

ISL Ch5 Lab and Exercises

```
library(tidyverse)

## -- Attaching packages ----- tidyverse 1.3.0.9000 --
## v ggplot2 3.3.0      v purrr  0.3.3
## v tibble  3.0.1      v dplyr  0.8.5
## v tidyr   1.0.2      v stringr 1.4.0
## v readr   1.3.1      v forcats 0.5.0

## -- Conflicts ----- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()     masks stats::lag()

set.seed(1)
x <- 1:12
# a random permutation
is.integer(sample(x))

## [1] TRUE

# bootstrap resampling -- only if length(x) > 1 !
sample(x, replace = TRUE)

## [1] 5 10 6 10 7 9 5 5 9 9 5 5

# 100 Bernoulli trials
sample(c(0,1), 100, replace = TRUE)

## [1] 1 1 1 0 1 0 0 1 0 1 1 1 1 0 1 1 1 1 0 0 1 0 1 1 0 0 1 1 1 0 0 1 1 1 1 1
## [38] 1 0 1 1 1 1 0 0 0 1 1 0 0 1 1 1 0 0 0 1 0 1 0 1 0 0 1 1 0 0 0 1 1 0 1 1 1
## [75] 0 0 0 0 1 0 0 0 0 0 1 1 1 1 0 1 1 1 1 0 0 1 0 0 1 1

# create divide a dataframe into folds using samples:
(theDf <- tibble(x=1:56))

## # A tibble: 56 x 1
##       x
##   <int>
## 1     1
## 2     2
## 3     3
## 4     4
## 5     5
## 6     6
## 7     7
## 8     8
## 9     9
## 10    10
## # ... with 46 more rows
```

```

k = 10
folds <- sample(1:k, size = nrow(theDf), replace = T) #10 fold CV
table(folds)

## folds
##  1  2  3  4  5  6  7  8  9 10
##  8  6 10  3  4  5  5  5  4  6

# folds with same size
sameSizefolds <- sample(rep(1:k, length.out = nrow(theDf)), size = nrow(theDf), replace = F)
table(sameSizefolds)

## sameSizefolds
##  1  2  3  4  5  6  7  8  9 10
##  6  6  6  6  6  6  5  5  5  5

# train data set
theDf[folds != 3, ]

## # A tibble: 46 x 1
##       x
##   <int>
## 1     1
## 2     2
## 3     3
## 4     4
## 5     5
## 6     6
## 7     7
## 8     8
## 9     9
## 10    10
## # ... with 36 more rows

# test data set
theDf[folds == 3, ]

## # A tibble: 10 x 1
##       x
##   <int>
## 1    13
## 2    15
## 3    23
## 4    25
## 5    27
## 6    28
## 7    41
## 8    43
## 9    53
## 10   56

## More careful bootstrapping -- Consider this when using sample()
## programmatically (i.e., in your function or simulation)!

# sample()'s surprise -- example
# x <- 1:10
#   sample(x[x > 8]) # length 2

