Chapter 3 Lab

Mohammed Ali

January 4, 2018

Contents

Libraries

Load needed libraries

```
library (MASS) #which is a very large collection of data sets and functions
library (ISLR) #which includes the data sets associated with this book
```

```
## Warning: package 'ISLR' was built under R version 3.4.3
```

```
library(dplyr) #for data set manuplations
```

```
## Warning: package 'dplyr' was built under R version 3.4.2
```

```
library(ggplot2) #for visualization
```

```
## Warning: package 'ggplot2' was built under R version 3.4.2
```

```
library(ggfortify) #for lm visualization
```

```
## Warning: namespace 'DBI' is not available and has been replaced
## by .GlobalEnv when processing object 'call.'
```

```
## Warning: namespace 'DBI' is not available and has been replaced
## by .GlobalEnv when processing object 'call.'
```

```
library(epiDisplay) # for enhanced display for the model
```

```
## Warning: package 'epiDisplay' was built under R version 3.4.3
```

```
library(car) #for lm visualization
```

```
## Warning: package 'car' was built under R version 3.4.3
```

Simple Linear Regression

The *MASS* library contains the `Boston` data set, which records `medv` (median house value) for 506 neighborhoods around Boston. We will seek to predict `medv` using 13 predictors such as `rm` (average number of rooms per house), age (average age of houses), and lstat (percent of households with low socioeconomic status).

```
glimpse(Boston)
```

```
## Observations: 506
## Variables: 14
## $ crim    <dbl> 0.00632, 0.02731, 0.02729, 0.03237, 0.06905, 0.02985, ...
## $ zn      <dbl> 18.0, 0.0, 0.0, 0.0, 0.0, 0.0, 12.5, 12.5, 12.5, 12.5,...
## $ indus   <dbl> 2.31, 7.07, 7.07, 2.18, 2.18, 2.18, 7.87, 7.87, 7.87, ...
## $ chas    <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ nox     <dbl> 0.538, 0.469, 0.469, 0.458, 0.458, 0.458, 0.524, 0.524...
## $ rm      <dbl> 6.575, 6.421, 7.185, 6.998, 7.147, 6.430, 6.012, 6.172...
## $ age     <dbl> 65.2, 78.9, 61.1, 45.8, 54.2, 58.7, 66.6, 96.1, 100.0,...
## $ dis     <dbl> 4.0900, 4.9671, 4.9671, 6.0622, 6.0622, 6.0622, 5.5605...
## $ rad     <int> 1, 2, 2, 3, 3, 3, 5, 5, 5, 5, 5, 5, 5, 4, 4, 4, 4, 4, ...
## $ tax     <dbl> 296, 242, 242, 222, 222, 222, 311, 311, 311, 311, 311,...
## $ ptratio <dbl> 15.3, 17.8, 17.8, 18.7, 18.7, 18.7, 15.2, 15.2, 15.2, ...
## $ black   <dbl> 396.90, 396.90, 392.83, 394.63, 396.90, 394.12, 395.60...
## $ lstat   <dbl> 4.98, 9.14, 4.03, 2.94, 5.33, 5.21, 12.43, 19.15, 29.9...
## $ medv    <dbl> 24.0, 21.6, 34.7, 33.4, 36.2, 28.7, 22.9, 27.1, 16.5, ...
```

We will start by using the `lm` function to fit a simple linear regression model, with `medv` as the response and `lstat` as the predictor. The basic syntax is `lm(y ~ x,data)`, where y is the response, x is the predictor, and *data* is the data set in which these two variables are kept.

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

If we type `lm.fit`, some basic information about the model is output. For more detailed information, we use `summary(lm.fit)`. This gives us *p-values* and *standard errors* for the coefficients, as well as the R^2 statistic and *F-statistic* for the model.

```
lm.fit
```

```
##
## Call:
## lm(formula = medv ~ lstat, data = Boston)
##
## Coefficients:
## (Intercept)        lstat
##       34.55        -0.95
```

```
summary(lm.fit)
```

```
##
## 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
```

Let us remember the interpretation.

