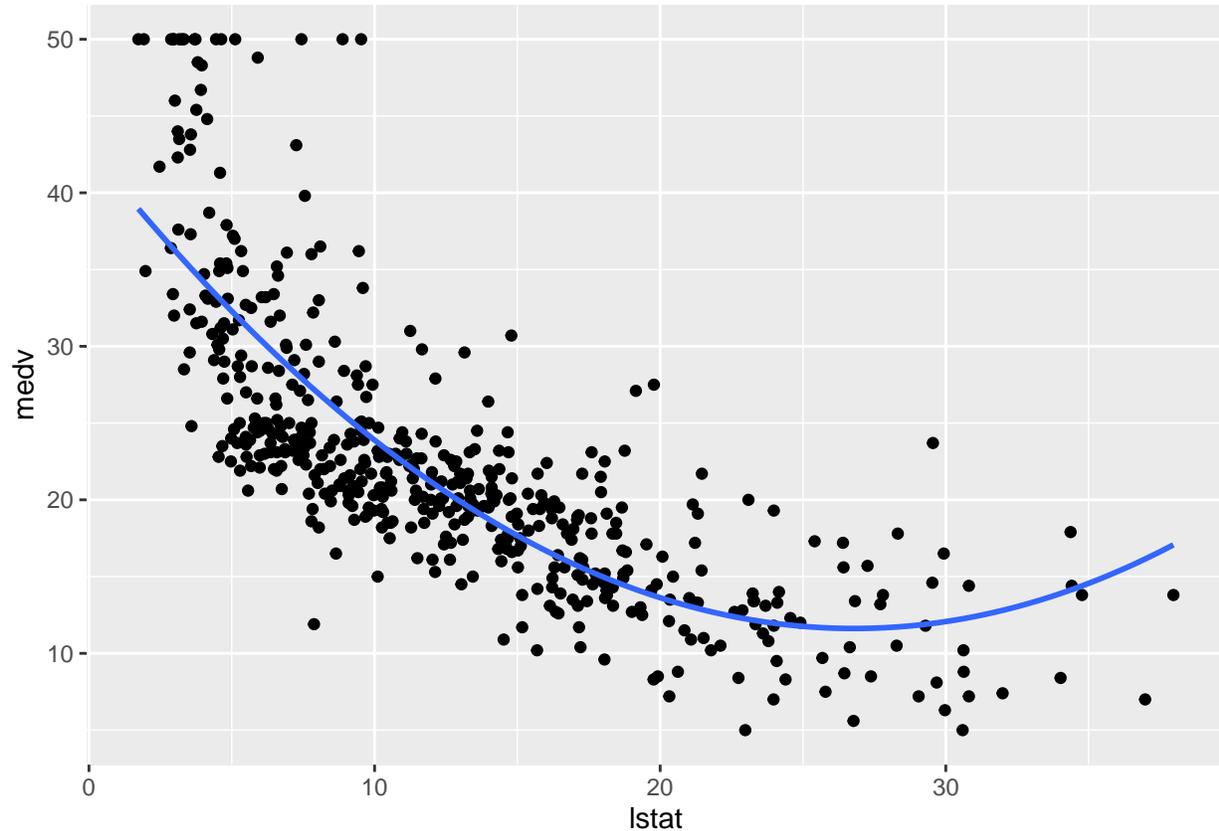
Clearly, there's an increasing trend in the residuals.

Maybe a parabola is a better fit. To do this, we simply do a multiple regression by adding the new predictor  $lstat^2$ . This is done by adding  $I(lstat^2)$  as an additive term on the right side of the formula. The  $I()$  is necessary for  $lm()$  to recognize the  $\wedge$  syntax, denoting the exponent.