```

```
# sample(x[x > 9]) # oops -- length 10: If x has length 1, is numeric and x >= 1, sampling via sam
# sample(x[x > 10]) # length 0
#
# ## safer version:
# x[sample.int(length(x)) > 8] # length 2
# x[sample.int(length(x)) > 9] # length 1
# x[sample.int(length(x)) > 10] # length 0
#
# ## R 3.x.y only
# sample.int(1e10, 12, replace = TRUE)
# sample.int(1e10, 12) # not that there is much chance of duplicates
```

```
library(tidyverse)
# Sample fixed number per group
auto.df = read.csv("/Users/shahrdadshadab/env/my-R-project/ISLR/Data/Auto.csv", header=T, stringsAsFacto
(auto.df = as_tibble(auto.df))
```

```
## # A tibble: 397 x 9
##   mpg cylinders displacement horsepower weight acceleration year origin
##   <dbl>      <int>      <dbl>      <int>  <int>      <dbl> <int> <int>
## 1    18         8        307        130   3504         12     70     1
## 2    15         8        350        165   3693        11.5    70     1
## 3    18         8        318        150   3436         11     70     1
## 4    16         8        304        150   3433         12     70     1
## 5    17         8        302        140   3449        10.5    70     1
## 6    15         8        429        198   4341         10     70     1
## 7    14         8        454        220   4354          9     70     1
## 8    14         8        440        215   4312         8.5    70     1
## 9    14         8        455        225   4425         10     70     1
## 10   15         8        390        190   3850         8.5    70     1
## # ... with 387 more rows, and 1 more variable: name <fct>
```

```
(by_cyl <- auto.df %>%
  group_by(year))
```

```
## # A tibble: 397 x 9
## # Groups:   year [13]
##   mpg cylinders displacement horsepower weight acceleration year origin
##   <dbl>      <int>      <dbl>      <int>  <int>      <dbl> <int> <int>
## 1    18         8        307        130   3504         12     70     1
## 2    15         8        350        165   3693        11.5    70     1
## 3    18         8        318        150   3436         11     70     1
## 4    16         8        304        150   3433         12     70     1
## 5    17         8        302        140   3449        10.5    70     1
## 6    15         8        429        198   4341         10     70     1
## 7    14         8        454        220   4354          9     70     1
## 8    14         8        440        215   4312         8.5    70     1
## 9    14         8        455        225   4425         10     70     1
## 10   15         8        390        190   3850         8.5    70     1
## # ... with 387 more rows, and 1 more variable: name <fct>
```

```
sample_n(auto.df, 10)
```

```
## # A tibble: 10 x 9
```

```
##      mpg cylinders displacement horsepower weight acceleration year origin
##      <dbl>      <int>      <dbl>      <int> <int>      <dbl> <int> <int>
## 1  29          4          97          75  2171          16    75     3
## 2  36.4        5         121          67  2950          19.9   80     2
## 3  15          8         350         145  4082          13    73     1
## 4  33.7        4         107          75  2210          14.4   81     3
## 5  19          4         122          85  2310          18.5   73     1
## 6  29          4          97          78  1940          14.5   77     2
## 7  19          4         121         112  2868          15.5   73     2
## 8  15          8         304         150  3892          12.5   72     1
## 9  10          8         307         200  4376          15    70     1
## 10 14          8         351         153  4154          13.5   71     1
## # ... with 1 more variable: name <fct>
```

```
sample_n(auto.df, 50, replace = TRUE)
```

```
## # A tibble: 50 x 9
##      mpg cylinders displacement horsepower weight acceleration year origin
##      <dbl>      <int>      <dbl>      <int> <int>      <dbl> <int> <int>
## 1  24          4         113          95  2372          15    70     3
## 2  31.9        4          89          71  1925          14    79     2
## 3  21          4         122          86  2226          16.5   72     1
## 4  32          4         144          96  2665          13.9   82     3
## 5  32          4          71          65  1836          21    74     3
## 6  16          6         225         105  3439          15.5   71     1
## 7  36          4         107          75  2205          14.5   82     3
## 8  34.5        4         100          NA   2320          15.8   81     2
## 9  18          4         121         112  2933          14.5   72     2
## 10 12          8         400         167  4906          12.5   73     1
## # ... with 40 more rows, and 1 more variable: name <fct>
```

```
sample_n(auto.df, 10, weight = as.integer(mpg))
```

```
## # A tibble: 10 x 9
##      mpg cylinders displacement horsepower weight acceleration year origin
##      <dbl>      <int>      <dbl>      <int> <int>      <dbl> <int> <int>
## 1  24          4         140          92  2865          16.4   82     1
## 2  17          8         260         110  4060          19    77     1
## 3  38          4          91          67  1995          16.2   82     3
## 4  20.5        6         200          95  3155          18.2   78     1
## 5  21.5        6         231         115  3245          15.4   79     1
## 6  27          4         112          88  2640          18.6   82     1
## 7  27.2        4         119          97  2300          14.7   78     3
## 8  25.1        4         140          88  2720          15.4   78     1
## 9  14          8         340         160  3609           8    70     1
## 10 32          4          91          67  1965          15.7   82     3
## # ... with 1 more variable: name <fct>
```

```
sample_n(by_cyl, 3)
```

```
## # A tibble: 39 x 9
## # Groups:   year [13]
##      mpg cylinders displacement horsepower weight acceleration year origin
##      <dbl>      <int>      <dbl>      <int> <int>      <dbl> <int> <int>
## 1  14          8         455         225  4425          10    70     1
## 2  26          4          97          46  1835          20.5   70     2
```

```
## 3 15 8 350 165 3693 11.5 70 1
## 4 19 6 232 100 2634 13 71 1
## 5 14 8 400 175 4464 11.5 71 1
## 6 28 4 140 90 2264 15.5 71 1
## 7 18 4 121 112 2933 14.5 72 2
## 8 28 4 98 80 2164 15 72 1
## 9 23 4 97 54 2254 23.5 72 2
## 10 24 4 121 110 2660 14 73 2
## # ... with 29 more rows, and 1 more variable: name <fct>
```

```
sample_n(by_cyl, 10, replace = TRUE)
```

```
## # A tibble: 130 x 9
## # Groups:   year [13]
##   mpg cylinders displacement horsepower weight acceleration year origin
##   <dbl>     <int>     <dbl>     <int>  <int>     <dbl> <int> <int>
## 1 18         8       318       150   3436       11    70    1
## 2 15         8       390       190   3850       8.5    70    1
## 3 15         8       429       198   4341       10    70    1
## 4 11         8       318       210   4382       13.5   70    1
## 5 14         8       454       220   4354        9    70    1
## 6 18         6       199        97   2774       15.5   70    1
## 7 18         8       307       130   3504       12    70    1
## 8 25         4       104        95   2375       17.5   70    2
## 9 18         8       307       130   3504       12    70    1
## 10 18        8       318       150   3436       11    70    1
## # ... with 120 more rows, and 1 more variable: name <fct>
```

```
sample_n(by_cyl, 3, weight = mpg / mean(mpg))
```

```
## # A tibble: 39 x 9
## # Groups:   year [13]
##   mpg cylinders displacement horsepower weight acceleration year origin
##   <dbl>     <int>     <dbl>     <int>  <int>     <dbl> <int> <int>
## 1 22         6       198        95   2833       15.5   70    1
## 2 10         8       307       200   4376       15    70    1
## 3 15         8       400       150   3761       9.5    70    1
## 4 35         4        72        69   1613       18    71    3
## 5 19         6       232       100   2634       13    71    1
## 6 26         4        91        70   1955       20.5   71    1
## 7 28         4        98        80   2164       15    72    1
## 8 21         4       120        87   2979       19.5   72    2
## 9 12         8       350       160   4456       13.5   72    1
## 10 14        8       318       150   4237       14.5   73    1
## # ... with 29 more rows, and 1 more variable: name <fct>
```

```
# Sample fixed fraction per group
# Default is to sample all data = randomly resample rows
sample_frac(auto.df)
```

```
## # A tibble: 397 x 9
##   mpg cylinders displacement horsepower weight acceleration year origin
##   <dbl>     <int>     <dbl>     <int>  <int>     <dbl> <int> <int>
## 1 29         4        90        70   1937       14    75    2
## 2 14         8       454       220   4354        9    70    1
## 3 18         3        70        90   2124       13.5   73    3
```

```
## 4 10      8      307      200  4376      15      70      1
## 5 30.5     4       97       78  2190     14.1     77      2
## 6 23       8      350      125  3900     17.4     79      1
## 7 20.2     6      200       85  2965     15.8     78      1
## 8 18       6      225      105  3613     16.5     74      1
## 9 13       8      350      175  4100     13       73      1
## 10 13      8      400      175  5140     12       71      1
## # ... with 387 more rows, and 1 more variable: name <fct>
```

```
sample_frac(auto.df, 0.1)
```

```
## # A tibble: 40 x 9
##   mpg cylinders displacement horsepower weight acceleration year origin
##   <dbl>     <int>     <dbl>     <int>  <int>     <dbl> <int> <int>
## 1 26.4         4       140        88   2870     18.1    80     1
## 2 32.2         4       108        75   2265     15.2    80     3
## 3 16          8      351       149   4335     14.5    77     1
## 4 28          4       116        90   2123     14       71     2
## 5 27.5         4       134        95   2560     14.2    78     3
## 6 27          4        97        60   1834     19       71     2
## 7 27.2         4       141        71   3190     24.8    79     2
## 8 20          6       156       122   2807     13.5    73     3
## 9 19          4       121       112   2868     15.5    73     2
## 10 25         4       104        95   2375     17.5    70     2
## # ... with 30 more rows, and 1 more variable: name <fct>
```

```
sample_frac(auto.df, 1.5, replace = TRUE)
```

```
## # A tibble: 596 x 9
##   mpg cylinders displacement horsepower weight acceleration year origin
##   <dbl>     <int>     <dbl>     <int>  <int>     <dbl> <int> <int>
## 1 44          4        97        52   2130     24.6    82     2
## 2 18          4       121       112   2933     14.5    72     2
## 3 25          4       140        92   2572     14.9    76     1
## 4 27.5         4       134        95   2560     14.2    78     3
## 5 19          4       122        85   2310     18.5    73     1
## 6 21          6       155       107   2472     14       73     1
## 7 22          6       225       100   3233     15.4    76     1
## 8 32.3         4        97        67   2065     17.8    81     3
## 9 15.5         8      350       170   4165     11.4    77     1
## 10 30.5        4        98        63   2051     17       77     1
## # ... with 586 more rows, and 1 more variable: name <fct>
```

```
sample_frac(auto.df, 0.1, weight = 1 / mpg)
```

```
## # A tibble: 40 x 9
##   mpg cylinders displacement horsepower weight acceleration year origin
##   <dbl>     <int>     <dbl>     <int>  <int>     <dbl> <int> <int>
## 1 26          4        97        46   1950     21       73     2
## 2 28.4         4       151        90   2670     16       79     1
## 3 16          6      250       100   3278     18       73     1
## 4 25          4       116        81   2220     16.9    76     2
## 5 15          8      350       165   3693     11.5    70     1
## 6 16          6       225       105   3439     15.5    71     1
## 7 24          4       134        96   2702     13.5    75     3
## 8 18          6       171        97   2984     14.5    75     1
```

```
## 9 29.8      4      134      90 2711      15.5 80      3
## 10 20      8      262     110 3221      13.5 75      1
## # ... with 30 more rows, and 1 more variable: name <fct>
```

```
sample_frac(by_cyl, 0.2)
```

```
## # A tibble: 81 x 9
## # Groups:   year [13]
##   mpg cylinders displacement horsepower weight acceleration year origin
##   <dbl>     <int>     <dbl>     <int>  <int>     <dbl> <int> <int>
## 1    15         8      429      198  4341      10    70     1
## 2    24         4      113      95   2372      15    70     3
## 3    26         4      121     113   2234     12.5   70     2
## 4    15         8      390     190   3850      8.5    70     1
## 5    27         4       97      88   2130     14.5   70     3
## 6    15         8      383     170   3563      10    70     1
## 7    35         4       72      69   1613      18    71     3
## 8    14         8      400     175   4464     11.5   71     1
## 9    25         4      113      95   2228      14    71     3
## 10   13         8      400     170   4746      12    71     1
## # ... with 71 more rows, and 1 more variable: name <fct>
```

```
sample_frac(by_cyl, 1, replace = TRUE)
```

```
## # A tibble: 397 x 9
## # Groups:   year [13]
##   mpg cylinders displacement horsepower weight acceleration year origin
##   <dbl>     <int>     <dbl>     <int>  <int>     <dbl> <int> <int>
## 1    14         8      455     225  4425      10    70     1
## 2    15         8      350     165  3693     11.5   70     1
## 3    14         8      455     225  4425      10    70     1
## 4    15         8      383     170  3563      10    70     1
## 5    14         8      440     215  4312      8.5    70     1
## 6    26         4      121     113  2234     12.5   70     2
## 7    15         8      429     198  4341      10    70     1
## 8    16         8      304     150  3433      12    70     1
## 9    10         8      307     200  4376      15    70     1
## 10   15         8      390     190  3850      8.5    70     1
## # ... with 387 more rows, and 1 more variable: name <fct>
```

```
library(tidyverse)
```

```
auto.df = read.csv("/Users/shahrdadshadab/env/my-R-project/ISLR/Data/Auto.csv", header=T, stringsAsFactors=FALSE)
(auto.df = as_tibble(auto.df))
```

```
## # A tibble: 397 x 9
##   mpg cylinders displacement horsepower weight acceleration year origin
##   <dbl>     <int>     <dbl>     <int>  <int>     <dbl> <int> <int>
## 1    18         8      307     130  3504      12    70     1
## 2    15         8      350     165  3693     11.5   70     1
## 3    18         8      318     150  3436      11    70     1
## 4    16         8      304     150  3433      12    70     1
## 5    17         8      302     140  3449     10.5   70     1
## 6    15         8      429     198  4341      10    70     1
## 7    14         8      454     220  4354       9    70     1
## 8    14         8      440     215  4312      8.5    70     1
```

```
## 9      14      8      455      225  4425      10      70      1
## 10     15      8      390      190  3850      8.5      70      1
## # ... with 387 more rows, and 1 more variable: name <fct>
```

```
# Check a particular column that has NA and include the record
# dfSubsetWithNaInOneCol <- auto.df[is.na(auto.df$Directions), ]
# head(dfSubsetWithNaInOneCol)
```

```
# get a subset of records in dataframe with no NA in any column:
auto.df <- auto.df[rowSums(is.na(auto.df)) == 0, ]
```

```
# Now find a subset of records that have at least one NA
auto.df[rowSums(is.na(auto.df)) > 0,]
```

```
## # A tibble: 0 x 9
## # ... with 9 variables: mpg <dbl>, cylinders <int>, displacement <dbl>,
## #   horsepower <int>, weight <int>, acceleration <dbl>, year <int>,
## #   origin <int>, name <fct>
```

```
set.seed(1)
```

```
train <- sample(dim(auto.df)[1],196) # choose a sample of size 196 from row indices of dataframe auto.d.
```

