chap5

#Conceptual 1-4 + #labs + #Applied 5-9

#Conceptual #1.

#see the process in the .qmd file, the latex can't render

#2.

#a. #p = 1 -1/n #Since each draw is independent and there are n equally likely observations to choose from

#b #same with a, since replacement = True

#c #(1 -1/n)^n # the draws are independent, the probability of the jth observation not being chosen in each draw multiplies across the n draws.

#d. #0.67232

```
#e.
1- (1 -1/100)^100
```

```
[1] 0.6339677
```

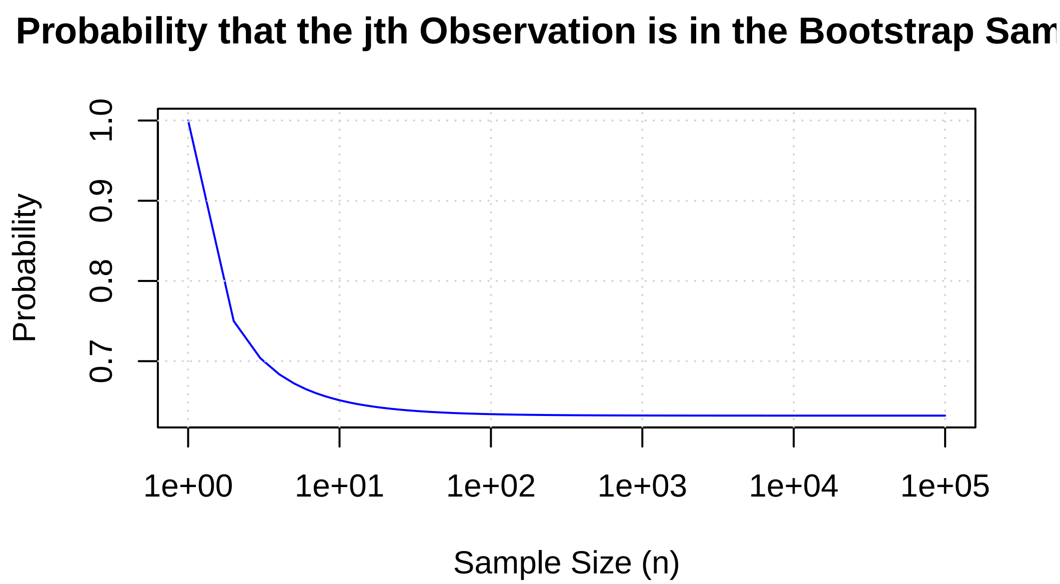
```
#f
1- (1 -1/10000)^10000
```

```
[1] 0.632139
```

```
#g
n_values <- 1:100000
prob_in_sample <- 1 - (1 - 1/n_values)^n_values

# Plotting
```

```r
plot(n_values, prob_in_sample, type = 'l', log = "x",
     xlab = "Sample Size (n)", ylab = "Probability",
     main = "Probability that the jth Observation is in the Bootstrap Sample",
     col = "blue")
grid()
```



#bottom-left corner seems in the middle between (1e+00,1e+01) #h

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

```
[1] 0.6335
```

#the result is correctly close to 1- (1 -1/100)^100 = 0.634 #3. #a.perform k times from the first validation set to the last #validation set : n/k #training set : n(k-1)/k

#b. #i. k-fold cross-validation v.s The validation set approach? #ad:validation set's error rate may tend to overestimate the test error rate than k-fold cross-validation #disad:The validation set approach is Simpler and faster as it only requires one split

#ii. k-fold cross-validation v.s LOOCV? #ad:k-fold CV with k<n has a computational advantage to LOOCV # k-fold CV often gives more accurate estimates of the test error rate than does LOOCV #disad:if k is small (e.g., 5 or 10), K-fold CV may suffer from a slight bias because a larger proportion of the data is excluded in each fold compared to LOOCV