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

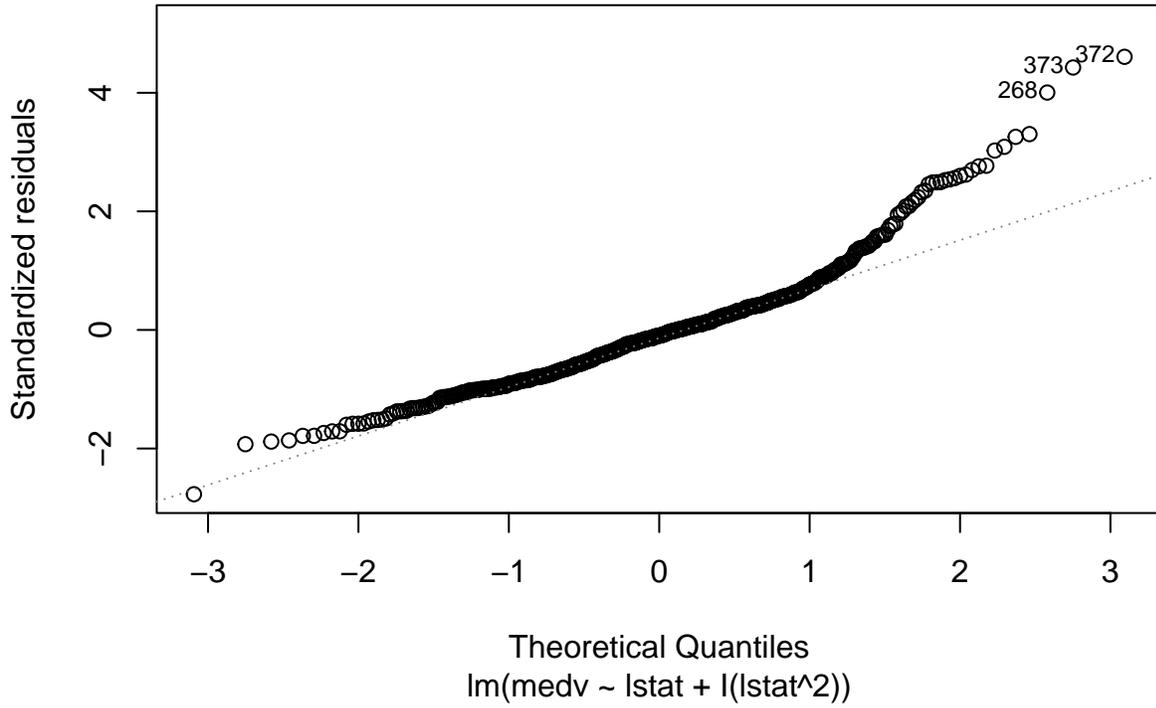
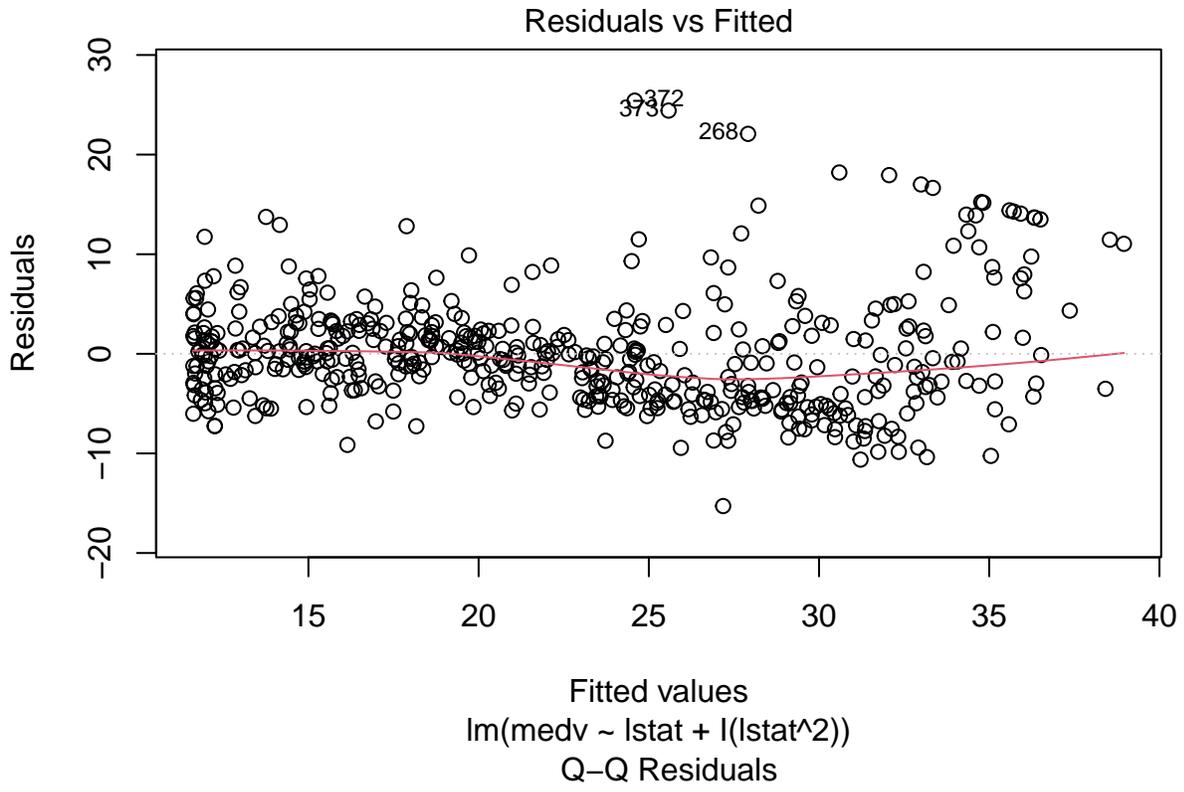
How do we plot this? The easiest way is to use `geom_smooth()`, specifying the `formula` argument with the same formula used in our `lm()` call, replacing `medv` and `lstat` with `y` and `x`, respectively (`geom_smooth()` reads these from the `x` and `y` aesthetics from `ggplot()`. You could override this by specifying different `x` and `y` aesthetics within `aes()` in `geom_smooth()`):

