Chapter 5 Resampling Methods

Exercise 1

$$Var(\alpha X + (1-\alpha)Y) = Var(\alpha X) + Var((1-\alpha)Y) + 2Cov(\alpha X, (1-\alpha)Y)$$

$$= \alpha^2\sigma_X^2 + (1-\alpha)^2\sigma_Y^2 + 2\alpha\sigma_{XY} - 2\alpha^2\sigma_{XY}$$

$$= (\sigma_X^2 + \sigma_Y^2 - 2\sigma_{XY})\alpha^2 - (2\sigma_Y^2 - 2\sigma_{XY})\alpha + \sigma_Y^2$$

Take derivative with respect to

$$\alpha$$

and set it to zero, we get

$$2(\sigma_X^2 + \sigma_Y^2 - 2\sigma_{XY})\alpha - (2\sigma_Y^2 - 2\sigma_{XY}) = 0$$

$$\Leftrightarrow \alpha = \frac{2\sigma_Y^2 - 2\sigma_{XY}}{2(\sigma_X^2 + \sigma_Y^2 - 2\sigma_{XY})} = \frac{\sigma_Y^2 - \sigma_{XY}}{\sigma_X^2 + \sigma_Y^2 - 2\sigma_{XY}}$$

Exercise 2

a

The probability that the first bootstrap observation is the jth observation from the original sample is

$$1/n$$

Therefore the probability that the first bootstrap observation is not the jth observation from the original sample is

$$1 - 1/n$$

b

$$1 - 1/n$$

c

The probability that the jth observation is not in the bootstrap sample is the product of the probabilities that the first, the second, and so on to the last bootstrap observation is not the jth observation from the original sample. Mathematically writing, it is

$$(1-1/n)(1-1/n)...(1-1/n) = (1-1/n)^n$$

d

When n = 5, the probability that the jth observation is in the bootstrap sample is