#4. #fit statistical learning ->using resampling methods to refit and make predictions for X ->Compute Prediction Variability & Estimate std

#labs

```r
library(ggplot2)
```

```r
library(ISLR2)
set.seed(1)
train <- sample(392, 196)
```

```r
lm.fit <- lm(mpg ~ horsepower, data = Auto, subset = train)
```

#the -train index belowselects only the observations that are not in the training set

```r
attach(Auto)
```

```
The following object is masked from package:ggplot2:

    mpg
```

```r
mean((mpg - predict(lm.fit, Auto))[-train]^2)
```

```
[1] 23.26601
```

```r
lm.fit2 <- lm(mpg ~ poly(horsepower, 2), data = Auto,subset = train)

mean((mpg - predict(lm.fit2, Auto))[-train]^2)
```

```
[1] 18.71646
```

```r
lm.fit3 <- lm(mpg ~ poly(horsepower, 3), data = Auto,subset = train)
mean((mpg - predict(lm.fit3, Auto))[-train]^2)
```

```
[1] 18.79401
```

```r
set.seed(2)
train <- sample(392, 196)
lm.fit <- lm(mpg ~ horsepower, subset = train)
mean((mpg - predict(lm.fit, Auto))[-train]^2)
```

```
[1] 25.72651
```

```r
lm.fit2 <- lm(mpg ~ poly(horsepower, 2), data = Auto, subset = train)
mean((mpg - predict(lm.fit2, Auto))[-train]^2)
```

```
[1] 20.43036
```

```r
lm.fit3 <- lm(mpg ~ poly(horsepower, 3), data = Auto,subset = train)
mean((mpg - predict(lm.fit3, Auto))[-train]^2)
```

```
[1] 20.38533
```

#LOOCV

```r
glm.fit <- glm(mpg ~ horsepower, data = Auto)
coef(glm.fit)
```

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

```r
lm.fit <- lm(mpg ~ horsepower, data = Auto)
coef(lm.fit)
```

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

```r
library(boot)
glm.fit <- glm(mpg ~ horsepower, data = Auto)
```

```r
cv.err <- cv.glm(Auto, glm.fit)
cv.err$delta
```

```
[1] 24.23151 24.23114
```

```r
cv.error <- rep(0, 10)
for (i in 1:10) {
  glm.fit <- glm(mpg ~ poly(horsepower, i), data = Auto)
  cv.error[i] <- cv.glm(Auto, glm.fit)$delta[1]
  }
cv.error
```

```
 [1] 24.23151 19.24821 19.33498 19.42443 19.03321 18.97864 18.83305 18.96115
 [9] 19.06863 19.49093
```

#k-Fold Cross-Validation

```r
set.seed(17)
cv.error.10 <- rep(0, 10)
for (i in 1:10) {
  glm.fit <- glm(mpg ~ poly(horsepower, i), data = Auto)
  cv.error.10[i] <- cv.glm(Auto, glm.fit, K = 10)$delta[1]
  }
cv.error.10
```

```
 [1] 24.27207 19.26909 19.34805 19.29496 19.03198 18.89781 19.12061 19.14666
 [9] 18.87013 20.95520
```

Bootstrap

```r
alpha.fn <- function(data, index) {
  X <- data$X[index]
  Y <- data$Y[index]
  (var(Y) - cov(X, Y)) / (var(X) + var(Y) - 2 * cov(X, Y))
  }
```

#an estimate for   based on applying (5.7) to the observations indexed by the argument index.

```r
alpha.fn(Portfolio, 1:100) # estimate  using all 100 observations.
```

```
[1] 0.5758321
```

randomly select 100 observations from the range 1 to 100

```r
set.seed(7)
alpha.fn(Portfolio, sample(100, 100, replace = T))
```

```
[1] 0.5385326
```

