Resampling Methods

We are going to use the Auto data to illustrate the results of various resampling methods, so lets load it from the `ISLR` package and explore.

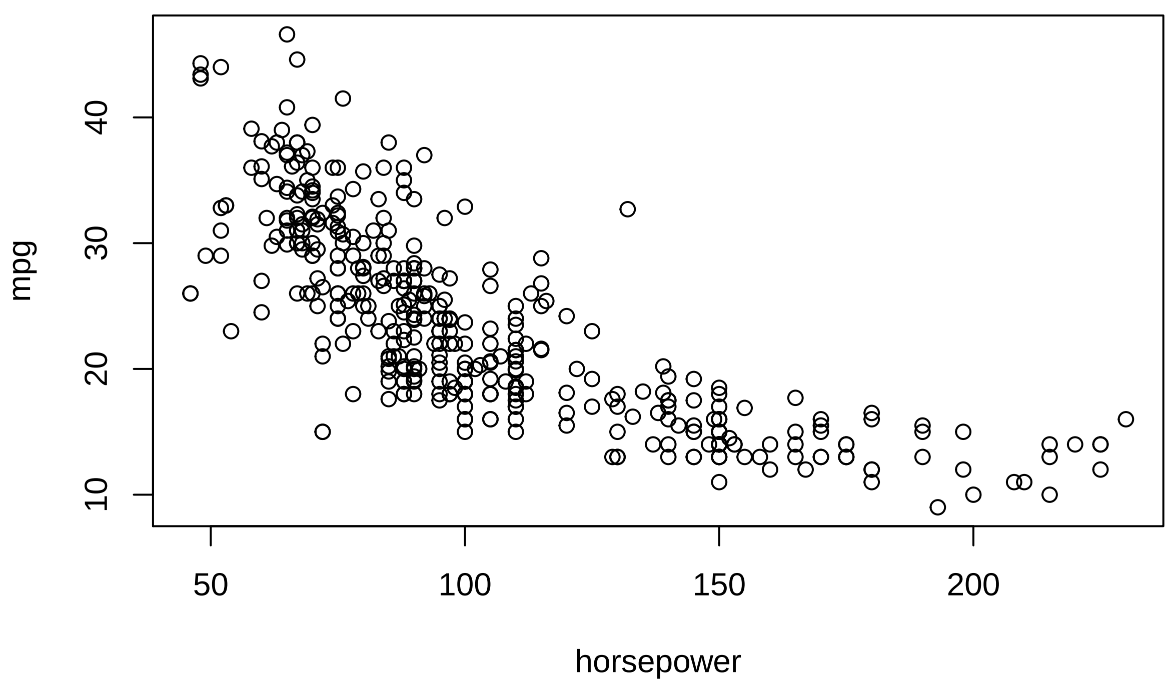
```
library(ISLR)
data(Auto)
```

```
str(Auto[, -9])
```

```
## 'data.frame':    392 obs. of  8 variables:
##  $ mpg         : num  18 15 18 16 17 15 14 14 14 15 ...
##  $ cylinders   : num  8 8 8 8 8 8 8 8 8 8 ...
##  $ displacement: num  307 350 318 304 302 429 454 440 455 390 ...
##  $ horsepower  : num  130 165 150 150 140 198 220 215 225 190 ...
##  $ weight      : num  3504 3693 3436 3433 3449 ...
##  $ acceleration: num  12 11.5 11 12 10.5 10 9 8.5 10 8.5 ...
##  $ year        : num  70 70 70 70 70 70 70 70 70 70 ...
##  $ origin      : num  1 1 1 1 1 1 1 1 1 1 ...
```

A plot is always a nice place to start with a new data set.

```
plot(mpg ~ horsepower, data = Auto)
```



As is exploring the available documentation.

```
?Auto
```

The Leave-One-Out Cross-Validation (LCOOV) method.

