Assignment 2: Data and Index

STA3005 Statistical Computing

Due date: Mar 3, 11:59pm

- Name: **Yuntao Xue**
- Student ID: **122040070**
- Collaborated with: **Li Zichen**

You can collaborate with your classmates, but you must identify their names above, and you must submit **your own** lab as an knitted (complied) PDF or html file on blackboard system.

States data set

Below we construct a data frame, of 50 states by 10 variables. The first 8 variables are numeric and the last 2 are factors. The numeric variables here come from the built-in `state.x77` matrix, which records various demographic factors on 50 US states, measured in the 1970s. You can learn more about this state data set by typing `?state.x77` into your R console.

```
state.df = data.frame(state.x77, Region=state.region, Division=state.division)
```

Q1. Basic data frame manipulations

- **1a.** Add a column to `state.df`, containing the state abbreviations that are stored in the built-in vector `state.abb`. Name this column `Abbr`. You can do this in (at least) two ways: by using a call to `data.frame()`, or by directly defining `state.df$Abbr`. Display the first 5 rows and all 11 columns of the new `state.df`.

```
# YOUR CODE GOES HERE
state.df$abbr <- state.abb
head(state.df, 5)
```

```
##            Population Income Illiteracy Life.Exp Murder HS.Grad Frost   Area
## Alabama          3615   3624        2.1    69.05   15.1    41.3    20  50708
## Alaska            365   6315        1.5    69.31   11.3    66.7   152 566432
## Arizona          2212   4530        1.8    70.55    7.8    58.1    15 113417
## Arkansas         2110   3378        1.9    70.66   10.1    39.9    65  51945
## California      21198   5114        1.1    71.71   10.3    62.6    20 156361
##                Region           Division abbr
## Alabama     South East South Central   AL
## Alaska        West            Pacific   AK
## Arizona       West           Mountain   AZ
## Arkansas    South West South Central   AR
## California    West            Pacific   CA
```

- **1b.** Remove the `Region` column from `state.df`. You can do this in (at least) two ways: by using negative indexing, or by directly setting `state.df$Region` to be `NULL`. Display the first 5 rows and all 10 columns of `state.df`.

```
# YOUR CODE GOES HERE
state.df$Region <- NULL
head(state.df, 5)
```

```
##            Population Income Illiteracy Life.Exp Murder HS.Grad Frost   Area
## Alabama          3615   3624        2.1    69.05   15.1    41.3    20  50708
## Alaska            365   6315        1.5    69.31   11.3    66.7   152 566432
## Arizona          2212   4530        1.8    70.55    7.8    58.1    15 113417
## Arkansas         2110   3378        1.9    70.66   10.1    39.9    65  51945
## California      21198   5114        1.1    71.71   10.3    62.6    20 156361
##                      Division abbr
## Alabama    East South Central   AL
## Alaska                Pacific   AK
## Arizona              Mountain   AZ
## Arkansas   West South Central   AR
## California            Pacific   CA
```

- **1c.** Add two columns to `state.df`, containing the x and y coordinates (longitude and latitude, respectively) of the center of the states, that are stored in the (existing) list `state.center`. Hint: take a look at this list in the console, to see what its elements are named. Name these two columns `Center.x` and `Center.y`. Display the first 5 rows and all 12 columns of `state.df`.

```
# YOUR CODE GOES HERE
state.df$Center.x <- state.center$x
state.df$Center.y <- state.center$y
head(state.df, 5)
```

```
##            Population Income Illiteracy Life.Exp Murder HS.Grad Frost   Area
## Alabama          3615   3624        2.1    69.05   15.1    41.3    20  50708
## Alaska            365   6315        1.5    69.31   11.3    66.7   152 566432
## Arizona          2212   4530        1.8    70.55    7.8    58.1    15 113417
## Arkansas         2110   3378        1.9    70.66   10.1    39.9    65  51945
## California      21198   5114        1.1    71.71   10.3    62.6    20 156361
##                      Division abbr  Center.x Center.y
## Alabama    East South Central   AL  -86.7509  32.5901
## Alaska                Pacific   AK -127.2500  49.2500
## Arizona              Mountain   AZ -111.6250  34.2192
## Arkansas   West South Central   AR  -92.2992  34.7336
## California            Pacific   CA -119.7730  36.5341
```