#R = 1, 000 bootstrap estimates for

```r
boot(Portfolio , alpha.fn, R = 1000)
```

```
ORDINARY NONPARAMETRIC BOOTSTRAP


Call:
boot(data = Portfolio, statistic = alpha.fn, R = 1000)


Bootstrap Statistics :
     original        bias    std. error
t1* 0.5758321 0.0007959475  0.08969074
```

```r
boot.fn <-
  function(data, index) +
  coef(lm(mpg ~ horsepower, data = data, subset = index))
boot.fn(Auto, 1:392)
```

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

```r
set.seed(1)
boot.fn(Auto, sample(392, 392, replace = T))
```

```
(Intercept)  horsepower 
 40.3404517  -0.1634868 
```

```r
boot.fn(Auto, sample(392, 392, replace = T))
```

```
(Intercept)  horsepower 
 40.1186906  -0.1577063 
```

compute the standard errors of 1,000 bootstrap estimates for the intercept and slope terms

```r
boot(Auto, boot.fn, 1000)
```

```

ORDINARY NONPARAMETRIC BOOTSTRAP


Call:
boot(data = Auto, statistic = boot.fn, R = 1000)


Bootstrap Statistics :
      original        bias    std. error
t1* 39.9358610  0.0544513229 0.841289790
t2* -0.1578447 -0.0006170901 0.007343073
```

```r
summary(lm(mpg ~ horsepower, data = Auto))$coef
```

```
              Estimate  Std. Error   t value      Pr(>|t|)
(Intercept) 39.9358610 0.717498656  55.65984 1.220362e-187
horsepower  -0.1578447 0.006445501 -24.48914  7.031989e-81
```

```r
boot.fn <-
  function(data, index) +
  coef(lm(mpg ~ horsepower + I(horsepower^2),
          data = data, subset = index) )
set.seed(1)
boot(Auto, boot.fn, 1000)
```

```
ORDINARY NONPARAMETRIC BOOTSTRAP


Call:
boot(data = Auto, statistic = boot.fn, R = 1000)


Bootstrap Statistics :
        original        bias     std. error
t1* 56.900099702  3.511640e-02 2.0300222526
t2* -0.466189630 -7.080834e-04 0.0324241984
t3*  0.001230536  2.840324e-06 0.0001172164
```

```r
summary(lm(mpg ~ horsepower + I(horsepower^2), data = Auto))$coef
```

```
                    Estimate   Std. Error   t value      Pr(>|t|)
(Intercept)     56.900099702 1.8004268063  31.60367 1.740911e-109
horsepower      -0.466189630 0.0311246171 -14.97816  2.289429e-40
I(horsepower^2)  0.001230536 0.0001220759  10.08009  2.196340e-21
```

##########exercise

```r
glm.fits <- glm(
  default ~ income + balance,
  data = Default, family = binomial, subset = train
)
glm.probs <- predict(glm.fits, Default, type = "response")

glm.pred <- rep("No", 10000)
glm.pred[glm.probs > .5] <- "Yes"

table(glm.pred, Default$default)
```

```
glm.pred   No  Yes
     No  9603  205
     Yes   64  128
```

```r
mean(glm.pred == Default$default)
```

```
[1] 0.9731
```

#b.

```r
set.seed(0)
train_indices <- sample(nrow(Default), 5000)
validation_indices <- setdiff(1:nrow(Default), train_indices)


glm.fits <- glm(default ~ income + balance, data = Default, family = binomial, subset = tr

glm.probs <- predict(glm.fits, newdata = Default[validation_indices, ], type = "response")

glm.pred <- ifelse(glm.probs > 0.5, "Yes", "No")

actual_defaults <- Default$default[validation_indices]

validation_error <- mean(glm.pred != actual_defaults)
print(validation_error)
```

```
[1] 0.0256
```

```r
for (i in 1:3) {
  set.seed(i)
  train_indices <- sample(nrow(Default), 5000)
  validation_indices <- setdiff(1:nrow(Default), train_indices)


  glm.fits <- glm(default ~ income + balance, data = Default, family = binomial, subset = 
  glm.probs <- predict(glm.fits, newdata = Default[validation_indices, ], type = "response
  glm.pred <- ifelse(glm.probs > 0.5, "Yes", "No")
  actual_defaults <- Default$default[validation_indices]
```

```
  validation_error <- mean(glm.pred != actual_defaults)
  cat(sprintf("Validation error for seed %d: %.4f\n", i, validation_error))
}
```

```
Validation error for seed 1: 0.0254
Validation error for seed 2: 0.0238
Validation error for seed 3: 0.0264
```

