Lab 2: Linear Regression in R

Yifan Jin

9/16/2020

Import dataset

We'll work with the Auto dataset from the Lab 1 exercise.

```
library(ISLR)

data(Auto)
# what are the variables?
names(Auto)
```

```
## [1] "mpg"          "cylinders"    "displacement" "horsepower"
## [5] "weight"       "acceleration" "year"         "origin"
## [9] "name"
```

The ISLR package provides a more detailed description if you type `?Auto` in the console.

We'll begin by fitting a simple linear regression to predict mpg given horsepower using the lm function.

Fitting simple linear regression with `lm()`

```
# the basic syntax is lm(y~x,data), which produces a linear model object
lm.fit <- lm(mpg~horsepower, Auto)
```

Alternatively, you can specify your dataset by using `attach()`:

```
attach(Auto)

lm.fit <- lm(mpg~horsepower)
```

Inspect fitted model

We can see basic output by printing `lm.fit`. More detailed output is given by calling `summary(lm.fit)`, this gives us standard errors and p-values for the coefficients, as well as R^2 statistic and F-statistic for the model.

```
# basic lm output
lm.fit
```

```
##
## Call:
## lm(formula = mpg ~ horsepower)
##
## Coefficients:
## (Intercept)   horsepower
##     39.9359      -0.1578
```

```
# detailed lm output
summary(lm.fit)
```

```
##
## Call:
## lm(formula = mpg ~ horsepower)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -13.5710  -3.2592  -0.3435   2.7630  16.9240
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 39.935861   0.717499   55.66   <2e-16 ***
## horsepower  -0.157845   0.006446  -24.49   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.906 on 390 degrees of freedom
## Multiple R-squared:  0.6059, Adjusted R-squared:  0.6049
## F-statistic: 599.7 on 1 and 390 DF,  p-value: < 2.2e-16
```

Some words on standard errors

Example with a simple linear regression:

```r
#------generate one data set with epsilon ~ N(0, 0.25)------
seed <- 1152 #seed
n <- 100     #nb of observations
a <- 5       #intercept
b <- 2.7     #slope

set.seed(seed)
epsilon <- rnorm(n, mean=0, sd=sqrt(0.25))
x <- sample(x=c(0, 1), size=n, replace=TRUE)
y <- a + b * x + epsilon
```

```r
#------using lm------
mod <- lm(y ~ x)
```

```r
#------using the explicit formulas------
X <- cbind(1, x)
betaHat <- solve(t(X) %*% X) %*% t(X) %*% y
var_betaHat <- anova(mod)[[3]][2] * solve(t(X) %*% X)
anova(mod)
```

```
## Analysis of Variance Table
##
## Response: y
##           Df  Sum Sq Mean Sq F value    Pr(>F)
## x          1 188.615 188.615  802.82 < 2.2e-16 ***
## Residuals 98  23.024   0.235
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
#------comparison------
#estimate
mod$coef
```

```
## (Intercept)           x
##    5.020261    2.755577
```

```r
c(betaHat[1], betaHat[2])
```

```
## [1] 5.020261 2.755577
```

```r
#standard error
summary(mod)$coefficients[, 2]
```

```
## (Intercept)           x
##  0.06596021  0.09725302
```

```r
sqrt(diag(var_betaHat))
```

```
##                       x
## 0.06596021 0.09725302
```

Mean squared error(MSE)

MSE is an estimator for σ^2. In general, it is equal to

$$\sigma^2 = \frac{\sum(Y_i - \hat{Y}_i)^2}{n-p}$$

where p is the rank of the projection matrix $X(X^TX)^{-1}X^T$.

t-statistics

The t-statistics can be computed as $t_i = \frac{\hat{\beta}_i}{\hat{\sigma}_i}$. It is a measure of how many standard deviations our coefficient estimate is far away from 0(based on the hypothesis $H_0 : \beta_i = 0$ and assumptions that ε_i are independent and identical.)

$Pr(> |t|)$

It is the probability of observing any value equal or larger than t.

R^2

The R^2 is computed as

$$R^2 = \frac{SSE}{SST} = \frac{\sum_i(\hat{y}_i - \bar{y})^2}{\sum_i(y_i - \bar{y})^2}$$

where SSE stands for explained sum of square, SST stands for total sum of square. Note R^2 is always between 0 and 1. And $SST = SSE + SSR$, where $SSR = \sum_i(y_i - \hat{y}_i)^2$(Proof?). R^2 is a measure of the linear relationship between our predictor variable and our response(i.e.: a number near 0 represents a regression that does not explain the variance in the response variable well and a number close to 1 does explain the observed variance in the response variable).