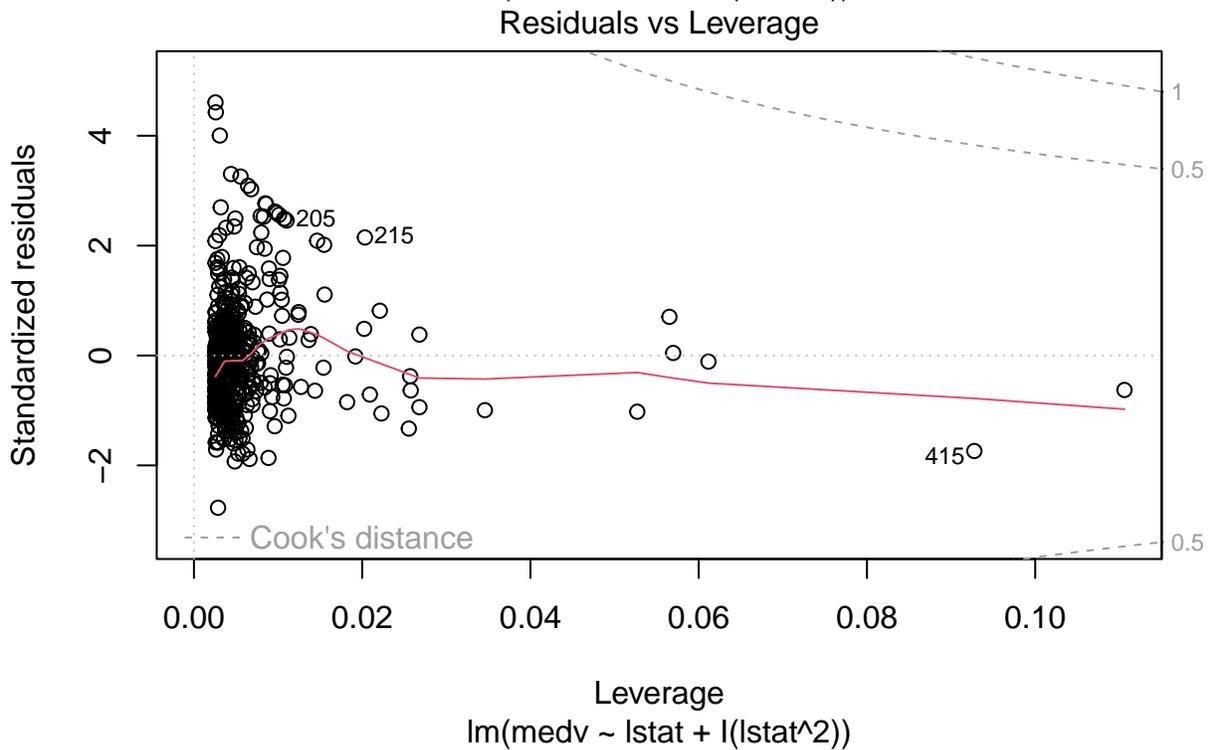
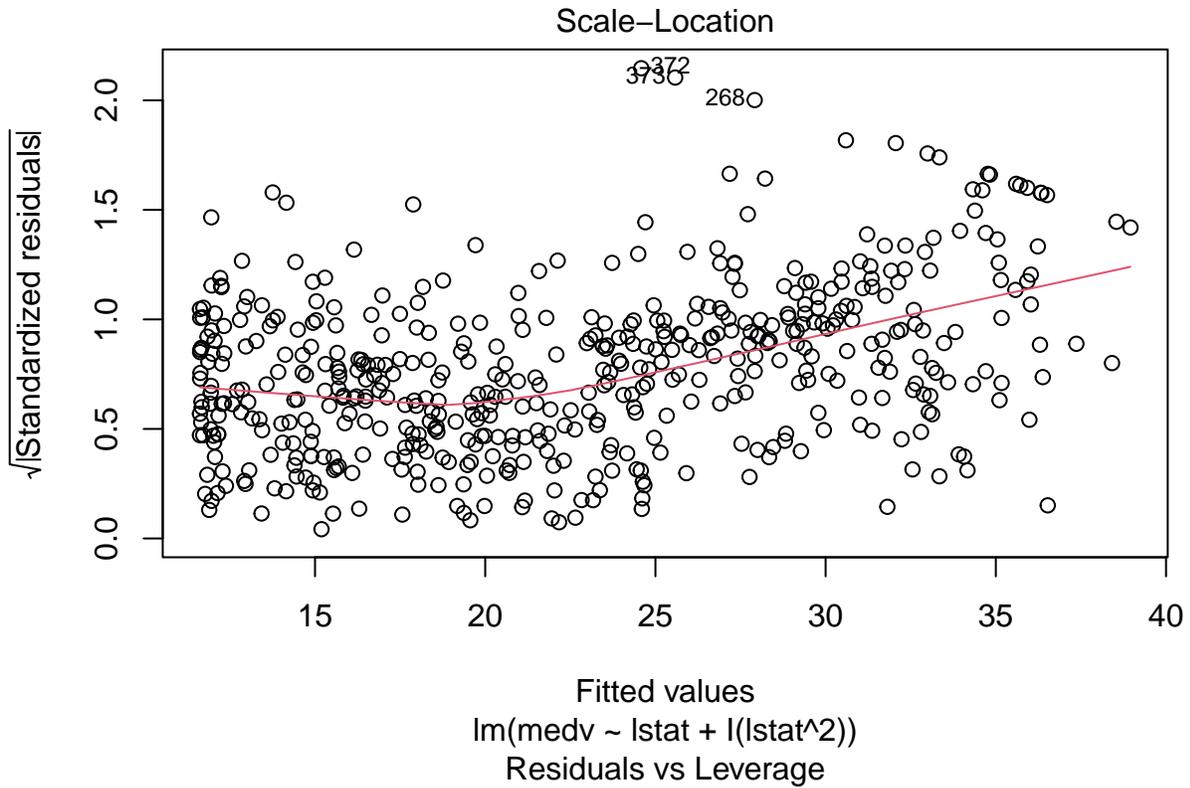
```
Boston %>%  
  ggplot(aes(x=lstat, y=medv)) +  
  geom_point() + # adds scatterplot  
  geom_smooth(method="lm", formula=y ~ x + I(x^2), se=FALSE)
```



