RESAMPLING METHODS

1. Validation Set Approach

```
> attach(Auto); n = length(mpg);
> Z = sample( n, n/2 )                              # Random subsample of size n/2
                                                    # Works similarly to generating a Bernoulli variable
> reg.fit = lm( mpg ~ weight + horsepower + acceleration, subset=Z )          # Fit using training data
> mpg_predicted = predict( reg.fit, Auto )
                                                    # Use this model to predict the testing data [-Z]
> plot(mpg[-Z],mpg_predicted[-Z])                   # We can see a nonlinear component
> abline(0,1)                                       # Compare with the line y=x.

> mean( (mpg - mpg_predicted) [-Z]^2 )              # Estimate the mean-squared error MSE
[1] 49.00679
```

2. Jackknife (Leave-One-Out Cross-Validation, LOOCV)

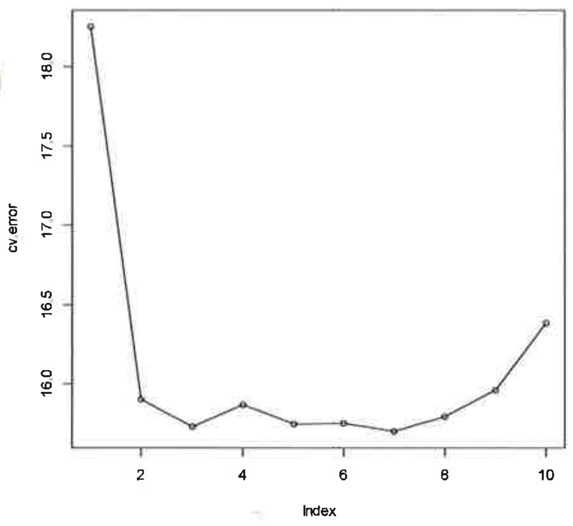
```
> library(boot)
> glm.fit = glm(mpg ~ weight + horsepower + acceleration) # Default "family" is Normal, so this GLM model
> cv.error = cv.glm( Auto, glm.fit )                      # is the same as LM, standard linear regression.
> names(cv.error)                                         # This cross-validation tool as several outputs
[1] "call"  "K"    "delta" "seed"                         # We are interested in "delta"

> error$delta
[1] 18.25595 18.25542
```

# Delta consists of 2 numbers – estimated prediction error and its version adjusted for the lost sample size due to cross-validation.

# Example – what power of "horsepower" is optimal for this prediction?

```
> cv.error = rep(0,10)                                          # Initiate a vector of estimated errors
> for (p in 1:10){                                                      # Fit polynomial regression models
+ glm.fit = glm( mpg ~ weight + poly(horsepower,p) + acceleration )     # with power p=1..10
+ cv.error[p] = cv.glm( Auto, glm.fit )$delta[1]  }                     # Save prediction errors
> cv.error                                                              # Look at the results
 [1] 18.25595 15.90163 15.72995 15.86879 15.74517 15.74989 15.70073 15.79314 15.95933 16.38301
# Although p=7 yields the lowest estimated prediction error, after p=2, the improvement is very little.
> plot(cv.error)
> lines(cv.error)
```



3. K-Fold Cross-Validation

# We can specify K within cv.glm (K=1 is LOOCV, but it is not allowed. Omitted K is K=1 by default).

```
> cv.error = rep(0,10)
> for (p in 1:10){ glm.fit = glm( mpg ~ weight + poly(horsepower,p) + acceleration )
+ cv.error[p] = cv.glm( Auto, glm.fit, K=30 )$delta[1]    }
> cv.error
 [1] 18.22699 15.87030 15.73565 15.85911 15.80686 15.71585 16.09625 15.67797 15.82097 16.48196
```

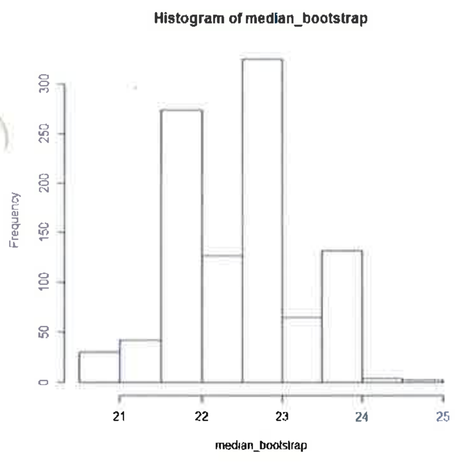
4. BOOTSTRAP

(4a) A simple example: use bootstrap to estimate the standard deviation of a sample median.

```
> B = 1000                                # Number of bootstrap samples
> n = length(mpg)                         # Sample size (and the size of each bootstrap sample)
> median_bootstrap = rep(0,B)             # Initiate a vector of bootstrap medians

> for (k in 1:B){                         # Do-loop to produce B bootstrap samples.
+ b = sample( n, n, replace=TRUE )        # Take a random subsample of size n with replacement
+ median_bootstrap[k] = median( mpg[b] )  # Compute the sample median of each bootstrap subsample
+ }

> hist(median_bootstrap)
```



```
> median(mpg)
[1] 22.75
> sd(median_bootstrap)
[1] 0.7661376
```