Residual

Difference between what the model predicted and the actual value of y. We can calculate the Residuals section like so:

```
summary(lm.fit$residuals)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -15.167  -3.990  -1.318   0.000   2.034  24.500
```

Residual Standard Error

Essentially standard deviation of residuals / errors of your regression model. *Standard deviation* is the square root of *variance*. Standard Error is very similar. The only difference is that instead of dividing by `n-1`, you subtract `n minus 1 + # of variables involved`.

```
#Residual Standard error (Like Standard Deviation)
k <- length(lm.fit$coefficients)-1 #Subtract one to ignore intercept
SSE <- sum(lm.fit$residuals**2)
n <- length(lm.fit$residuals)
sqrt(SSE/(n-(1+k))) #Residual Standard Error
```

```
## [1] 6.21576
```

It is the same value as above

Coefficients

These are the weights that minimize the sum of the square of the errors.
Std. Error –> is Residual Standard Error (see below) divided by the square root of the sum of the square of that particular x variable.

t value –> Estimate divided by *Std. Error*

Pr(>|t|) –> Look up your t value in a T distribution table with the given degrees of freedom.

Multiple R-Squared

Percent of the variance of Y intact after subtracting the error of the model. Also called the *coefficient of determination*, this is an oft-cited measurement of how well our model fits to the data. While there are many issues with using it alone, it's a quick and pre-computed check for our model.

R-Squared subtracts the residual error from the variance in Y. The bigger the error, the worse the remaining variance will appear.

```
#Multiple R-Squared (Coefficient of Determination)
SSyy <- sum((Boston$medv - mean(Boston$medv))**2)
SSE <- sum(lm.fit$residuals**2)
(SSyy-SSE)/SSyy
```

```
## [1] 0.5441463
```

```
#Alternatively
1-SSE/SSyy
```

```
## [1] 0.5441463
```

Same as above. Please note numerator doesn't have to be positive. If the model is so bad, you can actually end up with a negative R-Squared.

Adjusted R-Squared

Same as multiple *R-Squared* but takes into account the number of samples and variables we're using. *Multiple R-Squared* works great for simple linear (one variable) regression. However, in most cases, the model has multiple variables. The more variables we add, the more variance we're going to explain. So we have to control for the extra variables.

Adjusted R-Squared normalizes Multiple R-Squared by taking into account how many samples you have and how many variables we're using.

```
#Adjusted R-Squared
n <- length(Boston$medv)
k <- length(lm.fit$coefficients) - 1 #Subtract one to ignore intercept
SSE <- sum(lm.fit$residuals ** 2)
SSyy <- sum((Boston$medv - mean(Boston$medv)) ** 2)
1-(SSE/SSyy)*(n-1)/(n-(k+1))
```

```
## [1] 0.5432418
```

Same as above. A larger normalizing value is going to make the Adjusted R-Squared worse since we're subtracting its product from one.

F-Statistic

Global test to check if your model has at least one significant variable. Takes into account number of variables and observations used. Including the t-tests, this is the second *test* that the summary function produces for lm models.

```
#F-Statistic
#Ho: All coefficients are zero
#Ha: At least one coefficient is nonzero
#Compare test statistic to F Distribution table
n <- length(Boston$medv)
SSE <- sum(lm.fit$residuals**2)
SSyy <- sum((Boston$medv-mean(Boston$medv))**2)
k <- length(lm.fit$coefficients)-1
((SSyy-SSE)/k) / (SSE/(n-(k+1)))
```

```
## [1] 601.6179
```

Same as above.

p-value

The *p-value* for each term tests the null hypothesis that the coefficient is equal to zero (no effect). A low `p-value (< 0.05)` indicates that you can reject the null hypothesis. In other words, a predictor that has a low *p-value* is likely to be a meaningful addition to your model because changes in the predictor's value are related to changes in the response variable.

Conversely, a larger (insignificant) *p-value* suggests that changes in the predictor are not associated with changes in the response. It seems our predictor above is fine.