This looks a bit better, although it seems like it might tend to under-predict the home values of some of the low `lstat` homes. Again we can look at the residual plot:

```
plot(boston_quadratic_model)
```





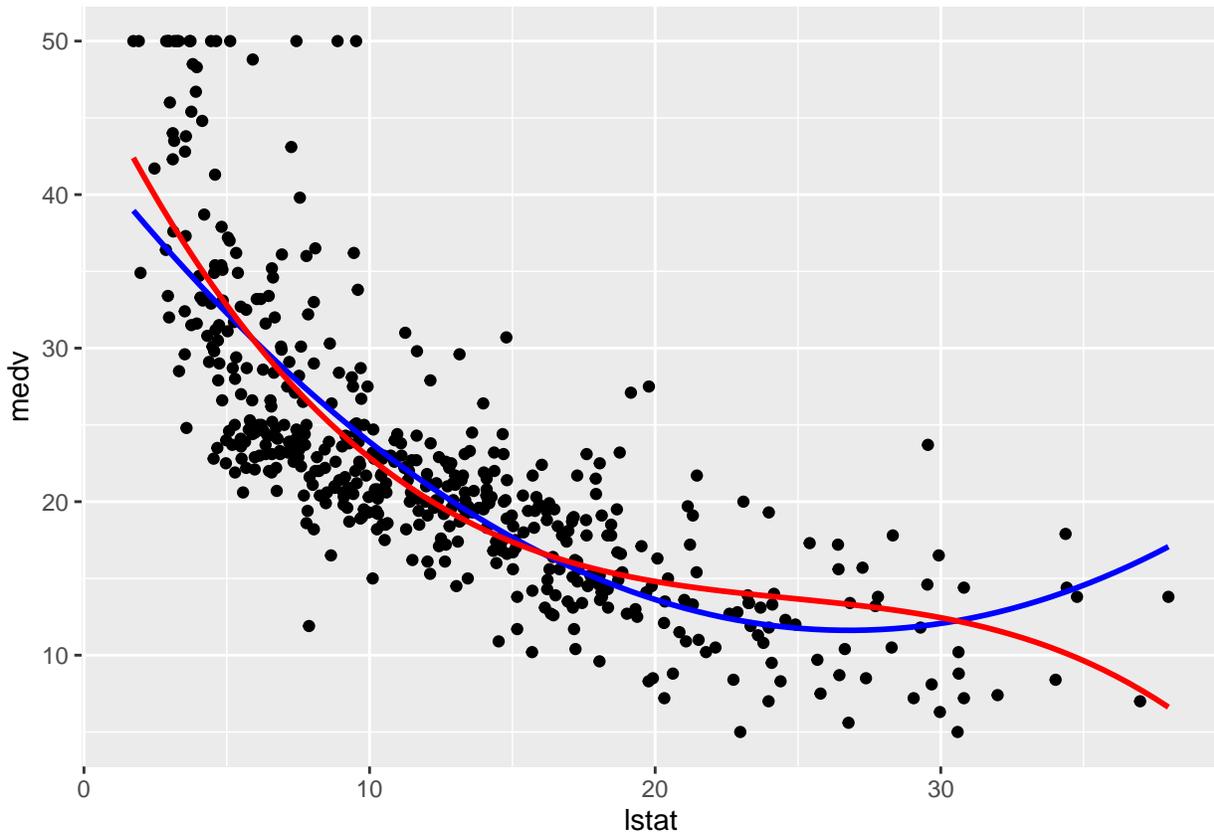
This looks not too bad. There's no clear pattern I can discern.

Like with other multiple regression models, we can use `predict()` to get a prediction for our quadratic regression model.

4. Compare the predictions for `boston_linear_model()` and `boston_quadratic_model()` for a neighborhood with `lstat=30`.

What happens if we try fitting a 3rd degree polynomial?

```
Boston %>%  
  ggplot(aes(x=lstat, y=medv)) +  
  geom_point() +  
  geom_smooth(method="lm", formula=y ~ x + I(x^2), color="blue", se=FALSE) +  
  geom_smooth(method="lm", formula=y ~ x + I(x^2) + I(x^3), color="red", se=FALSE)
```