# The actual sample median is 22.75; bootstrap medians range
# from 20.5 to 25.0.

# The bootstrap estimator of SD(median) is 0.7661376.

(4b) R package "boot"

# There is a special bootstrap tool in R package boot.
# To use it for the same problem, we define a function that computes a sample median...

```
> median.fn = function(X,subsample){
+ return(median(X[subsample]))
+ }
```

# For example, here is a sample median for a small subsample of size 10 without replacement...

```
> median.fn( mpg, sample(n,10) )
[1] 24.95
```

# If the subsample is the whole sample, with all its indices, we get our original sample median...

```
> median.fn( mpg,  )
[1] 22.75
```

# Then we apply this function to many bootstrap samples (R is their number) and summarize the obtained statistics...

```
> library (boot)
> boot( mpg, median.fn, R=10000 )

ORDINARY NONPARAMETRIC BOOTSTRAP

Call:
boot(data = mpg, statistic = median.fn, R = 10000)

Bootstrap Statistics :
    original  bias    std. error
t1*    22.75 -0.1406   0.7687722
```

# The bootstrap estimate of the standard deviation of a sample median is `0.7687722`, and the bootstrap estimate of the bias is `-0.1406`. If we run the same command again, we might get somewhat different estimates, because the bootstrap method is based on random resampling.

```
> boot( mpg, median.fn, R=10000 )
    original   bias    std. error
t1*    22.75 -0.13368   0.7727318

> boot( mpg, median.fn, R=10000 )
    original   bias    std. error
t1*    22.75 -0.13437   0.7617304
```

(4c) Another problem – find a bootstrap estimate of the standard deviation of each regression coefficient

# Define a function returning regression intercept and slope…

```
> slopes.fn = function( dataset, subsample ){
+ reg = lm( mpg ~ horsepower, data=Auto[subsample, ] )
+ beta = coef(reg)
+ return(beta)
+ }

> boot( Auto, slopes.fn, R=10000 )

Bootstrap Statistics :
      original        bias    std. error
t1* 39.9358610  0.0329999048 0.857890744
t2* -0.1578447 -0.0003929433 0.007444107
```

# We got bootstrap estimates, Std(intercept) = 0.85789, Std(slope) = 0.007444.
# Actually, this can be obtained easily, and without bootstrap…

```
> summary( lm( mpg ~ horsepower, data=Auto ) )

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 ***
```

# But…. Why are the bootstrap estimates of standard errors higher than the ones obtained by regression formulae? Random variation due to bootstrapping? Let's try bootstrap a few more times.

```
> boot( Auto, slopes.fn, R=10000 )
      original        bias    std. error
t1* 39.9358610  0.0366632516 0.864274619
t2* -0.1578447 -0.0003676081 0.007459809

> boot( Auto, slopes.fn, R=10000 )
      original       bias    std. error
t1* 39.9358610  0.024793987 0.866532993
t2* -0.1578447 -0.000309597 0.007501155

> boot( Auto, slopes.fn, R=10000 )
Bootstrap Statistics :
      original        bias    std. error
t1* 39.9358610  0.0411404502 0.868404735
t2* -0.1578447 -0.0004208256 0.007500111
```

A comment about discrepancy with the standard estimates. Bootstrap estimates are pretty close to each other. They should be, because each of them is based on 10,000 bootstrap samples. However, the bootstrap standard errors of slopes are still a little higher than the estimates obtained by the standard regression analysis. The book explains this phenomenon by a number of assumptions that the standard regression requires. In particular, nonrandom predictors X and a constant variance of responses $\mathrm{Var}(Y) = \sigma^2$. Bootstrap is a nonparametric procedure, it is not based on any particular model. Thus, it can account for the variance of predictors. Here, "horsepower" is the predictor, and it is probably random.

Since some of the standard regression assumptions are questionable here, the bootstrap method for estimating standard errors is more reliable.