Adjusted R^2

Adjusted R^2 is computed as

$$1 - \frac{SSR/(n-p-1)}{SST/(n-1)} = 1 - (1-R^2)\frac{n-1}{n-p-1}$$

The adjusted R^2 is the same thing as R^2, but adjusted for the complexity (i.e. the number of parameters) of the model. Given a model with a single parameter, with a certain R^2, if we add another parameter to this model, the R^2 of the new model has to increase, even if the added parameter has no statistical power. The adjusted R^2 accounts for this by including the number of parameters in the model.

F-statistics

The F-statistics is the ratio of two variances

$$F = \frac{SSR}{SSE}$$

In our Auto data example, under null hypothesis of no effect(no relationship between mpg and horsepower), F-statistics follows a F distribution with degree of freedom 1 and 390. A large F-statistics(low p value) incicates we can reject the null hypothesis.

We can use the `names()` function in order to find out what other pieces of information are stored in `lm.fit`.

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

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

Although we can extract these quantities by name:

```
lm.fit$coefficients
```

```
## (Intercept)  horsepower
##  39.9358610  -0.1578447
```

it is safer to use the extractor functions like `coef()`

```
coef(lm.fit)
```

```
## (Intercept)  horsepower
##  39.9358610  -0.1578447
```

R will automatically provide confidence intervals for the parameters of the fitted model, and predictions for new data

```
# confidence intervals for coefficient vector
confint(lm.fit)
```

```
##                 2.5 %     97.5 %
## (Intercept) 38.525212 41.3465103
## horsepower  -0.170517 -0.1451725
```

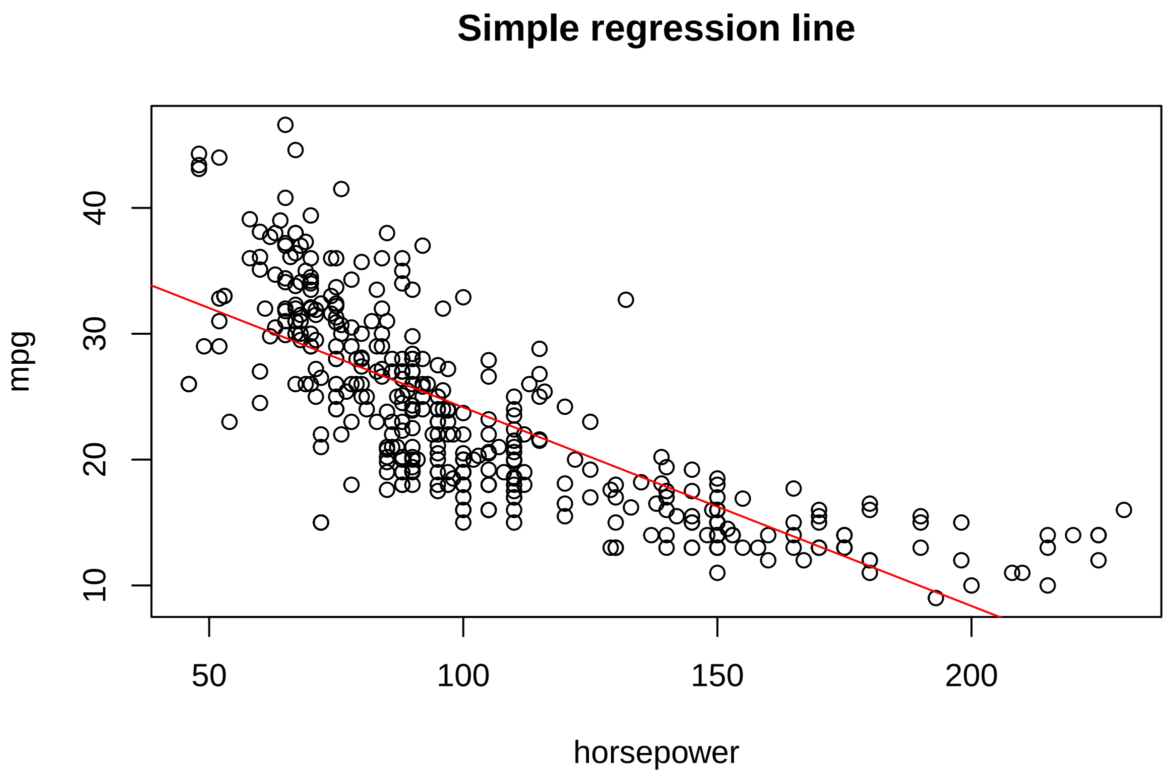
```
# predictions for new data (with error bounds)
predict(lm.fit,data.frame(horsepower=c(100,120,200)),interval="prediction")
```

```
##         fit       lwr      upr
## 1 24.151388 14.493888 33.80889
## 2 20.994493 11.335155 30.65383
## 3  8.366914 -1.365999 18.09983
```

Plots and visual diagnostics

We can plot the fitted regression line with the abline function.