Other fields

We can use *R* built in `names()` function to know all `lm.fit` model properties

```
names(lm.fit)
```

```
##  [1] "coefficients"  "residuals"     "effects"       "rank"
##  [5] "fitted.values" "assign"        "qr"            "df.residual"
##  [9] "xlevels"       "call"          "terms"         "model"
```

and we can used *extractor* methods to get their values (i.e `coef` method)

```
confint(lm.fit)
```

```
##                 2.5 %     97.5 %
## (Intercept) 33.448457 35.6592247
## lstat       -1.026148 -0.8739505
```

predict

The `predict()` function can be used to produce confidence intervals and prediction intervals for the prediction of `medv` for a given value of `lstat`.

```
predict (lm.fit ,data.frame(lstat=c(5 ,10 ,15) ), interval ="confidence")
```

```
##        fit      lwr      upr
## 1 29.80359 29.00741 30.59978
## 2 25.05335 24.47413 25.63256
## 3 20.30310 19.73159 20.87461
```

```
predict (lm.fit ,data.frame(lstat=c(5 ,10 ,15) ), interval ="prediction")
```

```
##        fit       lwr      upr
## 1 29.80359 17.565675 42.04151
## 2 25.05335 12.827626 37.27907
## 3 20.30310  8.077742 32.52846
```

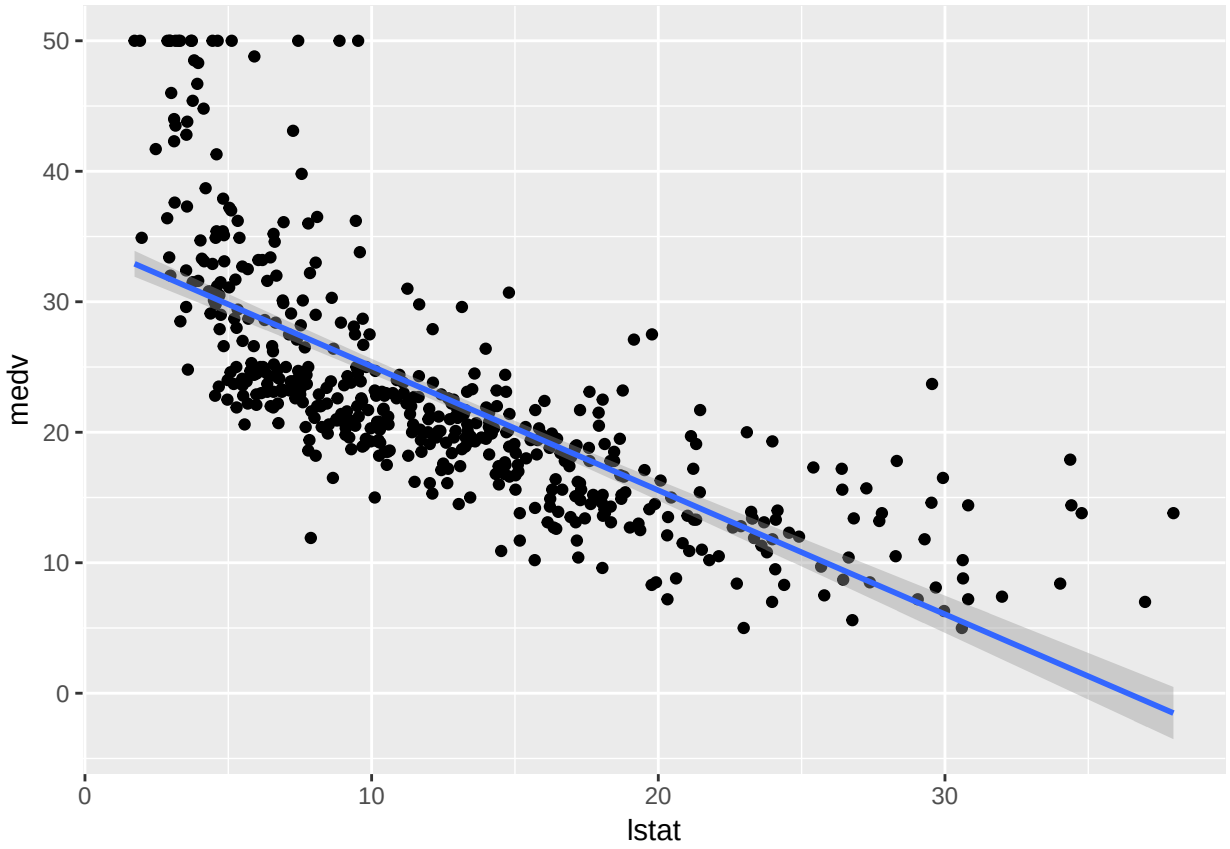
For instance, the 95% confidence interval associated with a `lstat` value of `10` is `(24.47, 25.63)`, and the 95% prediction interval is `(12.828, 37.28)`. As expected, the confidence and prediction intervals are centered around the same point (a predicted value of `25.05` for medv when `lstat` equals 10), but the latter are substantially wider.