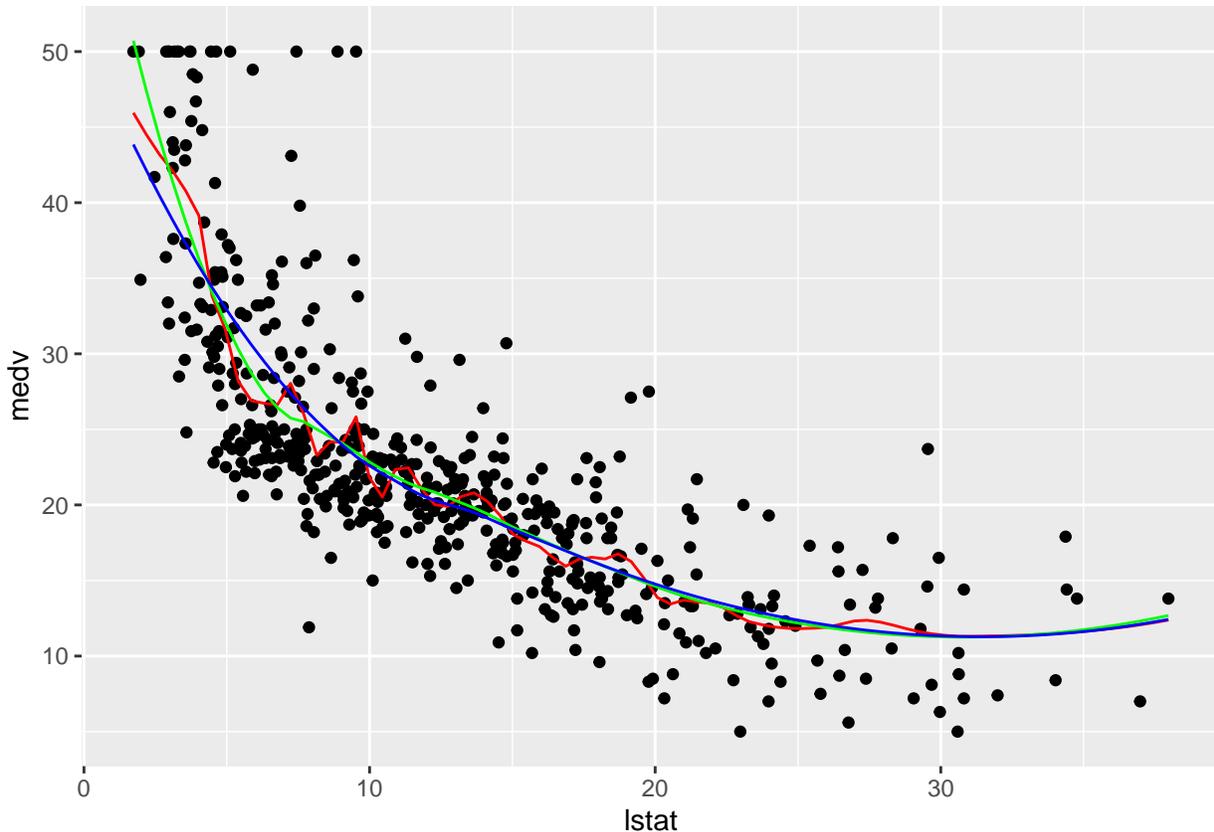
The third degree polynomial, in blue (note the `color` arguments to `geom_smooth` are specified outside of `aes()`) seems broadly similar to the quadratic in red, except when `lstat > 30`. For those values, the quadratic curves up while the cubic curves down. Visually, it seems like the predictions should instead be flat around there.

Now let's see what happens with locally weighted regression. While locally weighted regression lets us fit the most flexible curve, there are two main drawbacks:

- No explicit formula for prediction (must be done numerically)
- Need to choose `span` parameter appropriately to avoid overfitting

Let's look at the loess curves for some different `span`s:

```
Boston %>%  
  ggplot(aes(x=lstat, y=medv)) +  
  geom_point() +  
  geom_smooth(method="loess", formula=y ~ x, span=0.1, colour="red", lwd=0.5, se=FALSE) +  
  geom_smooth(method="loess", formula=y ~ x, span=0.5, colour="green", lwd=0.5, se=FALSE) +  
  geom_smooth(method="loess", formula=y ~ x, span=0.9, colour="blue", lwd=0.5, se=FALSE)
```