$$1 - (1 - 1/5)^5 = 0.6723$$

e

0.6340

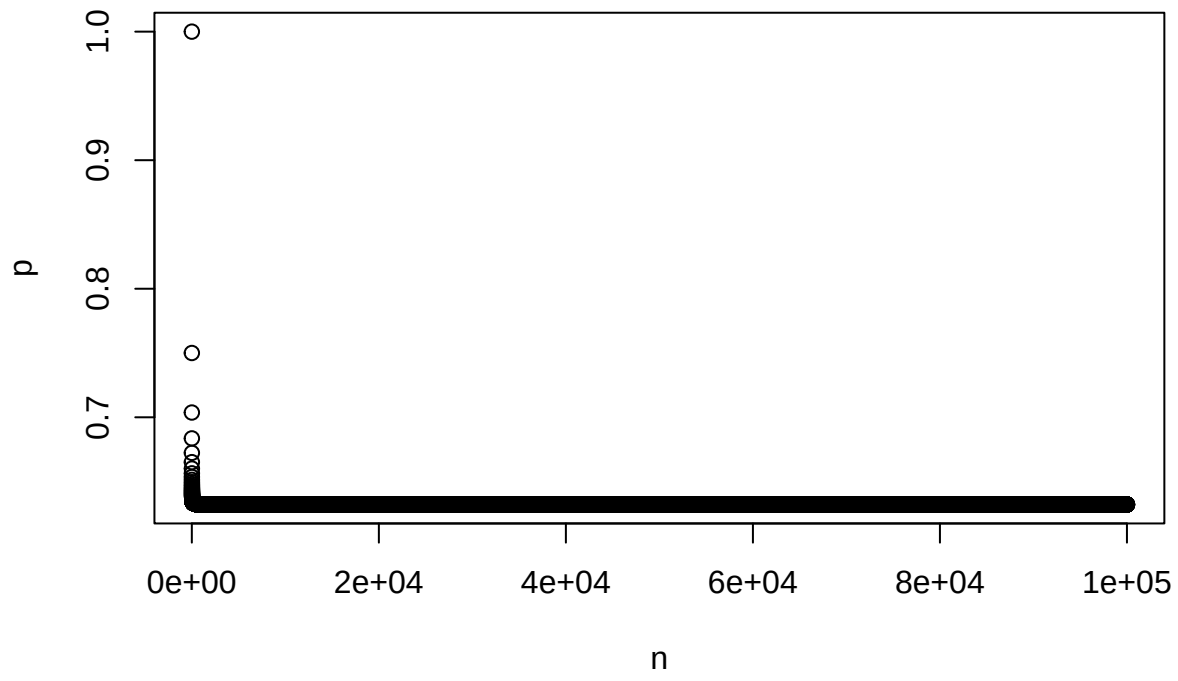
f

0.6321

g

The probability that the jth observation is in the bootstrap sample approaches a specific number when n is approaches infinity. The specific number here is approximatly 0.6321.

```
n = 1:1e+5
p = 1 - (1 - 1/n)^n
plot(n, p)
```



h

The mean of the probability that the fourth observation is in the bootstrap sample is very close to the specific number 0.6321 mentioned in (f) and (g)

```
store = rep(NA , 10000)
for (i in 1:10000) {
  store[i] = sum(sample(1:100 , rep = TRUE) == 4) > 0
}
mean(store)
```

```
## [1] 0.6337
```

Exercise 3

a

The data set is divided into k parts. In the first iteration, the first part is used as a validation set and the remaining (k - 1) parts are used as a training set. In the second iteration, the second part is used as a validation set and the remaining (k - 1) parts are used as a training set. This process is iteratively implemented for k times.

b

i

K-fold validation can be understood as implementing the validation set approach multiple times. If we divide the data into k parts, each part will be treated as a validation set k times which results in k test errors. This tackles the high variance disadvantage of the validation set approach and also avoids overestimating the test MSE.

On the other hand, k-fold validation set is more computationally expensive and more time-consuming.

ii

We can consider LOOCV as k-fold cross validation when k = n. Therefore, as n is large, LOOCV takes more time than k-fold cross validation. In other words, k-fold CV has a computational advantage to LOOCV when k $<$ n.

There is a bias-variance trade-off associated with the choice of k in k-fold cross-validation. Typically, given these considerations, one performs k-fold cross-validation using k = 5 or k = 10, as these values have been shown empirically to yield test error rate estimates that suffer neither from excessively high bias nor from very high variance.

Exercise 4

We would want to use bootstrap method to estimate the standard deviation. We randomly select n observations from the data set in order to produce a bootstrap data set, the sampling is performed with replacement.

This procedure is repeated B times for some large value of B, in order to produce B different bootstrap data sets B. We can compute the standard error of these bootstrap estimates using the formula (5.8).

Exercise 5

```
library(ISLR2)
head(Default)
```

```
##   default student   balance    income
## 1      No      No  729.5265 44361.625
## 2      No     Yes  817.1804 12106.135
## 3      No      No 1073.5492 31767.139
## 4      No      No  529.2506 35704.494
## 5      No      No  785.6559 38463.496
## 6      No     Yes  919.5885  7491.559
```

a

```
set.seed(1)
lgr_model = glm(default ~ income + balance, data = Default, family = binomial)
summary(lgr_model)
```

```
##
## Call:
## glm(formula = default ~ income + balance, family = binomial,
##     data = Default)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.4725  -0.1444  -0.0574  -0.0211   3.7245
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.154e+01  4.348e-01 -26.545  < 2e-16 ***
## income       2.081e-05  4.985e-06   4.174 2.99e-05 ***
## balance      5.647e-03  2.274e-04  24.836  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2920.6  on 9999  degrees of freedom
## Residual deviance: 1579.0  on 9997  degrees of freedom
## AIC: 1585
##
## Number of Fisher Scoring iterations: 8
```

b

i

```r
set.seed(1)
n = dim(Default)[1]
train = sample(n, n/2, replace = F)
train_set = Default[train, ]
val_set = Default[-train, ]
```