#d

```
set.seed(0)
glm.fits <- glm(default ~ income + balance + student, data = Default, family = binomial, s

glm.probs <- predict(glm.fits, newdata = Default[validation_indices, ], type = "response")

glm.pred <- ifelse(glm.probs > 0.5, "Yes", "No")

actual_defaults <- Default$default[validation_indices]

validation_error <- mean(glm.pred != actual_defaults)
print(validation_error)
```

```
[1] 0.0272
```

#6. #summary() and glm()

```
glm.fits <- glm(
  default ~ income + balance,
  data = Default, family = "binomial"
)
summary(glm.fits)
```

```
Call:
glm(formula = default ~ income + balance, family = "binomial",
    data = Default)

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
```

```r
summary(glm.fits)$coefficients[, 2] #std error for coeffcients
```

```
 (Intercept)       income      balance 
4.347564e-01 4.985167e-06 2.273731e-04 
```

#b.

```r
boot.fn <- function(data, index) {
  fit <- glm(default ~ income + balance, data = data, subset = index, family = "binomial")
  return(c(coef(fit)['income'],
         coef(fit)['balance']))
}
set.seed(1)
indices <- sample(nrow(Default), size = 10000, replace = TRUE)
boot.fn(Default,indices)
```

```
      income      balance 
0.0000281529 0.0058371168 
```

#c.

```r
set.seed(1)
boot(Default, statistic = boot.fn, R = 100)
```

```
ORDINARY NONPARAMETRIC BOOTSTRAP


Call:
boot(data = Default, statistic = boot.fn, R = 100)


Bootstrap Statistics :
        original        bias     std. error
t1* 2.080898e-05 -3.993598e-07 4.186088e-06
t2* 5.647103e-03 -4.116657e-06 2.226242e-04
```

#the std errors are quite close #7.

```r
glm.fits <- glm(
  Direction ~ Lag1 + Lag2 ,
  data = Weekly, family = binomial
)
summary(glm.fits)
```

```
Call:
glm(formula = Direction ~ Lag1 + Lag2, family = binomial, data = Weekly)

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  0.22122    0.06147   3.599 0.000319 ***
Lag1        -0.03872    0.02622  -1.477 0.139672
Lag2         0.06025    0.02655   2.270 0.023232 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1496.2  on 1088  degrees of freedom
Residual deviance: 1488.2  on 1086  degrees of freedom
AIC: 1494.2

Number of Fisher Scoring iterations: 4
```

```r
glm.fits <- glm(
  Direction ~ Lag1 + Lag2 ,
  data = Weekly[-1, ], family = binomial
)
summary(glm.fits)
```

```
Call:
glm(formula = Direction ~ Lag1 + Lag2, family = binomial, data = Weekly[-1,
    ])

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  0.22324    0.06150   3.630 0.000283 ***
Lag1        -0.03843    0.02622  -1.466 0.142683
Lag2         0.06085    0.02656   2.291 0.021971 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1494.6  on 1087  degrees of freedom
Residual deviance: 1486.5  on 1085  degrees of freedom
AIC: 1492.5

Number of Fisher Scoring iterations: 4
```

```r
glm.pred[glm.probs > .5] <- "Yes"

mean(glm.pred == Default$default)
```

```
[1] 0.9529
```