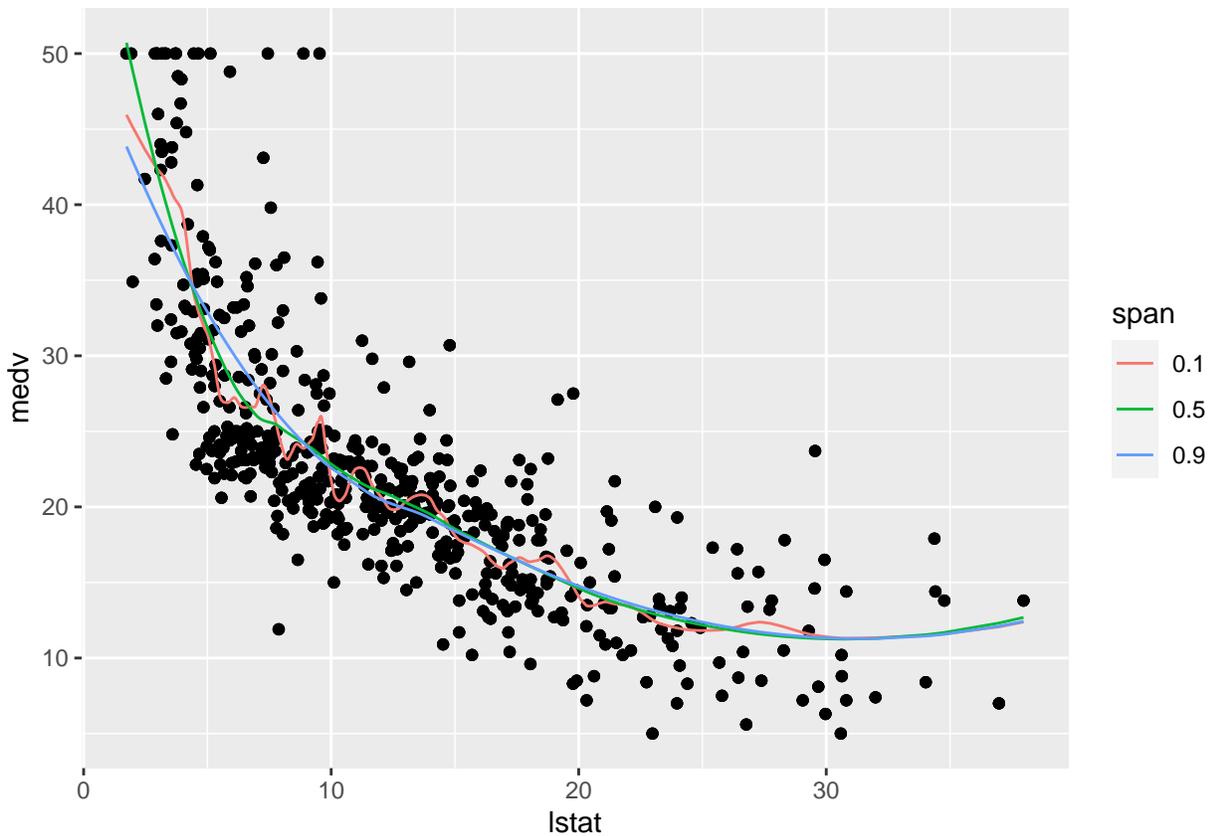
## Adding a legend

It turns out not to be super intuitive to figure out how to add a legend showing the colors corresponding to each `span` using `ggplot2`. The issue is that legends in `ggplot2` are tied to aesthetics within geoms (think of the legends automatically created in Lab 5).

Given you want to plot the prediction from  $q$  different locally weighted regression models with different `span` values, one solution would be to add  $q$  columns to your dataset, each containing the predictions from one of your models. Then you can use `pivot_longer()` to modify the tibble so that there is only one column of predictions but now an additional column `span` specifying the which span the prediction is from. Then use `geom_line()` to plot the predictions, and make sure the aesthetic (e.g. color) varies based on `span`.

```
model_0.1 <- loess(medv ~ lstat, span=0.1, data=Boston)
model_0.5 <- loess(medv ~ lstat, span=0.5, data=Boston)
model_0.9 <- loess(medv ~ lstat, span=0.9, data=Boston)
```