ii

```r
set.seed(1)
lgr_model = glm(default ~ income + balance, data = train_set, family = binomial)
summary(lgr_model)
```

```
##
## Call:
## glm(formula = default ~ income + balance, family = binomial,
##     data = train_set)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.5830  -0.1428  -0.0573  -0.0213   3.3395
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.194e+01  6.178e-01 -19.333  < 2e-16 ***
## income       3.262e-05  7.024e-06   4.644 3.41e-06 ***
## balance      5.689e-03  3.158e-04  18.014  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1523.8  on 4999  degrees of freedom
## Residual deviance:  803.3  on 4997  degrees of freedom
## AIC: 809.3
##
## Number of Fisher Scoring iterations: 8
```

iii

```r
val_probs = predict(lgr_model, newdata = val_set, type = 'response')
val_preds = rep('No', n/2)
val_preds[val_probs > 0.5] = 'Yes'
```

iv

```r
1 - mean(val_preds == val_set$default)
```

```
## [1] 0.0254
```

c

The results are different but the difference is not great.

```
train = sample(n, n/2, replace = F)
train_set = Default[train, ]
val_set = Default[-train, ]

lgr_model = glm(default ~ income + balance, data = train_set, family = binomial)

val_probs = predict(lgr_model, newdata = val_set, type = 'response')
val_preds = rep('No', n/2)
val_preds[val_probs > 0.5] = 'Yes'

1 - mean(val_preds == val_set$default)
```

```
## [1] 0.0254
```

```
train = sample(n, n/2, replace = F)
train_set = Default[train, ]
val_set = Default[-train, ]

lgr_model = glm(default ~ income + balance, data = train_set, family = binomial)

val_probs = predict(lgr_model, newdata = val_set, type = 'response')
val_preds = rep('No', n/2)
val_preds[val_probs > 0.5] = 'Yes'

1 - mean(val_preds == val_set$default)
```

```
## [1] 0.0274
```

```
train = sample(n, n/2, replace = F)
train_set = Default[train, ]
val_set = Default[-train, ]

lgr_model = glm(default ~ income + balance, data = train_set, family = binomial)

val_probs = predict(lgr_model, newdata = val_set, type = 'response')
val_preds = rep('No', n/2)
val_preds[val_probs > 0.5] = 'Yes'

1 - mean(val_preds == val_set$default)
```

```
## [1] 0.0244
```

d

It seems to be that including the dummy variable "student" does not lead to a reduction in the test error.

```
train = sample(n, n/2, replace = F)
train_set = Default[train, ]
val_set = Default[-train, ]

lgr_model = glm(default ~ income + balance + student,
                data = train_set, family = binomial)

val_probs = predict(lgr_model, newdata = val_set, type = 'response')
val_preds = rep('No', n/2)
val_preds[val_probs > 0.5] = 'Yes'

1 - mean(val_preds == val_set$default)
```

```
## [1] 0.0242
```

Exercise 6

a

```
set.seed(1)
lgr_model = glm(default ~ income + balance, data = Default, family = binomial)
summary(lgr_model)
```

```
##
## Call:
## glm(formula = default ~ income + balance, family = binomial,
##     data = Default)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.4725  -0.1444  -0.0574  -0.0211   3.7245
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.154e+01  4.348e-01 -26.545  < 2e-16 ***
## income       2.081e-05  4.985e-06   4.174 2.99e-05 ***
## balance      5.647e-03  2.274e-04  24.836  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2920.6  on 9999  degrees of freedom
## Residual deviance: 1579.0  on 9997  degrees of freedom
## AIC: 1585
##
## Number of Fisher Scoring iterations: 8
```

b

```
boot.fn = function(data, index){
  model = glm(default ~ income + balance, data = Default,
              subset = index, family = binomial)
  coef(model)
}

boot.fn(Default, 1:n)
```

```
##   (Intercept)        income       balance
## -1.154047e+01  2.080898e-05  5.647103e-03
```

c

```
indices = sample(n, n, replace = T)
boot.fn(Default, indices)
```

```
##   (Intercept)        income       balance
## -1.209470e+01  2.815290e-05  5.837117e-03
```

```
library(boot)
boot(Default, boot.fn, R = 1000)
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = Default, statistic = boot.fn, R = 1000)
##
##
## Bootstrap Statistics :
##          original        bias     std. error
## t1* -1.154047e+01 -3.883682e-02 4.345921e-01
## t2*  2.080898e-05  1.598152e-07 4.860169e-06
## t3*  5.647103e-03  1.838192e-05 2.298390e-04
```

d

The estimated standard errors obtained using the glm() function are very close to the estimated standard errors obtained using bootstrap function