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

```
# test error MSE
```

```
mean((auto.df$mpg - predict(lm.fit, auto.df))[-train]^2, na.rm = T)
```

```
## [1] 23.26601
```

```
# calculate test.mse for polynomial regression
```

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

```
# test error MSE
```

```
mean((auto.df$mpg - predict(lm.fit2, auto.df))[-train]^2, na.rm = T)
```

```
## [1] 18.71646
```

```
# calculate test.mse for cubic regression
```

```
lm.fit3 <- lm(mpg~poly(horsepower, 3), data=auto.df, subset = train)
```

```
# test error MSE
```

```
mean((auto.df$mpg - predict(lm.fit3, auto.df))[-train]^2, na.rm = T)
```

```
## [1] 18.79401
```

```
# if we sample again and create another training sample MSE values will be different
```

```
train <- sample(dim(auto.df)[1],196) # choose a sample of size 196 from row indices of dataframe auto.d.
```

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

```
# test error MSE
```

```
mean((auto.df$mpg - predict(lm.fit, auto.df))[-train]^2, na.rm = T)
```

```
## [1] 26.83974
```



```

# calculate test.mse for polynomial regression

lm.fit2 <- lm(mpg~poly(horsepower, 2), data=auto.df, subset = train)

# test error MSE
mean((auto.df$mpg - predict(lm.fit2, auto.df))[-train]^2, na.rm = T)

## [1] 19.56785

# calculate test.mse for cubic regression

lm.fit3 <- lm(mpg~poly(horsepower, 3), data=auto.df, subset = train)

# test error MSE
mean((auto.df$mpg - predict(lm.fit3, auto.df))[-train]^2, na.rm = T)

## [1] 19.62272

library(tidyverse)
library(boot)

auto.df = read.csv("/Users/shahrdadshadab/env/my-R-project/ISLR/Data/Auto.csv", header=T, stringsAsFactors=F)
auto.df = as_tibble(auto.df)

# remove NAs
# is.na(auto.df)
is.matrix(is.na(auto.df))

## [1] TRUE

# rowSums(is.na(auto.df))

auto.df <- auto.df[rowSums(is.na(auto.df)) == 0,]

# now fit the data on the whole data
glm.fit <- glm(mpg ~ horsepower, data = auto.df)

cv.err <- cv.glm(auto.df, glm.fit)
str(cv.err)

## List of 4
## $ call : language cv.glm(data = auto.df, glmfit = glm.fit)
## $ K : num 392
## $ delta: num [1:2] 24.2 24.2
## $ seed : int [1:626] 10403 583 1654269195 -1877109783 -961256264 1403523942 124639233 261424787 183...

cv.err$delta

## [1] 24.23151 24.23114

# redo this for polynomials and save the results in a vector
cv.error <- NULL
for (i in 1:5){
  glm.fit <- glm(mpg ~ poly(horsepower,i), data = auto.df)
  cv.error <- rbind(cv.error, cv.glm(auto.df, glm.fit)$delta)
}
cv.error