#c

```r
glm.probs <- predict(glm.fits, Weekly[1, ],
                     type = "response")

glm.pred <- ifelse(glm.probs > 0.5, "Up", "Down")
```

```r
mean(glm.pred == Weekly[1, 'Direction'])
```

```
[1] 0
```

#not correctly classifed #d.

```r
errors <- numeric(nrow(Weekly))

for (i in 1:nrow(Weekly)) {
  glm.fits <- glm(Direction ~ Lag1 + Lag2, data = Weekly[-i, ], family = binomial)

  glm.probs <- predict(glm.fits, newdata = Weekly[i, ], type = "response")

  glm.pred <- ifelse(glm.probs > 0.5, "Up", "Down")

  errors[i] <- as.numeric(glm.pred != Weekly$Direction[i])
}

error_rate <- mean(errors)
print(error_rate)
```

```
[1] 0.4499541
```

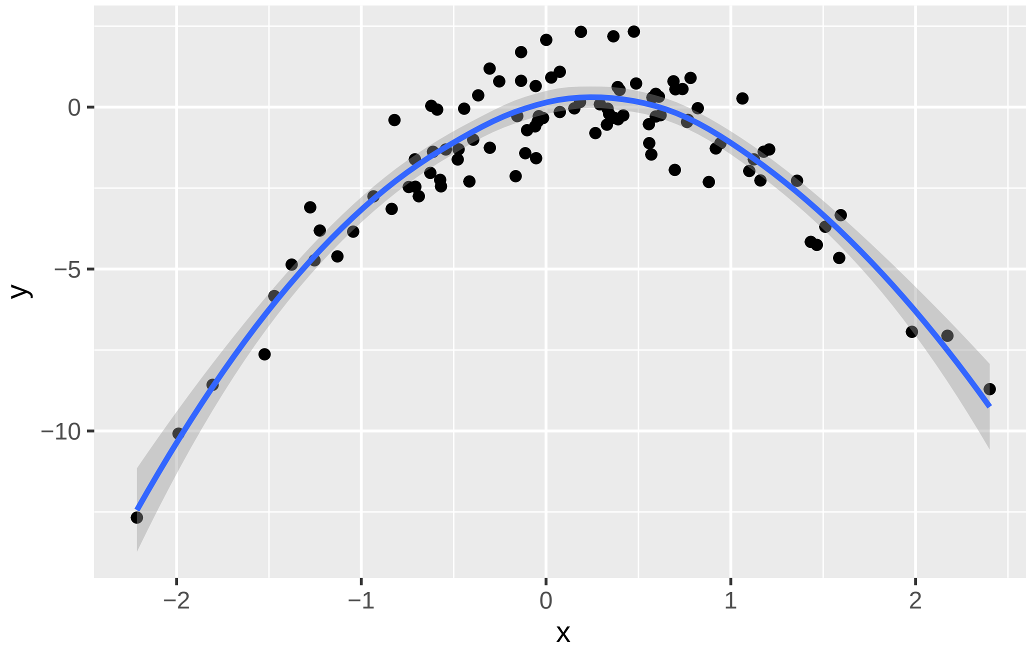
#8.cross-validation

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

```r
data <- data.frame(x, y)

ggplot(data, aes(x = x, y = y)) +
  geom_point()+
  geom_smooth()
```

```
`geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```




```
set.seed(1)
cv_errors <- numeric(4)

for (i in 1:4) {
  glm.fit <- glm(y ~ poly(x, i), data = data)

  cv.err <- cv.glm(data, glm.fit, K = nrow(data))
  cv_errors[i] <- cv.err$delta[1]
}

print(cv_errors)
```

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

```
set.seed(6)
cv_errors <- numeric(4)

for (i in 1:4) {
  glm.fit <- glm(y ~ poly(x, i), data = data)

  cv.err <- cv.glm(data, glm.fit, K = nrow(data))
```

```
  cv_errors[i] <- cv.err$delta[1]
}

print(cv_errors)
```

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

#no changes when set.seed change, the cv_errors aren't random #the quadratic model