Exercise 7

```
head(Weekly)
```

```
##   Year   Lag1   Lag2   Lag3   Lag4   Lag5    Volume  Today Direction
## 1 1990  0.816  1.572 -3.936 -0.229 -3.484 0.1549760 -0.270      Down
## 2 1990 -0.270  0.816  1.572 -3.936 -0.229 0.1485740 -2.576      Down
## 3 1990 -2.576 -0.270  0.816  1.572 -3.936 0.1598375  3.514        Up
## 4 1990  3.514 -2.576 -0.270  0.816  1.572 0.1616300  0.712        Up
## 5 1990  0.712  3.514 -2.576 -0.270  0.816 0.1537280  1.178        Up
## 6 1990  1.178  0.712  3.514 -2.576 -0.270 0.1544440 -1.372      Down
```

```
n = dim(Weekly)[1]
```

a

```
lgr_model = glm(Direction ~ Lag1 + Lag2, data = Weekly, family = binomial)
```

b

```
lgr_model = glm(Direction ~ Lag1 + Lag2, data = Weekly,
                subset = 2:n, family = binomial)
```

c

The estimated probability is greater than 0.5 which implies a "Down" as prediction for the first observation. This is a correct classification.

```
predict(lgr_model, newdata = Weekly[1, ], type = 'response')
```

```
##         1
## 0.5713923
```

d

```
probs = rep(NA, n)
preds = rep('Down', n)

for (i in 1:n){
  model = glm(Direction ~ Lag1 + Lag2, data = Weekly[-i, ], family = binomial)
  probs[i] = predict(model, newdata = Weekly[i, ], type = 'response')
}
```

```
preds[probs > 0.5] = 'Up'
```

e

The LOOCV estimate for the test error is about 45%

```
1 - mean(preds == Weekly$Direction)
```

```
## [1] 0.4499541
```

Exercise 8

a

n = 100 and p = 1. The model equation is

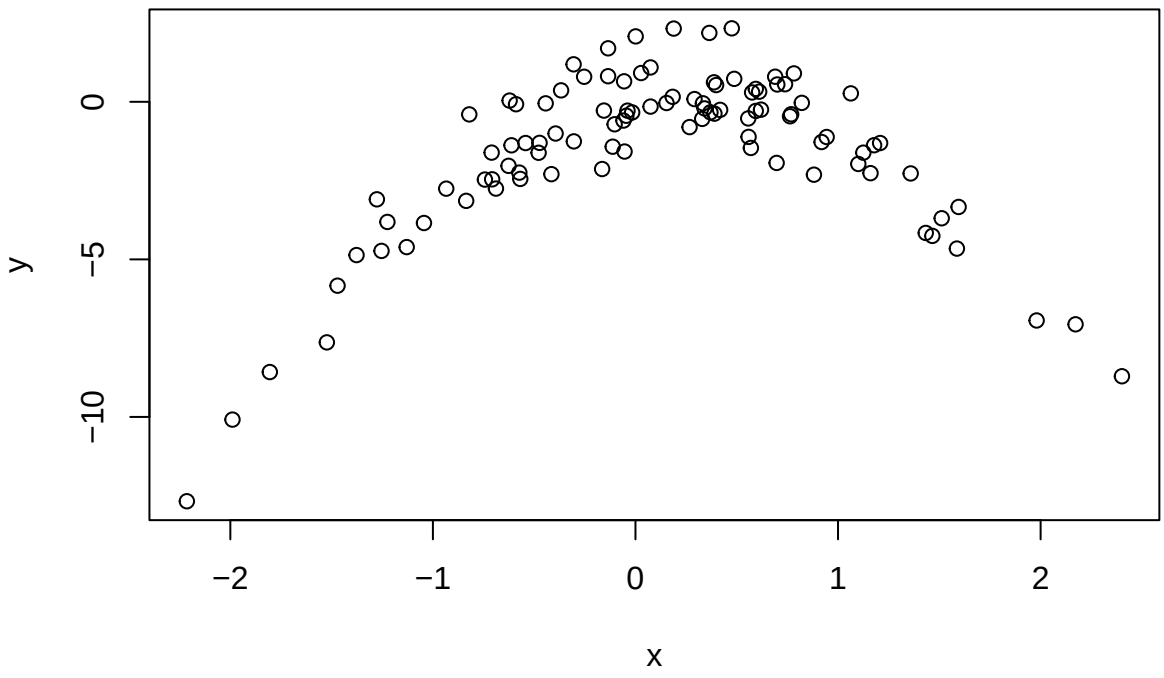
$$y = x - 2x^2 + \epsilon$$

```
set.seed(1)
x = rnorm(100)
y = x - 2 * x^2 + rnorm(100)
```