```

```

##           [,1]      [,2]
## [1,] 24.23151 24.23114
## [2,] 19.24821 19.24787
## [3,] 19.33498 19.33448
## [4,] 19.42443 19.42371
## [5,] 19.03321 19.03242

library(tidyverse)
library(class)
library(boot)

set.seed(17)
weekly.df = read.csv("/Users/shahrdadshadab/env/my-R-project/ISLR/Data/datasets/Weekly.csv",
                     header=T, stringsAsFactors = T, na.strings = "?")
weekly.df = tibble(weekly.df)

# create k-fold
k <- 10
threshold <- 0.5

folds <- sample(1:k, size = nrow(weekly.df), replace = T) #10 fold CV
table(folds)

## folds
##   1   2   3   4   5   6   7   8   9  10
## 100 106 106 116 111 107 108  99 111 125

# folds with same size
# sameSizefolds <- sample(rep(1:k, length.out = nrow(weekly.df)), size = nrow(weekly.df), replace = F)
# table(sameSizefolds)

# Run the model by making the model on 9 folds and predicting on the hold out:

results <- lapply(1:k, function(x){ # x is the index of test portion, the rest are for training
  glm.fit <- glm(Direction ~ Lag2, data = weekly.df[folds != x,], family = binomial)
  glm.probs <- predict(glm.fit, weekly.df[folds == x,], type = "response")
  # since contrasts(weekly.df$Direction) shows dummy variable 1 assigned to 'Up'
  # and since P(y=1|x) is glm.probs what we get is posterior of probability of 'Up' case
  glm.pred <- ifelse(glm.probs > threshold, "Up", "Down")
  return(data.frame(probs = glm.probs, predicted = glm.pred, real = weekly.df[folds == x,]$Direction))
})

# calculate confusion table and other measures

missclassificationRate = NULL
nullClassificationRate = NULL
FP_rates = NULL
TP_rates = NULL
precisions = NULL
specificities = NULL
confusionTables = NULL
aucs = NULL

library(pROC)

## Type 'citation("pROC")' for a citation.

```

```

##
## Attaching package: 'pROC'

## The following objects are masked from 'package:stats':
##
##      cov, smooth, var

for( df in results){
  confusion_table <- table(df$predicted, df$real)

  nullClassifier <- max(
    (confusion_table[1,1] + confusion_table[2,1])/(confusion_table[1,1] + confusion_table[2,1] + confusion_table[1,2] + confusion_table[2,2])
    (confusion_table[1,2] + confusion_table[2,2])/(confusion_table[1,1] + confusion_table[2,1] + confusion_table[1,2] + confusion_table[2,2])

  nullClassificationRate <- c(nullClassificationRate, nullClassifier)

  roc_obj <- roc(df$real, df$probs)
  aucs <- c(aucs, auc(roc_obj))

  confusionTables <- cbind(confusionTables, confusion_table)
  missclassificationRate <- c(missclassificationRate, mean(df$predicted != df$real))
  FP_rates <- c(FP_rates, confusion_table[2,1]/(confusion_table[2,1] + confusion_table[1,1]))
  TP_rates <- c(TP_rates, confusion_table[2,2]/(confusion_table[2,2] + confusion_table[1,2]))
  precisions <- c(precisions, confusion_table[2,2] / (confusion_table[2,2] + confusion_table[2,1]))
  specificities <- c(specificities , 1 - confusion_table[2,1]/(confusion_table[2,1] + confusion_table[1,1]))

  # overall fraction of wrong predictions:
  # print(confusion_table)
}

## Setting levels: control = Down, case = Up
## Setting direction: controls > cases
## Setting levels: control = Down, case = Up
## Setting direction: controls > cases
## Setting levels: control = Down, case = Up
## Setting direction: controls > cases
## Setting levels: control = Down, case = Up
## Setting direction: controls < cases
## Setting levels: control = Down, case = Up
## Setting direction: controls < cases
## Setting levels: control = Down, case = Up
## Setting direction: controls < cases
## Setting levels: control = Down, case = Up
## Setting direction: controls < cases
## Setting levels: control = Down, case = Up
## Setting direction: controls > cases
## Setting levels: control = Down, case = Up

```

```

## Setting direction: controls < cases
## Setting levels: control = Down, case = Up
## Setting direction: controls < cases
# average missclassification error rate
sprintf("Logistic Regression : Missclassification error rate : %s", mean(missclassificationRate))

## [1] "Logistic Regression : Missclassification error rate : 0.436181472329902"
sprintf("Logistic regression : Null Classifier: %s", mean(nullClassificationRate))

## [1] "Logistic regression : Null Classifier: 0.562748219282808"
sprintf("Logistic Regression AUC: %s", mean (aucs))

## [1] "Logistic Regression AUC: 0.545318579186755"
# FP rate:
sprintf("Logistic Regression : FP rate (TypeI error, 1 - specificity) : %s", mean(FP_rates))

## [1] "Logistic Regression : FP rate (TypeI error, 1 - specificity) : 0.935210201833379"
# TP rate:
sprintf("Logistic Regression : TP rate (1-TypeII error, power, sensetivity, recall) : %s", mean(TP_rates))

## [1] "Logistic Regression : TP rate (1-TypeII error, power, sensetivity, recall) : 0.962289778137507"
# precision:
sprintf("Logistic Regression : precision: %s", mean(precisions))

## [1] "Logistic Regression : precision: 0.562532496546535"
# specificity 1-FP/N:
sprintf("Logistic Regression : specificity 1-FP/N: %s", mean(specificities))

## [1] "Logistic Regression : specificity 1-FP/N: 0.0647897981666212"
library(tidyverse)
library(class)
library(boot)

weekly.df = read.csv("/Users/shahrdadshadab/env/my-R-project/ISLR/Data/datasets/Weekly.csv",
                     header=T, stringsAsFactors = T, na.strings = "?")
weekly.df = tibble(weekly.df)

# train <- (weekly.df$Year >= 1990 & weekly.df$Year <= 2008)
# test.Y <- weekly.df[!train,]$Direction
# test.X <- weekly.df[!train,]
# train.Y <- weekly.df[train,]$Direction
#
# train.X <- weekly.df[train,]

boot.fn <- function (df, index){
  lda.fit <- MASS::lda(Direction ~ Lag2, data = df,family = binomial, subset = index)
  lda.fit$scaling
}

```

```

# First estimate the coefficients on the full set
boot.fn(weekly.df, 1:nrow(weekly.df))

##          LD1
## Lag2 0.4251523

# we can use the function to create bootstrap estimate for LDA coefficient
# by randomly sampling from among the observations with replacement
set.seed(17)
boot.fn(weekly.df, sample(nrow(weekly.df), nrow(weekly.df), replace=T))

##          LD1
## Lag2 0.402531

boot.fn(weekly.df, sample(nrow(weekly.df), nrow(weekly.df), replace=T))

##          LD1
## Lag2 0.4010238

boot.fn(weekly.df, sample(nrow(weekly.df), nrow(weekly.df), replace=T))

##          LD1
## Lag2 0.4339658

boot.fn(weekly.df, sample(nrow(weekly.df), nrow(weekly.df), replace=T))

##          LD1
## Lag2 0.4161635

# next we plugin the function into 'boot()' to compute SE[] of 1000 bootstrap estimates for the LDA coe.
boot(data=weekly.df, statistic = boot.fn, R = 1000)

##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = weekly.df, statistic = boot.fn, R = 1000)
##
##
## Bootstrap Statistics :
##      original      bias    std. error
## t1* 0.4251523 0.001264081 0.01746007

library(tidyverse)
library(class)
library(boot)
set.seed(1)

default.df = read.csv("/Users/shahrdadshadab/env/my-R-project/ISLR/Data/Default.csv",
                      header=T, stringsAsFactors = T, na.strings = "?")
default.df = tibble(default.df)
colnames(default.df)

## [1] "default" "student" "balance" "income"

str(default.df)

## tibble [10,000 x 4] (S3: tbl_df/tbl/data.frame)