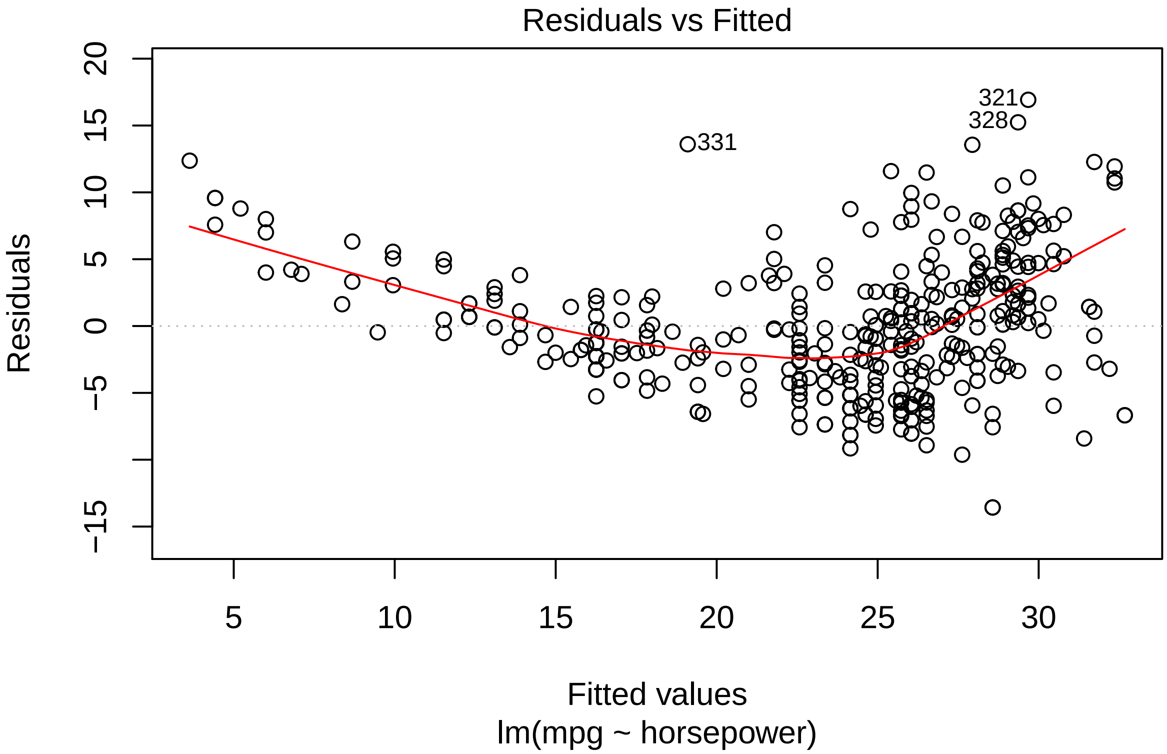
```r
# scatter plot of horsepower and mpg
plot(horsepower,mpg,main="Simple regression line")
# add regression line
abline(lm.fit,col="red")
```