- **1d.** Make a new data frame which contains only those states whose longitude is less than -100. Do this in two different ways: using manual indexing, and `subset()`. Check that they are equal to each other, using an appropriate function call.

```
# YOUR CODE GOES HERE
df_new_1 <-
  state.df[state.df$Center.x < -100, ]
df_new_2 <-
  subset(state.df,
         Center.x < -100)
head(df_new_1, 5)
```

```
##            Population Income Illiteracy Life.Exp Murder HS.Grad Frost   Area
## Alaska            365   6315        1.5    69.31   11.3    66.7   152 566432
## Arizona          2212   4530        1.8    70.55    7.8    58.1    15 113417
## California      21198   5114        1.1    71.71   10.3    62.6    20 156361
## Colorado         2541   4884        0.7    72.06    6.8    63.9   166 103766
```

```
## Hawaii            868   4963        1.9    73.60    6.2    61.9     0   6425
##            Division abbr Center.x Center.y
## Alaska      Pacific   AK -127.250  49.2500
## Arizona    Mountain   AZ -111.625  34.2192
## California  Pacific   CA -119.773  36.5341
## Colorado   Mountain   CO -105.513  38.6777
## Hawaii      Pacific   HI -126.250  31.7500
```

```
head(df_new_2, 5)
```

```
##            Population Income Illiteracy Life.Exp Murder HS.Grad Frost   Area
## Alaska            365   6315        1.5    69.31   11.3    66.7   152 566432
## Arizona          2212   4530        1.8    70.55    7.8    58.1    15 113417
## California      21198   5114        1.1    71.71   10.3    62.6    20 156361
## Colorado         2541   4884        0.7    72.06    6.8    63.9   166 103766
## Hawaii            868   4963        1.9    73.60    6.2    61.9     0   6425
##            Division abbr Center.x Center.y
## Alaska      Pacific   AK -127.250  49.2500
## Arizona    Mountain   AZ -111.625  34.2192
## California  Pacific   CA -119.773  36.5341
## Colorado   Mountain   CO -105.513  38.6777
## Hawaii      Pacific   HI -126.250  31.7500
```

```
if(identical(df_new_1, df_new_2) == TRUE)
{cat("they're equal")}else
    {cat("they're not equal")}
```

```
## they're equal
```

- **1e.** Make a new data frame which contains only the states whose longitude is less than -100, and whose murder rate is above 9%. Print this new data frame to the console. Among the states in this new data frame, which has the highest average life expectancy?

```
# YOUR CODE GOES HERE
df_new <-
  state.df[state.df$Murder > 9
           &
             state.df$Center.x < -100, ]
print(df_new)
```

```
##            Population Income Illiteracy Life.Exp Murder HS.Grad Frost   Area
## Alaska            365   6315        1.5    69.31   11.3    66.7   152 566432
## California      21198   5114        1.1    71.71   10.3    62.6    20 156361
## Nevada            590   5149        0.5    69.03   11.5    65.2   188 109889
## New Mexico       1144   3601        2.2    70.32    9.7    55.2   120 121412
##            Division abbr Center.x Center.y
## Alaska      Pacific   AK -127.250  49.2500
## California  Pacific   CA -119.773  36.5341
## Nevada     Mountain   NV -116.851  39.1063
## New Mexico Mountain   NM -105.942  34.4764
```

```
max_LF <-
  max(df_new$Life.Exp)
id <-
  which.max(df_new$Life.Exp) # index of max value
cat("the highest life expectancy:",
    max_LF,
```

```
    "\n")
```

```
## the highest life expectancy: 71.71
```

```
cat("the index of the one has the highest life expectancy:",
    id,
    "\n")
```

```
## the index of the one has the highest life expectancy: 2
```

```
cat("the one has the highest life expectancy: California")
```

```
## the one has the highest life expectancy: California
```

Prostate cancer data set

Below we read in the prostate cancer data set in the first assignment. You can remind yourself about what's been measured by looking back it.

```
pros.dat = read.table("pros.dat")
```

Q2. Practice with the apply family

- **2a.** Using `sapply()`, calculate the mean of each variable. Also, calculate the standard deviation of each variable. Each should require just one line of code. Display your results.

```
# YOUR CODE GOES HERE
sapply(pros.dat, mean)
```