```

```
## $ default: Factor w/ 2 levels "No","Yes": 1 1 1 1 1 1 1 1 1 1 ...
## $ student: Factor w/ 2 levels "No","Yes": 1 2 1 1 1 2 1 2 1 1 ...
## $ balance: num [1:10000] 730 817 1074 529 786 ...
## $ income : num [1:10000] 44362 12106 31767 35704 38463 ...
```

```
# a)
```

```
glm.fit <- glm(default ~ balance + income, data = default.df, family = binomial)
sprintf("summary of logistic regression: ")
```

```
## [1] "summary of logistic regression: "
```

```
summary(glm.fit)
```

```
##
## Call:
## glm(formula = default ~ balance + income, family = binomial,
##      data = default.df)
##
## 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 ***
## balance      5.647e-03  2.274e-04  24.836  < 2e-16 ***
## income       2.081e-05  4.985e-06   4.174  2.99e-05 ***
## ---
## 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)
```

```
# get a random sample
```

```
train <- sample(nrow(default.df), nrow(default.df)/2)
```

```
# let's create a function that get the full data and a subset of indices as training set
# and return the miss classification error rate on the validation set
```

```
classify <- function(df , train.indices){
```

```
# train on random set
```

```
glm.fit <- glm(default ~ balance + income, data = default.df, family = binomial, subset=train.indices)
```

```
# obtain the prediction of default in validation set
```

```
contrasts(default.df$default)
```

```
# since contrasts(default.df$default) shows dummy variable 1 assigned to "Yes"
```

```
# since P(y=1/x) is actually glm.probs what we get is posterior of probability of default = Yes
```

```

glm.probs <- predict(glm.fit, default.df[-train.indices, ], type = "response")

# convert posterior probabilities into "Yes" and "No"
glm.pred <- ifelse(glm.probs > 0.5, "Yes", "No")
stopifnot(length(glm.pred) == length(default.df[-train.indices, ]$default))

(confusion_matrix <- table(glm.pred, default.df[-train.indices, ]$default))

# validation set missclassification error rate
mean(glm.pred != default.df[-train.indices, ]$default)
}

classify(default.df, train)

## [1] 0.0254

# c)

train <- sample(nrow(default.df), nrow(default.df)/3)
classify(default.df, train)

## [1] 0.02759862

train <- sample(nrow(default.df), 2*nrow(default.df)/3)
classify(default.df, train)

## [1] 0.0284943

train <- sample(nrow(default.df), 4*nrow(default.df)/5)
classify(default.df, train)

## [1] 0.026

train <- sample(nrow(default.df), nrow(default.df)/5)
classify(default.df, train)

## [1] 0.026625

# in general all error rates are around 0.025

# d)
classify1 <- function(df , train.indices){

  # train on random set
  glm.fit <- glm(default ~ balance + income + student,
                 data = default.df, family = binomial, subset=train.indices)

  # obtain the prediction of default in validation set
  contrasts(default.df$default)

  # since contrasts(default.df$default) shows dummy variable 1 assigned to "Yes"
  # since  $P(y=1|x)$  is actually glm.probs what we get is posterior of probability of default = Yes

  glm.probs <- predict(glm.fit, default.df[-train.indices, ], type = "response")

  # convert posterior probabilities into "Yes" and "No"

```

```

glm.pred <- ifelse(glm.probs > 0.5, "Yes", "No")
stopifnot(length(glm.pred) == length(default.df[-train.indices, ]$default))

(confusion_matrix <- table(glm.pred, default.df[-train.indices, ]$default))

# validation set missclassification error rate
mean(glm.pred != default.df[-train.indices, ]$default)
}

train <- sample(nrow(default.df), nrow(default.df)/2)
classify(default.df, train)

```

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

```
# adding student actually increased the error rate
```

```

library(tidyverse)
library(class)
library(boot)
set.seed(1)

default.df = read.csv("/Users/shahrdadshadab/env/my-R-project/ISLR/Data/Default.csv",
                      header=T, stringsAsFactors = T, na.strings = "?")
default.df = tibble(default.df)
colnames(default.df)

```

```
## [1] "default" "student" "balance" "income"
```

```
glm.fit <- glm(default ~ balance + income, data = default.df, family = binomial)
```

```

# a)
summary(glm.fit)

```

```

##
## Call:
## glm(formula = default ~ balance + income, family = binomial,
##      data = default.df)
##
## 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 ***
## balance      5.647e-03  2.274e-04  24.836  < 2e-16 ***
## income       2.081e-05  4.985e-06   4.174  2.99e-05 ***
## ---
## 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(df, index) {
  glm.fit <- glm(default ~ balance + income ,data = default.df, family = binomial, subset = index)
  coefficients(glm.fit)
}

# c)
# next we plugin the boot.fn function into 'boot()' to compute SE[] of 1000
# bootstrap estimates for the logistic regression coefficients
(result <- boot(data=weekly.df, statistic = boot.fn, R = 100))

##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = weekly.df, statistic = boot.fn, R = 100)
##
##
## Bootstrap Statistics :
##      original      bias    std. error
## t1* -1.170135e+01 -4.555716e-01 1.337812e+00
## t2*  5.662070e-03  2.097998e-04 6.699781e-04
## t3*  3.355312e-05  2.448351e-06 1.542786e-05
print("----- Here are the values by capturing output:----- ")

## [1] "----- Here are the values by capturing output:----- "
library(stringr)

(x <- capture.output(result)) # store the output as text

## [1] ""
## [2] "ORDINARY NONPARAMETRIC BOOTSTRAP"
## [3] ""
## [4] ""
## [5] "Call:"
## [6] "boot(data = weekly.df, statistic = boot.fn, R = 100)"
## [7] ""
## [8] ""
## [9] "Bootstrap Statistics :"
## [10] "      original      bias    std. error"
## [11] "t1* -1.170135e+01 -4.555716e-01 1.337812e+00"
## [12] "t2*  5.662070e-03  2.097998e-04 6.699781e-04"
## [13] "t3*  3.355312e-05  2.448351e-06 1.542786e-05"
(x <- str_extract(x , "^t1.*$")) # grab the line that starts with t1

## [1] NA
## [2] NA
## [3] NA
## [4] NA
## [5] NA

```

```

## [6] NA
## [7] NA
## [8] NA
## [9] NA
## [10] NA
## [11] "t1* -1.170135e+01 -4.555716e-01 1.337812e+00"
## [12] NA
## [13] NA

(x <- x[!is.na(x)]) # remove all the lines we don't need

## [1] "t1* -1.170135e+01 -4.555716e-01 1.337812e+00"
# (se <- as.numeric(unlist(str_extract_all(x, '[0-9.]+$')))) # extract the final value (se)

# d)

# For balance SE is shrink from 0.005647 to 0.000709
# For income SE is changed from 2.081e-05 to 1.443805e-05 not much change

library(tidyverse)
library(class)
weekly.df = read.csv("/Users/shahrdadshadab/env/my-R-project/ISLR/Data/datasets/Weekly.csv", header=T, )
weekly.df = tibble(weekly.df)

# a) Logistic regression using full data set:
glm.fit1 <- glm(Direction ~ Lag1 + Lag2, data = weekly.df, family = binomial)

# b) Logistic regression using full data set but the first observation:
glm.fit2 <- glm(Direction ~ Lag1 + Lag2, data = weekly.df, family = binomial, subset=(1:nrow(weekly.df)))

# c) Predict first observation using model b

# first lets see the contrasts of Direction to know what is assigned to 1 and which is 2
contrasts(weekly.df$Direction)

##      Up
## Down  0
## Up    1

# Contrasts shows Down is 0 and Up is 1
# Since Posterior is  $P(Y=1|X)$  Tuse if posterior > 0.5 it should be "Up"

glm.probs <- predict(glm.fit2, weekly.df[1,], type = "response")
(glm.predict <- ifelse(glm.probs > 0.5 , "Up", "Down"))

##      1
## "Up"
weekly.df[1,]$Direction

## [1] Down
## Levels: Down Up