Visualization

I will just use `ggplot` instead of the built-in plotting system in `R` for its more enhanced results and better data manipulation

95% confidence interval

```
ggplot(Boston, aes(x = lstat, y = medv)) +
  geom_point() +
  geom_smooth(method=lm, se=TRUE) # Add linear regression line
```

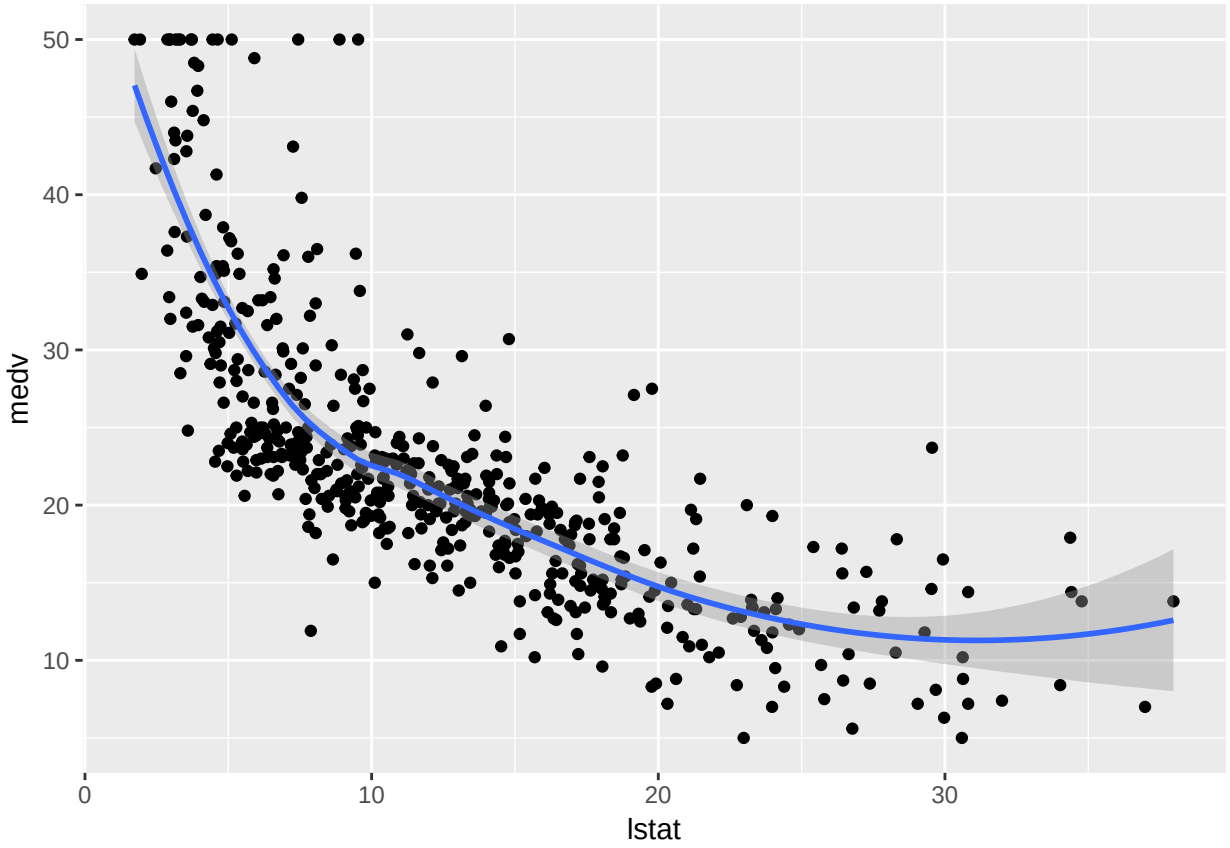



```
                                #(by default includes 95% confidence region)
                                # Add shaded confidence region
```