b

The relationship between x and y is obviously non-linear. The error term is not large.

```
plot(x, y)
```



c

```
set.seed(1)
simulated_data = data.frame(x, y)
loocv = rep(NA, 4)

for (i in 1:4){
  model = glm(y ~ poly(x, i), data = simulated_data)
  loocv[i] = cv.glm(simulated_data, model)$delta[1]
}
loocv
```

```
## [1] 7.2881616 0.9374236 0.9566218 0.9539049
```

d

The results from (c) and (d) are identical.

```
set.seed(10)

for (i in 1:4){
  model = glm(y ~ poly(x, i), data = simulated_data)
```

```
  loocv[i] = cv.glm(simulated_data, model)$delta[1]
}

loocv
```

```
## [1] 7.2881616 0.9374236 0.9566218 0.9539049
```

e

The second model obtained the smallest error simply since it almost captured the true relationship between x and y.

f

The summary from the second model says that predictors x and x^2 are statistically significant. Hence, the below results agree with the conclusions drawn based on the cross-validation results.

```
summary(glm(y ~ poly(x, 1), data = simulated_data))
```

```
##
## Call:
## glm(formula = y ~ poly(x, 1), data = simulated_data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -9.5161  -0.6800   0.6812   1.5491   3.8183
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   -1.550      0.260  -5.961 3.95e-08 ***
## poly(x, 1)     6.189      2.600   2.380   0.0192 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 6.760719)
##
##     Null deviance: 700.85  on 99  degrees of freedom
## Residual deviance: 662.55  on 98  degrees of freedom
## AIC: 478.88
##
## Number of Fisher Scoring iterations: 2
```

```
summary(glm(y ~ poly(x, 2), data = simulated_data))
```

```
##
## Call:
## glm(formula = y ~ poly(x, 2), data = simulated_data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
```

```
## -1.9650  -0.6254  -0.1288   0.5803   2.2700
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -1.5500     0.0958  -16.18  < 2e-16 ***
## poly(x, 2)1   6.1888     0.9580    6.46 4.18e-09 ***
## poly(x, 2)2 -23.9483     0.9580  -25.00  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 0.9178258)
##
##     Null deviance: 700.852  on 99  degrees of freedom
## Residual deviance:  89.029  on 97  degrees of freedom
## AIC: 280.17
##
## Number of Fisher Scoring iterations: 2
```

```
summary(glm(y ~ poly(x, 3), data = simulated_data))
```

```
##
## Call:
## glm(formula = y ~ poly(x, 3), data = simulated_data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.9765  -0.6302  -0.1227   0.5545   2.2843
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -1.55002    0.09626 -16.102  < 2e-16 ***
## poly(x, 3)1   6.18883    0.96263   6.429 4.97e-09 ***
## poly(x, 3)2 -23.94830    0.96263 -24.878  < 2e-16 ***
## poly(x, 3)3   0.26411    0.96263   0.274    0.784
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 0.9266599)
##
##     Null deviance: 700.852  on 99  degrees of freedom
## Residual deviance:  88.959  on 96  degrees of freedom
## AIC: 282.09
##
## Number of Fisher Scoring iterations: 2
```

```
summary(glm(y ~ poly(x, 4), data = simulated_data))
```

```
##
## Call:
## glm(formula = y ~ poly(x, 4), data = simulated_data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
```

```
## -2.0550  -0.6212  -0.1567   0.5952   2.2267
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -1.55002    0.09591 -16.162  < 2e-16 ***
## poly(x, 4)1   6.18883    0.95905   6.453 4.59e-09 ***
## poly(x, 4)2 -23.94830    0.95905 -24.971  < 2e-16 ***
## poly(x, 4)3   0.26411    0.95905   0.275    0.784
## poly(x, 4)4   1.25710    0.95905   1.311    0.193
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 0.9197797)
##
##     Null deviance: 700.852  on 99  degrees of freedom
## Residual deviance:  87.379  on 95  degrees of freedom
## AIC: 282.3
##
## Number of Fisher Scoring iterations: 2
```