```

```

# d)
# First observation is not correctly classified
# errors
errorList = NULL

for(i in 1:nrow(weekly.df)){
  glm.fit <- glm(Direction ~ Lag1 + Lag2, data = weekly.df, family = binomial, subset = (1:nrow(weekly.df) != i))
  # Predict ith observation using model that is trained on all records but the ith one
  glm.probs <- predict(glm.fit, weekly.df[i,], type = "response")
  (glm.predict <- ifelse(glm.probs > 0.5 , "Up", "Down"))

  # accumulate errors
  errorList <-c(errorList , glm.predict != weekly.df[i,]$Direction)
}

# e)
sprintf("average LOOC error rate: %s", mean(errorList) )

## [1] "average LOOC error rate: 0.449954086317723"

library(tidyverse)
library(class)
library(boot)

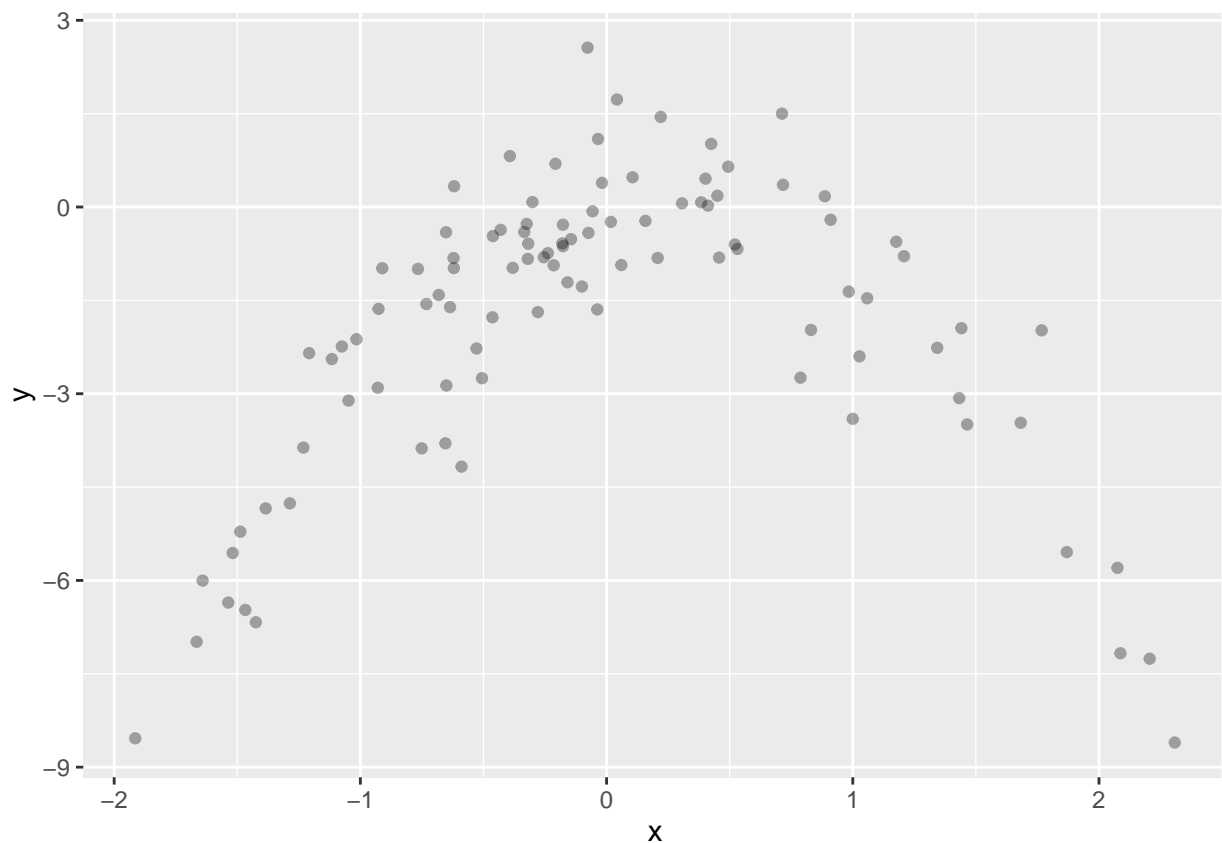
# a ) generate simulated data set:
set.seed(1)
y = rnorm(100)
x = rnorm(100)
y = x - 2*x^2+rnorm(100)

df <- tibble(y = y, x = x)
head(df)

## # A tibble: 6 x 2
##       y       x
##   <dbl> <dbl>
## 1 -0.981 -0.620
## 2  1.73   0.0421
## 3 -0.984 -0.911
## 4 -0.223  0.158
## 5 -3.80  -0.655
## 6 -1.98   1.77

# b)
df %>%
  ggplot(mapping = aes(x=x, y=y))+
  geom_point(alpha=1/3)