First, lets run a `glm` model on the `Auto` data set.

```
glm_auto <- glm(mpg ~ horsepower, data = Auto)
```

Next, load the `boot` package and check out the documentation for the Cross-validation for Generalized Linear Models function, or `cv.glm`.

```
library(boot)
```

```
?cv.glm
```

Then, apply `cv.glm` function to the `Auto` data set, using `glm_auto` model, returning the delta parameter.

```
cv.glm(Auto, glm_auto)$delta
```

```
## [1] 24.23151 24.23114
```

We can speed up the results by writing a function to use the formula displayed in section 5.2 (pg. 180) and then pass the `glm_auto` model to it.

```
loocv <- function(x){
          h <- lm.influence(x)$h
        mean((residuals(x)/(1-h))^2)
}
```

Is our new function faster? We can use the `system.time` function to compare both methods.

```
system.time(
cv.glm(Auto, glm_auto)$delta
)
```

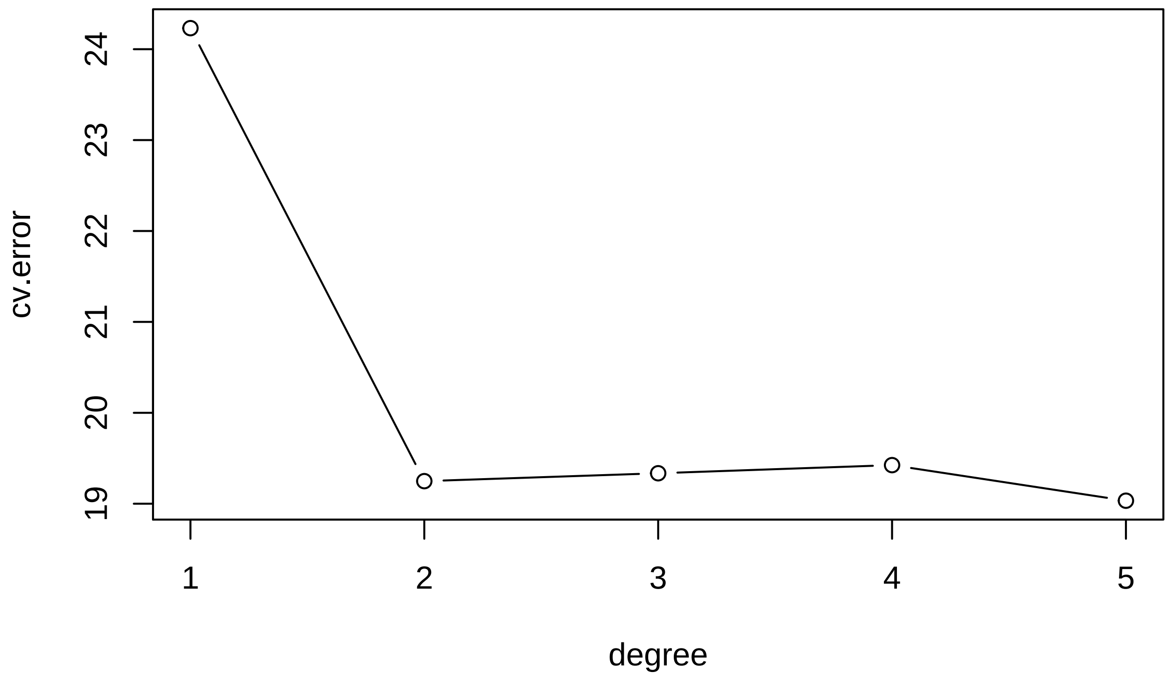
```
##    user  system elapsed
##     1.3     0.0     1.3
```

```
system.time(
loocv(glm_auto)
)
```

```
##    user  system elapsed
##       0       0       0
```