Exercise 9

```
head(Boston)
```

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

a

```
mean(Boston$medv)
```

```
## [1] 22.53281
```

b

```
std = function(data){
  n = length(data)
  mu_hat = mean(data)
  var = sum((mu_hat - data)^2)
  sqrt(var) / (n - 1)
}

std(Boston$medv)
```

```
## [1] 0.4092658
```

```
sd(Boston$medv) / sqrt(n - 1)
```

```
## [1] 0.2788282
```

c

The standard errors of

$$\hat{\mu}$$

from (b) and (c) are nearly the same.

```
boot_fn = function(data, indices){
  indices = sample(n, n, replace =T)
  mean(data[indices])
}

boot_fn(Boston$medv, n)
```

```
## [1] NA
```

```
set.seed(1)
boot_mu = boot(Boston$medv, boot_fn, 1000)
boot_mu
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = Boston$medv, statistic = boot_fn, R = 1000)
##
##
## Bootstrap Statistics :
## WARNING: All values of t1* are NA
```

d

The 95% confidence intervals for the mean using bootstrap and t-test are very close to each other.

```
22.61917 - 2 * 0.407274
```

```
## [1] 21.80462
```

```
22.61917 + 2 * 0.407274
```

```
## [1] 23.43372
```

```
t.test(Boston$medv)
```

```
##
##  One Sample t-test
##
## data:  Boston$medv
## t = 55.111, df = 505, p-value < 2.2e-16
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
##  21.72953 23.33608
## sample estimates:
## mean of x
##  22.53281
```

e

```
median(Boston$medv)
```

```
## [1] 21.2
```

f

The estimated median from bootstrap is really close to the median from (e). Also, the standard error for the estimated median is small (compared to the estimated median value).

```
boot_fn = function(data, indices){
  median(data[indices])
}

boot(Boston$medv, boot_fn, R = 1000)
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = Boston$medv, statistic = boot_fn, R = 1000)
##
##
## Bootstrap Statistics :
##     original   bias    std. error
## t1*     21.2 -0.03315   0.3640826
```

g

```
quantile(Boston$medv)
```

```
##     0%    25%    50%    75%   100%
##  5.000 17.025 21.200 25.000 50.000
```

```
quantile(Boston$medv, 0.1)
```

```
##   10%
## 12.75
```

h

As you might expected, there is no significant difference between results from (g) and (h). And, the standard error for the estimated tenth percentile is small (compared to the estimated tenth percentile value it self).

```
boot_fn = function(data, indices){
  quantile(data[indices], 0.1)
}

boot(Boston$medv, boot_fn, R = 1000)
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = Boston$medv, statistic = boot_fn, R = 1000)
##
##
## Bootstrap Statistics :
##     original  bias    std. error
## t1*    12.75  0.0126   0.4893984
```