```



```
# c)
set.seed(1)
errors <- NULL
for (i in 1:4){
  glm.fit <- glm(y ~ poly(x, i), data=df)
  errors <- rbind(errors, cv.glm(df, glm.fit)$delta)
}
errors
```

```
##           [,1]      [,2]
## [1,] 5.890979 5.888812
## [2,] 1.086596 1.086326
## [3,] 1.102585 1.102227
## [4,] 1.114772 1.114334
```

```
# d) repeat c with another random seed
set.seed(17)
errors <- NULL
for (i in 1:4){
  glm.fit <- glm(y ~ poly(x, i), data=df)
  summary(glm.fit)
  errors <- rbind(errors, cv.glm(df, glm.fit)$delta)
}
errors
```

```
##           [,1]      [,2]
## [1,] 5.890979 5.888812
## [2,] 1.086596 1.086326
```

```
## [3,] 1.102585 1.102227
## [4,] 1.114772 1.114334

# The result is the same for different seed vlues becuse there is no random
# componenet in LOOC procedure
```

```
# f)
```

```
for (i in 1:4){
  glm.fit <- glm(y ~ poly(x, i), data=df)
  print("----- Ploy Model of oder ----- ")
  print(i)

  print(summary(glm.fit))
}
```

```
## [1] "----- Ploy Model of oder ----- "
## [1] 1
##
## Call:
## glm(formula = y ~ poly(x, i), data = df)
##
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -7.3469  -0.9275   0.8028   1.5608   4.3974
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -1.8277     0.2362  -7.737 9.18e-12 ***
## poly(x, i)    2.3164     2.3622   0.981  0.329
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 5.580018)
##
##      Null deviance: 552.21  on 99  degrees of freedom
## Residual deviance: 546.84  on 98  degrees of freedom
## AIC: 459.69
##
## Number of Fisher Scoring iterations: 2
##
## [1] "----- Ploy Model of oder ----- "
## [1] 2
##
## Call:
## glm(formula = y ~ poly(x, i), data = df)
##
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -2.89884  -0.53765   0.04135   0.61490   2.73607
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -1.8277     0.1032 -17.704 <2e-16 ***
## poly(x, i)1    2.3164     1.0324   2.244  0.0271 *
```

```

## poly(x, i)2 -21.0586      1.0324 -20.399   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 1.06575)
##
##      Null deviance: 552.21  on 99  degrees of freedom
## Residual deviance: 103.38  on 97  degrees of freedom
## AIC: 295.11
##
## Number of Fisher Scoring iterations: 2
##
## [1] "----- Ploy Model of oder ----- "
## [1] 3
##
## Call:
## glm(formula = y ~ poly(x, i), data = df)
##
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -2.87250  -0.53881   0.02862   0.59383   2.74350
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -1.8277      0.1037  -17.621  <2e-16 ***
## poly(x, i)1   2.3164      1.0372   2.233   0.0279 *
## poly(x, i)2 -21.0586      1.0372 -20.302  <2e-16 ***
## poly(x, i)3  -0.3048      1.0372  -0.294   0.7695
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 1.075883)
##
##      Null deviance: 552.21  on 99  degrees of freedom
## Residual deviance: 103.28  on 96  degrees of freedom
## AIC: 297.02
##
## Number of Fisher Scoring iterations: 2
##
## [1] "----- Ploy Model of oder ----- "
## [1] 4
##
## Call:
## glm(formula = y ~ poly(x, i), data = df)
##
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -2.8914  -0.5244   0.0749   0.5932   2.7796
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -1.8277      0.1041  -17.549  <2e-16 ***
## poly(x, i)1   2.3164      1.0415   2.224   0.0285 *
## poly(x, i)2 -21.0586      1.0415 -20.220  <2e-16 ***

```

```
## poly(x, i)3 -0.3048      1.0415 -0.293  0.7704
## poly(x, i)4 -0.4926      1.0415 -0.473  0.6373
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 1.084654)
##
##      Null deviance: 552.21  on 99  degrees of freedom
## Residual deviance: 103.04  on 95  degrees of freedom
## AIC: 298.78
##
## Number of Fisher Scoring iterations: 2

# Clearly in all the models only beta 1 and beta 2 are statistically signifocant which matches
# The result from LOOCV that shows ploynomial of order 2 has smallest error

library(tidyverse)
library(class)
library(boot)
boston.df = read.csv("/Users/shahrdadshadab/env/my-R-project/ISLR/Data/Boston.csv",
                     header=T, stringsAsFactors = T, na.strings = "?")
boston.df = tibble(boston.df)
str(boston.df)

## tibble [506 x 14] (S3: tbl_df/tbl/data.frame)
##  $ crim   : num [1:506] 0.00632 0.02731 0.02729 0.03237 0.06905 ...
##  $ zn     : num [1:506] 18 0 0 0 0 12.5 12.5 12.5 12.5 ...
##  $ indus  : num [1:506] 2.31 7.07 7.07 2.18 2.18 2.18 7.87 7.87 7.87 ...
##  $ chas   : int [1:506] 0 0 0 0 0 0 0 0 0 ...
##  $ nox    : num [1:506] 0.538 0.469 0.469 0.458 0.458 0.458 0.524 0.524 0.524 ...
##  $ rm     : num [1:506] 6.58 6.42 7.18 7 7.15 ...
##  $ age    : num [1:506] 65.2 78.9 61.1 45.8 54.2 58.7 66.6 96.1 100 85.9 ...
##  $ dis    : num [1:506] 4.09 4.97 4.97 6.06 6.06 ...
##  $ rad    : int [1:506] 1 2 2 3 3 3 5 5 5 ...
##  $ tax    : int [1:506] 296 242 242 222 222 222 311 311 311 ...
##  $ ptratio: num [1:506] 15.3 17.8 17.8 18.7 18.7 18.7 15.2 15.2 15.2 ...
##  $ black  : num [1:506] 397 397 393 395 397 ...
##  $ lstat  : num [1:506] 4.98 9.14 4.03 2.94 5.33 ...
##  $ medv   : num [1:506] 24 21.6 34.7 33.4 36.2 28.7 22.9 27.1 16.5 18.9 ...

# a)
(muHat <- mean (boston.df$medv))

## [1] 22.53281

# b)
(SE_of_muHat <- sd(boston.df$medv) / sqrt(nrow(boston.df)))

## [1] 0.4088611

# c)
boot.fn <- function(df, index) mean (df[index, ]$medv)

(result <- boot(data=boston.df, statistic = boot.fn, R = 1000))

##
## ORDINARY NONPARAMETRIC BOOTSTRAP
```

```
##
##
## Call:
## boot(data = boston.df, statistic = boot.fn, R = 1000)
##
##
## Bootstrap Statistics :
##      original      bias      std. error
## t1* 22.53281 -0.01080613    0.416494

# comparing with part b) , bootstrap error is a bit larger

# d)
# lets calculate 95% confidence interval for the muHat estimator:
# [muHat - 2*SE[muHat], muHat + 2*SE[muHat]]
(leftBound <- 22.53281 - 2 * 0.4251931)

## [1] 21.68242

(rightBound <- 22.53281 + 2 * 0.4251931)

## [1] 23.3832

# e) provide an estimate for median value of the population based on the data set
(muHatMed <- median(boston.df$medv))

## [1] 21.2

# f) calculate SE of muhatMed using bootstrap

boot.fn <- function(df, index) median(df[index, ]$medv)
(result <- boot(data=boston.df, statistic = boot.fn, R = 1000))

##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = boston.df, statistic = boot.fn, R = 1000)
##
##
## Bootstrap Statistics :
##      original      bias      std. error
## t1*      21.2 -0.00995    0.3755923

# g) 10th percentile
percentiles <- quantile(boston.df$medv, probs = c(10, 25, 50, 75, 100)/100)
print("----- 10th percentile mdev: ----- ")

## [1] "----- 10th percentile mdev: ----- "

(muHat01 <- percentiles[1])

##      10%
## 12.75

# h) use bootstrap to find SE error for 10th percentile
boot.fn <- function(df, index) quantile(df[index, ]$medv, probs = c(10, 25, 50, 75, 100)/100)[1]
print("----- bootstrap 10th percentile mdev: ----- ")
```