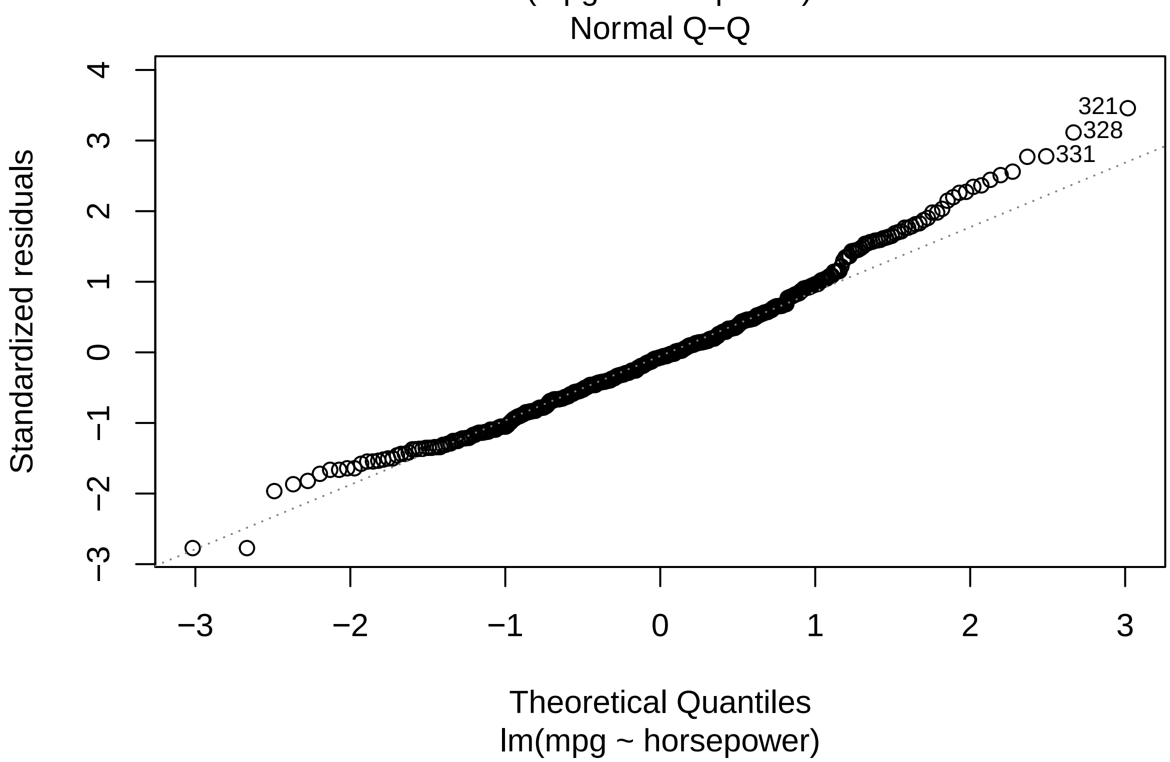


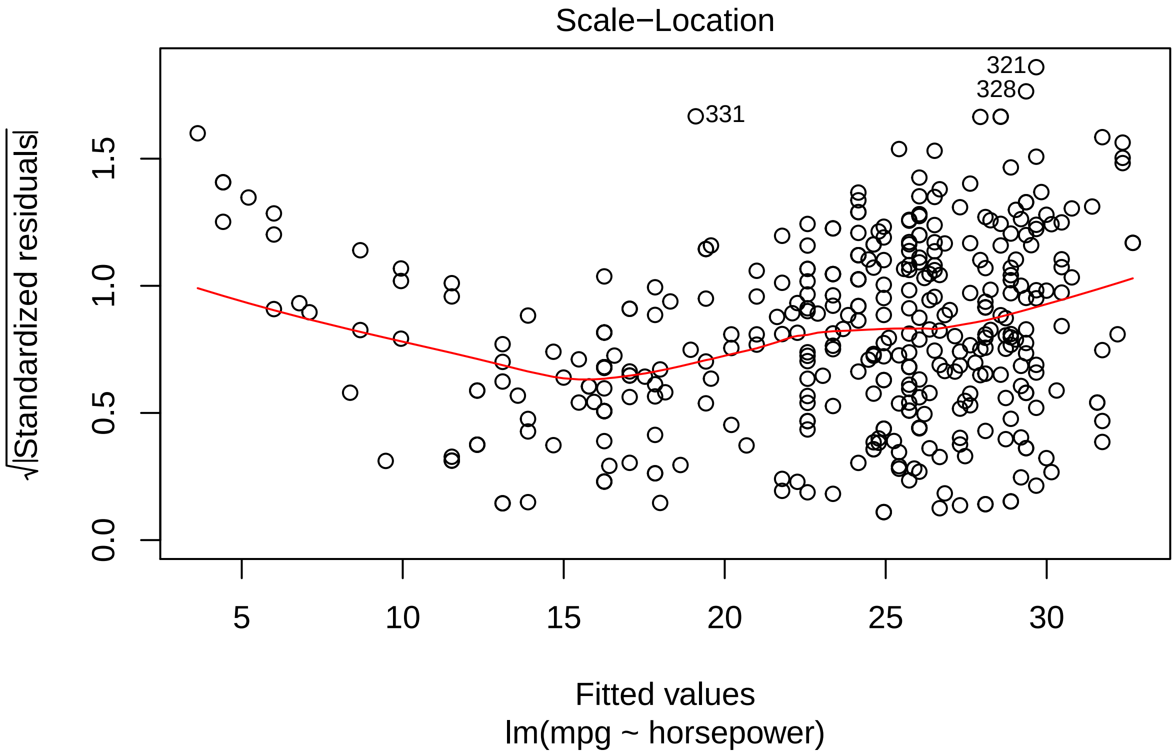
R also produces four basic diagnostic plots with `plot()`:

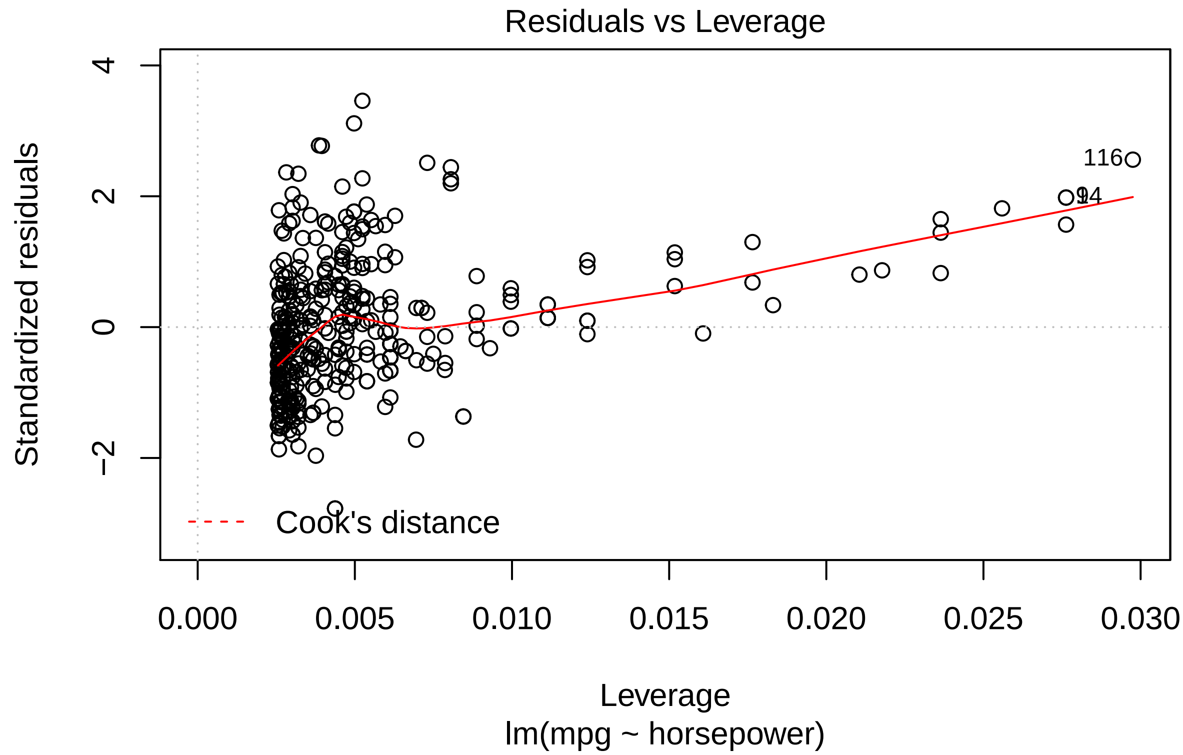
- Residuals vs fitted values
- Residual QQ-plot
- Standardized residuals vs fitted values
- Standardized resiudals vs leverage

```r
# diagnostic plots
plot(lm.fit)
```


Residuals vs Fitted
Residuals
Fitted values
lm(mpg ~ horsepower)
321
328
331


Normal Q−Q
Standardized residuals
Theoretical Quantiles
lm(mpg ~ horsepower)
321
328
331





In general, this command will produce one plot at a time, and hitting Enter will generate the next plot. However, it is often convenient to view all four plots together.

```
# set plot options so all four plots appear in the same pane
par(mfrow=c(2,2))
```

We can produce these plots ourselves by using the objects produced by `lm()`, for instance suppose we want to see the residuals vs fitted values plot.

```
# residuals v fitted values plot
y.hat <- predict(lm.fit)
e.hat <- residuals(lm.fit)
plot(y.hat,e.hat,
     xlab="Fitted values",ylab="Residuals",
     main="Residuals v Fitted values")
abline(h=0,lty=3)
```