It does not seem that linear model is the best fit, let us try *loose* method

```
ggplot(Boston, aes(x = lstat, y = medv)) +
  geom_point() +
  geom_smooth(se=TRUE) # Add linear regression line
```

```
## `geom_smooth()` using method = 'loess'
```




```
                                #(by default includes 95% confidence region)
                                # Add shaded confidence region
```

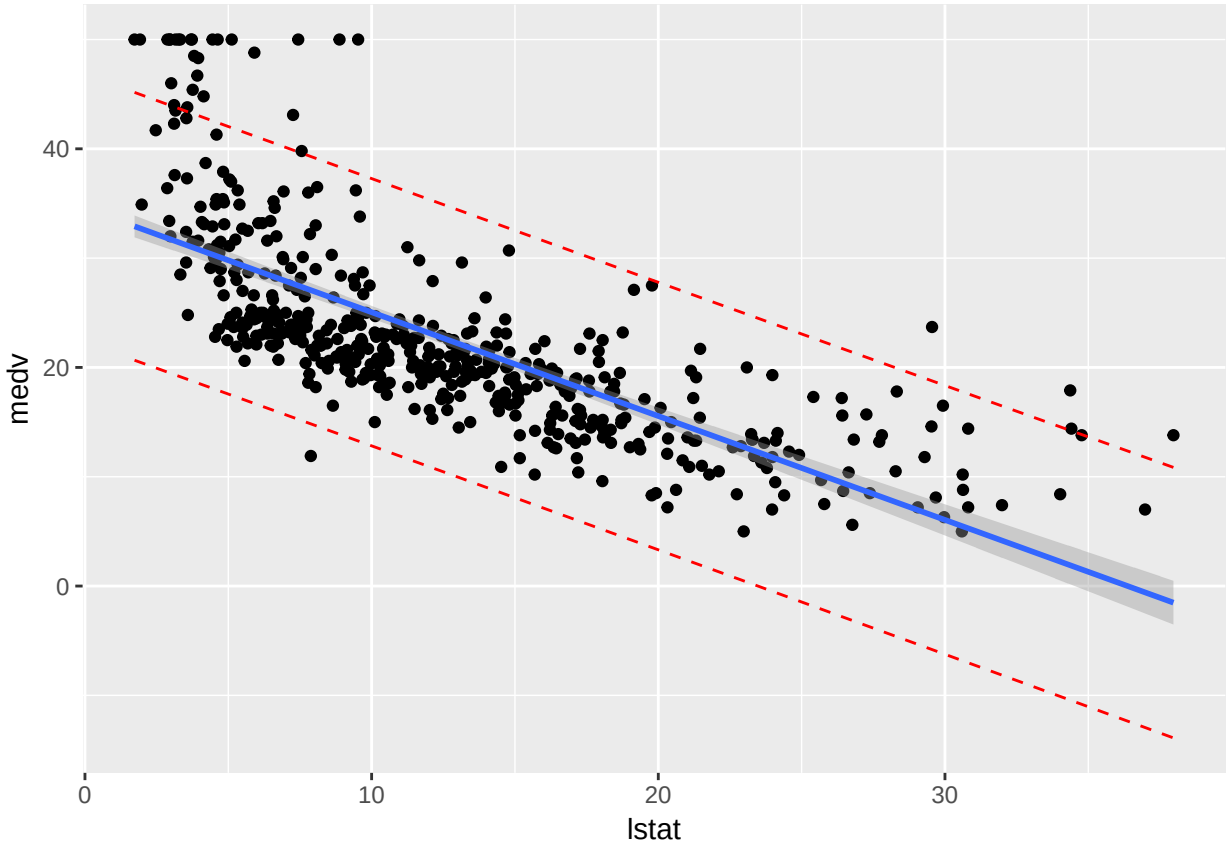
yeah, a little better.