```
models <- list()
summaries <- list()

for (i in 1:4) {
  glm.fit <- glm(y ~ poly(x, i), data = data)

  models[[i]] <- glm.fit
  summaries[[i]] <- summary(glm.fit)
}

for (i in 1:4) {
  cat(sprintf("\nSummary for model with polynomial degree %d:\n", i))
  print(summaries[[i]])
}
```

```
Summary for model with polynomial degree 1:

Call:
glm(formula = y ~ poly(x, i), data = data)

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   -1.550      0.260  -5.961 3.95e-08 ***
poly(x, i)     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 for model with polynomial degree 2:

Call:
glm(formula = y ~ poly(x, i), data = data)

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  -1.5500     0.0958  -16.18  < 2e-16 ***
poly(x, i)1   6.1888     0.9580    6.46 4.18e-09 ***
poly(x, i)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 for model with polynomial degree 3:

Call:
glm(formula = y ~ poly(x, i), data = data)

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  -1.55002    0.09626 -16.102  < 2e-16 ***
poly(x, i)1   6.18883    0.96263   6.429 4.97e-09 ***
poly(x, i)2 -23.94830    0.96263 -24.878  < 2e-16 ***
poly(x, i)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 for model with polynomial degree 4:

Call:
glm(formula = y ~ poly(x, i), data = data)

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  -1.55002    0.09591 -16.162  < 2e-16 ***
poly(x, i)1   6.18883    0.95905   6.453 4.59e-09 ***
poly(x, i)2 -23.94830    0.95905 -24.971  < 2e-16 ***
poly(x, i)3   0.26411    0.95905   0.275    0.784
poly(x, i)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
```

#base on the p-value from above the quadratic model's still fit better than others #the result agrees with LOOCV

#######9.Boston

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

```
[1] 22.53281
```

#b

```r
sd(Boston$medv)/sqrt(length(Boston$medv))
```

```
[1] 0.4088611
```

########c bootstrap

```r
set.seed(6)
mean.fn <- function(data, index) {
  return(mean(data$medv[index]))
}
bootstrap_results <-boot(Boston, mean.fn, R = 10000)
bootstrap_results
```

```
ORDINARY NONPARAMETRIC BOOTSTRAP


Call:
boot(data = Boston, statistic = mean.fn, R = 10000)


Bootstrap Statistics :
    original       bias    std. error
t1* 22.53281 -0.004025119   0.4061429
```

```r
boot.ci(bootstrap_results, type = "basic")
```

```
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 10000 bootstrap replicates

CALL :
boot.ci(boot.out = bootstrap_results, type = "basic")

Intervals :
Level      Basic
95%   (21.72, 23.33 )
Calculations and Intervals on Original Scale
```

```r
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

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

```
[1] 21.2
```

```r
set.seed(6)
median.fn <- function(data, index) {
  return(median(data$medv[index]))
}
bootstrap_results <-boot(Boston, median.fn, R = 10000)
bootstrap_results
```

```
ORDINARY NONPARAMETRIC BOOTSTRAP


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


Bootstrap Statistics :
    original    bias    std. error
t1*     21.2 -0.011945    0.380989
```

#g

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

```
  10%
12.75
```

#h

```r
set.seed(6)
tenth.fn <- function(data, index) {
  return(quantile(data$medv[index], 0.1))
}

bootstrap_results <-boot(Boston, tenth.fn, R = 10000)
bootstrap_results
```

```
ORDINARY NONPARAMETRIC BOOTSTRAP


Call:
boot(data = Boston, statistic = tenth.fn, R = 10000)


Bootstrap Statistics :
    original   bias    std. error
t1*    12.75 0.00577   0.4996871
```