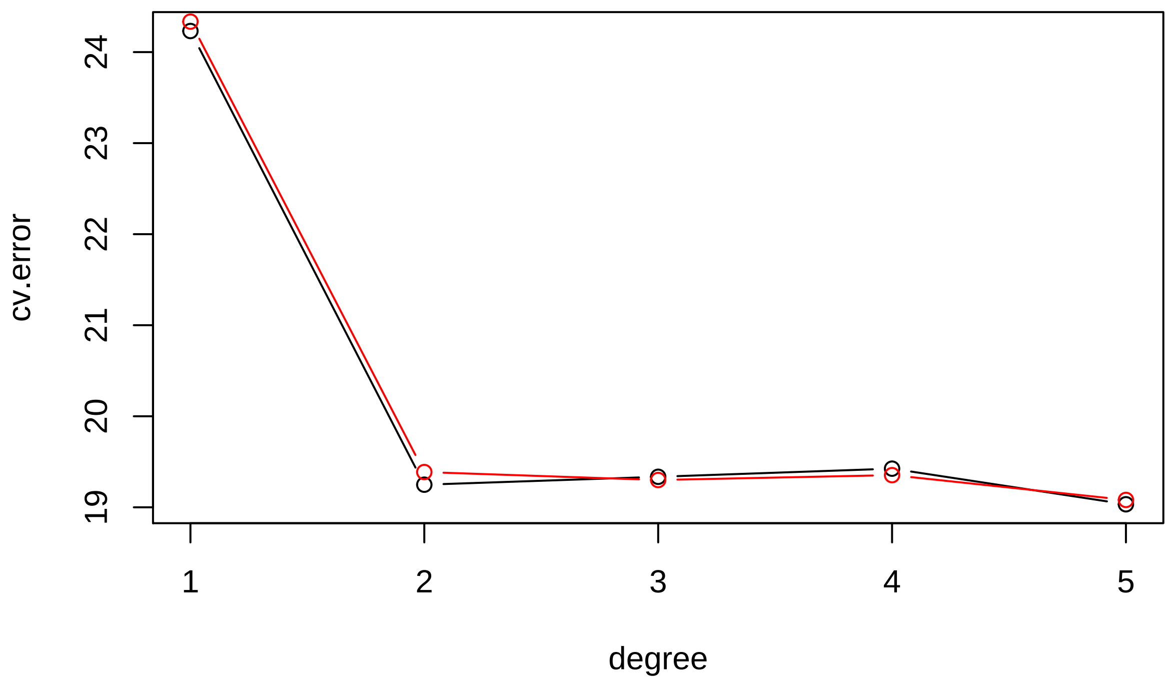
Next, lets use a `for` loop to efficiently create 5 new polynomial versions of the previous model, regressing `horsepower` against `mpg` and see if the results improve as polynomial order increases.

```
cv.error <- rep(0, 5)
degree <- 1:5

for(d in degree){
  glm.fit <- glm(mpg ~ poly(horsepower, d), data = Auto)
  cv.error[d] <- loocv(glm.fit)
}

plot(degree, cv.error, type = "b")
```



The10-fold Cross-Validation

```
cv.error10 <- rep(0, 5)

for(d in degree){
  glm.fit <- glm(mpg ~ poly(horsepower, d), data = Auto)
  cv.error10[d] <- cv.glm(Auto, glm.fit, K=10)$delta[1]
}
plot(degree, cv.error, type = "b")
lines(degree,cv.error10, type = "b", col = "red")
```



Bootstrap

Minimum risk investment fucntion from Section 5.2:

```
alpha <- function(x, y) {

          var_x <- var(x)
          var_y <- var(y)
          cov_xy <- cov(x, y)
          (var_y - cov_xy)/(var_x + var_y - 2 *cov_xy)
}

alpha(Portfolio$X, Portfolio$Y)
```

```
## [1] 0.5758321
```

So what is the standard error of alpha?

```r
alpha.fn <- function(data, index){
                with(data[index, ], alpha(X, Y))
}

alpha.fn(Portfolio, 1:100)
```

```
## [1] 0.5758321
```

```r
set.seed(1)
alpha.fn(Portfolio,sample(1:100, 100, replace = TRUE))
```

```
## [1] 0.5963833
```

```r
boot.out <- boot(Portfolio, alpha.fn, R = 1000)
boot.out$t0
```

```
## [1] 0.5758321
```

```r
plot(boot.out)
```