95% confidence and prediction intervals

```
predict <- predict (lm.fit, data = Boston$lstat, interval ="prediction")
```

```
## Warning in predict.lm(lm.fit, data = Boston$lstat, interval = "prediction"): predictions on current
```

```
all_data <- cbind(Boston, predict)
ggplot(all_data, aes(x = lstat, y = medv))+
    geom_point() +
    geom_line(aes(y=lwr), color = "red", linetype = "dashed")+
    geom_line(aes(y=upr), color = "red", linetype = "dashed")+
    geom_smooth(method=lm, se=TRUE)
```

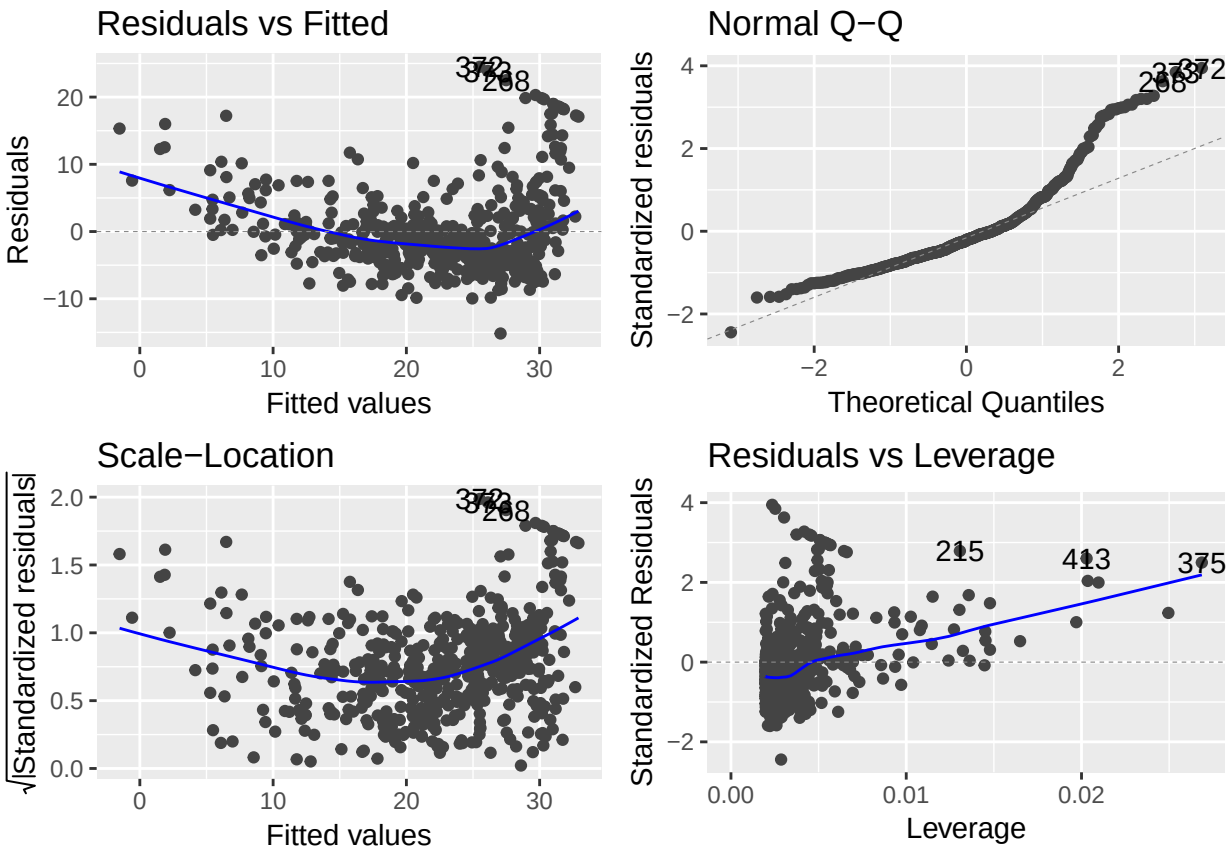


model vosualization

residual vs fitted plot interpretation

```
autoplot(lm.fit)
```

```
## Warning: package 'bindrcpp' was built under R version 3.4.2
```



References * http://www.learnbymarketing.com/tutorials/explaining-the-lm-summary-in-r/ * [How to Interpret Regression Analysis Results: P-values and Coefficients] http://blog.minitab.com/blog/adventures-in-statistics-2/how-to-interpret-regression-analysis-results-p-values-and-coefficients *https://rpubs.com/kaz_yos/car-residuals