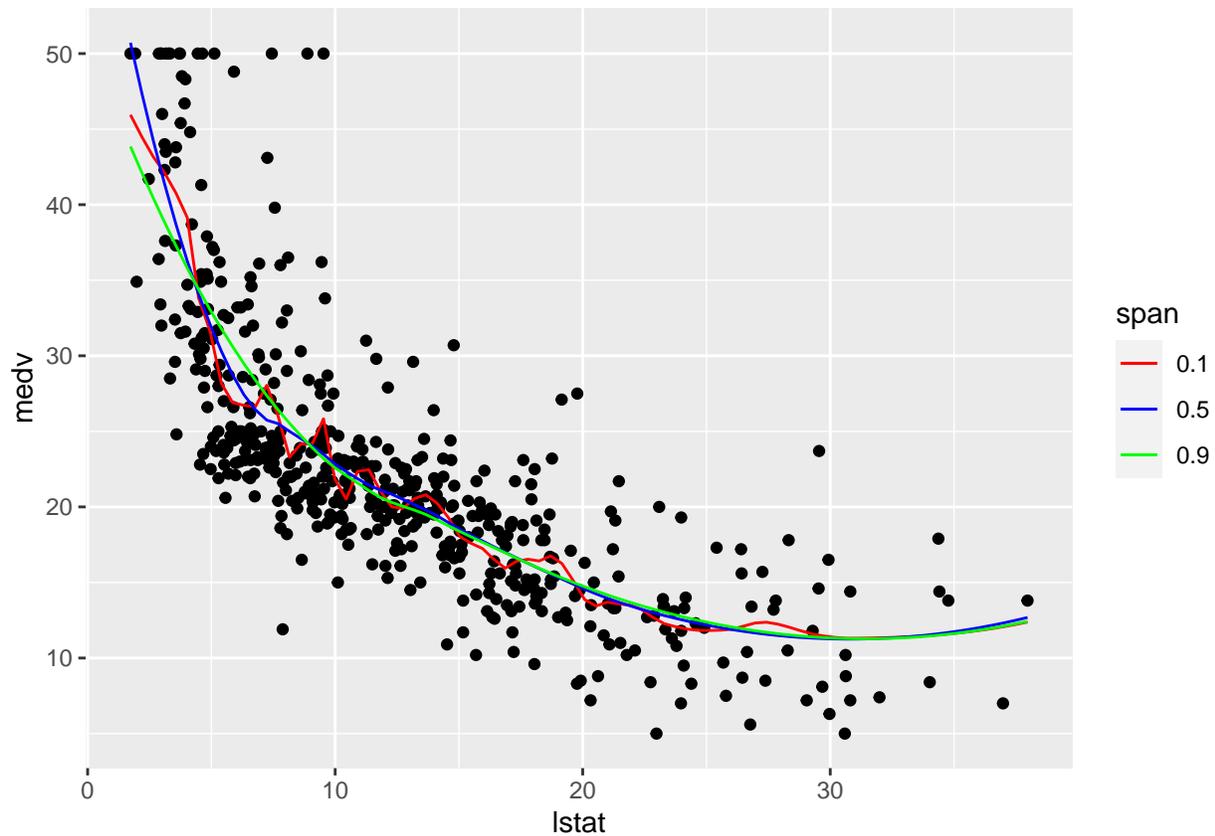
```
Boston %>%
  mutate("0.1"=predict(object=model_0.1),
         "0.5"=predict(object=model_0.5),
         "0.9"=predict(object=model_0.9)) %>%
  pivot_longer(cols=c("0.1", "0.5", "0.9"),
              names_to="span",
              values_to="predictions") %>%
  ggplot(aes(x=lstat, y=medv)) +
  geom_point() +
  geom_line(aes(x=lstat, y=predictions, colour=span))
```



5. Explain what each line of code in the chunk above does.

Another way of creating a legend that lets you use `geom_smooth()` is given below. It uses `scale_color_manual()` to define a custom color palette (this solution is taken from <https://aosmith.rbind.io/2018/07/19/manual-legends-ggplot2/>):

```
Boston %>%
  ggplot(aes(x=lstat, y=medv)) +
  geom_point() +
  geom_smooth(method="loess", formula=y ~ x, span=0.1, aes(color="0.1"), lwd=0.5, se=FALSE) +
  geom_smooth(method="loess", formula=y ~ x, span=0.5, aes(color="0.5"), lwd=0.5, se=FALSE) +
  geom_smooth(method="loess", formula=y ~ x, span=0.9, aes(color="0.9"), lwd=0.5, se=FALSE) +
  scale_color_manual(name="span",
                     breaks=c("0.1", "0.5", "0.9"),
                     values=c("0.1"="red", "0.5"="blue", "0.9"="green"))
```



## Back to locally weighted regression

We clearly see that lowering `span` leads to “wigglier” curves. In this example, it seems visually that a smoother curve is best.

Even though locally weighted regression curves are fit using `loess()` rather than `lm()`, the same `predict()` syntax works:

```
boston_loess_model <- loess(formula=medv ~ lstat, span=0.9, data=Boston)
predict(boston_loess_model, newdata=tibble(lstat=30))
```

```
##          1
## 11.32957
```

However, note there are no longer any coefficients, since the predictions from loess models do not take a simple mathematical form:

```
boston_loess_model$coefficients
```

```
## NULL
```