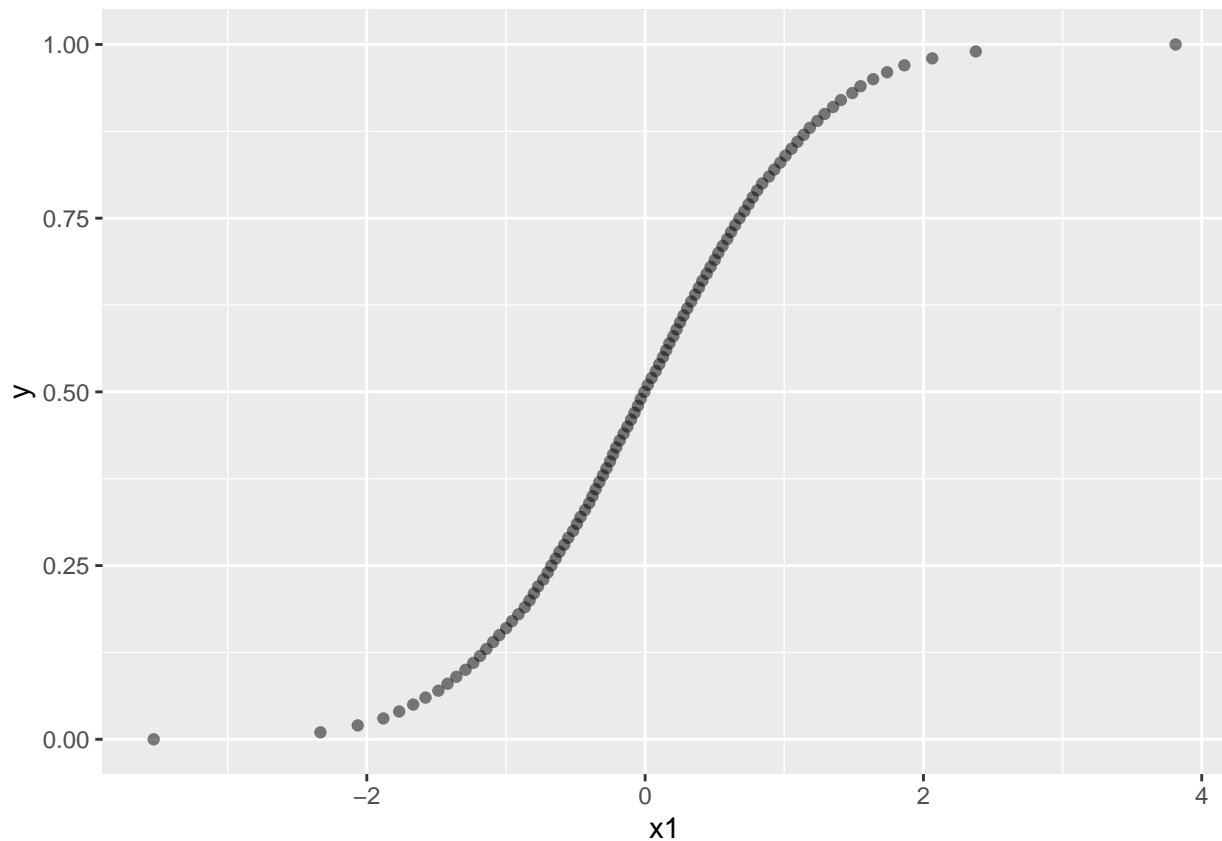
```
## [1] "----- bootstrap 10th percentile mdev: ----- "
```

```
(result <- boot(data=boston.df, statistic = boot.fn, R = 10000))
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = boston.df, statistic = boot.fn, R = 10000)
##
##
## Bootstrap Statistics :
##      original    bias      std. error
## t1*      12.75 0.000315      0.497474
```

```
set.seed(10)
x <- rnorm(10000)

tibble(x1 = quantile(x, probs = seq(0,1,0.01), type=1), y = seq(0,1,0.01)) %>%
  ggplot(mapping = aes(x=x1, y=y))+
  geom_point(alpha=1/2)
```



```
quantile(x) # Extremes & Quartiles by default
```

```
##           0%          25%          50%          75%          100%
## -3.531562865 -0.673447402 -0.005305007  0.678582296  3.812580336
```

```
quantile(x, probs = c(0, 0.1, 0.5, 1, 2, 5, 10, 25, 50, 75, 100)/100)
```

```
##           0%          0.1%          0.5%          1%          2%          5%
## -3.531562865 -3.011770339 -2.600016820 -2.329913624 -2.061742013 -1.665475833
##           10%          25%          50%          75%          100%
## -1.290686953 -0.673447402 -0.005305007  0.678582296  3.812580336
```

```
### Compare different types
```

```
quantAll <- function(x, prob, ...)
  t(vapply(1:9, function(typ) quantile(x, prob=prob, type = typ, ...), quantile(x, prob, type=1)))
p <- c(0.1, 0.5, 1, 2, 5, 10, 50)/100
signif(quantAll(x, p), 4)
```

```
##           0.1%    0.5%     1%     2%     5%    10%     50%
## [1,] -3.012 -2.601 -2.333 -2.065 -1.667 -1.291 -0.005325
## [2,] -3.012 -2.600 -2.332 -2.063 -1.666 -1.291 -0.005305
## [3,] -3.012 -2.601 -2.333 -2.065 -1.667 -1.291 -0.005325
## [4,] -3.012 -2.601 -2.333 -2.065 -1.667 -1.291 -0.005325
## [5,] -3.012 -2.600 -2.332 -2.063 -1.666 -1.291 -0.005305
## [6,] -3.012 -2.601 -2.333 -2.065 -1.667 -1.291 -0.005305
## [7,] -3.012 -2.600 -2.330 -2.062 -1.665 -1.291 -0.005305
## [8,] -3.012 -2.600 -2.332 -2.064 -1.666 -1.291 -0.005305
## [9,] -3.012 -2.600 -2.332 -2.064 -1.666 -1.291 -0.005305
```

```
## for complex numbers:
```

```
z <- complex(re=x, im = -10*x)
signif(quantAll(z, p), 4)
```

```
##           0.1%    0.5%     1%     2%     5%
## [1,] -3.01+30.12i -2.6+26.01i -2.33+23.33i -2.07+20.65i -1.67+16.67i
## [2,] -3.01+30.12i -2.6+26.00i -2.33+23.32i -2.06+20.63i -1.67+16.66i
## [3,] -3.01+30.12i -2.6+26.01i -2.33+23.33i -2.07+20.65i -1.67+16.67i
## [4,] -3.01+30.12i -2.6+26.01i -2.33+23.33i -2.07+20.65i -1.67+16.67i
## [5,] -3.01+30.12i -2.6+26.00i -2.33+23.32i -2.06+20.63i -1.67+16.66i
## [6,] -3.01+30.12i -2.6+26.01i -2.33+23.33i -2.07+20.65i -1.67+16.67i
## [7,] -3.01+30.12i -2.6+26.00i -2.33+23.30i -2.06+20.62i -1.67+16.65i
## [8,] -3.01+30.12i -2.6+26.00i -2.33+23.32i -2.06+20.64i -1.67+16.66i
## [9,] -3.01+30.12i -2.6+26.00i -2.33+23.32i -2.06+20.64i -1.67+16.66i
##           10%          50%
## [1,] -1.29+12.91i -0.00533+0.05325i
## [2,] -1.29+12.91i -0.00531+0.05305i
## [3,] -1.29+12.91i -0.00533+0.05325i
## [4,] -1.29+12.91i -0.00533+0.05325i
## [5,] -1.29+12.91i -0.00531+0.05305i
## [6,] -1.29+12.91i -0.00531+0.05305i
## [7,] -1.29+12.91i -0.00531+0.05305i
## [8,] -1.29+12.91i -0.00531+0.05305i
## [9,] -1.29+12.91i -0.00531+0.05305i
```