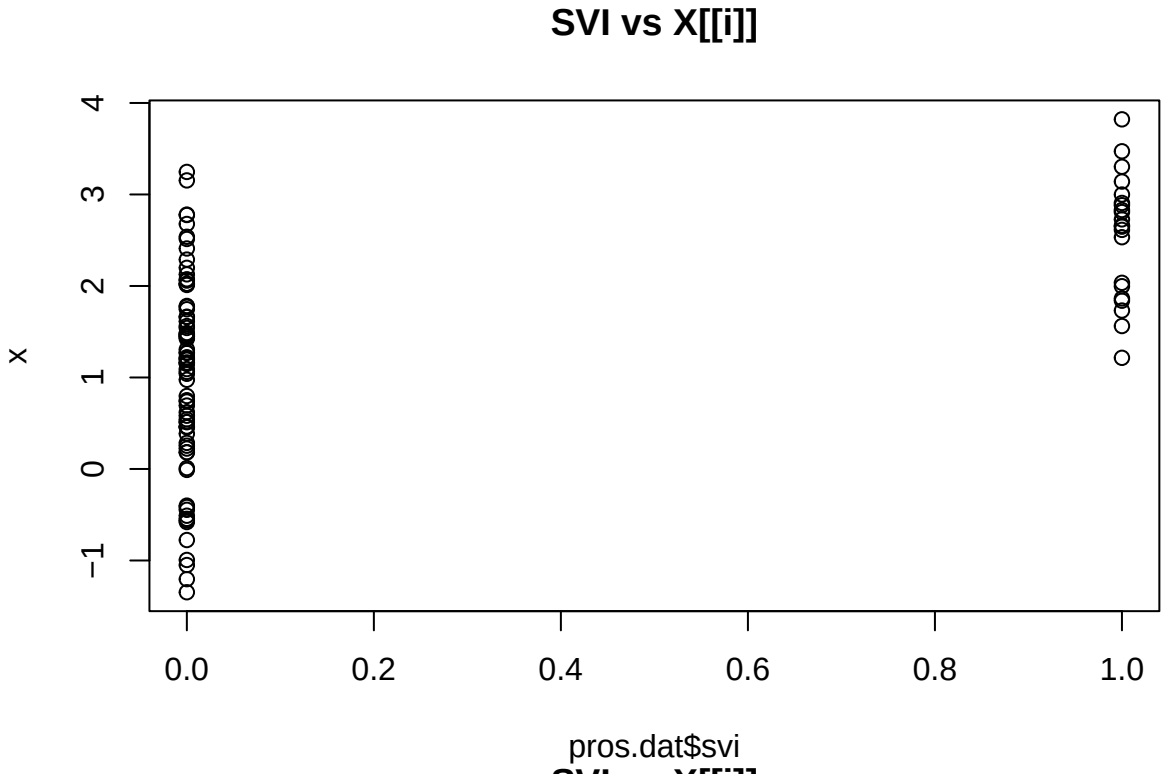
```
##     lcavol    lweight        age       lbph        svi        lcp    gleason
##  1.3500096  3.6289427 63.8659794  0.1003556  0.2164948 -0.1793656  6.7525773
##      pgg45       lpsa
## 24.3814433  2.4783869
```

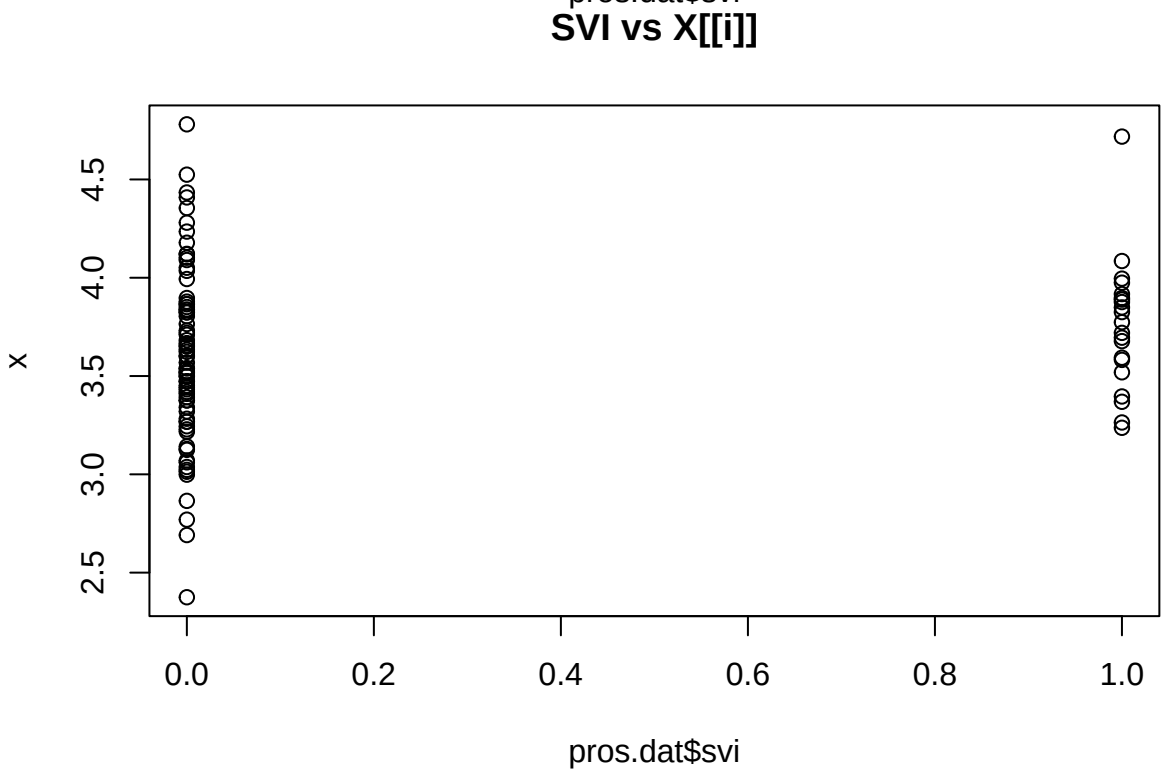
```
sapply(pros.dat, sd)
```

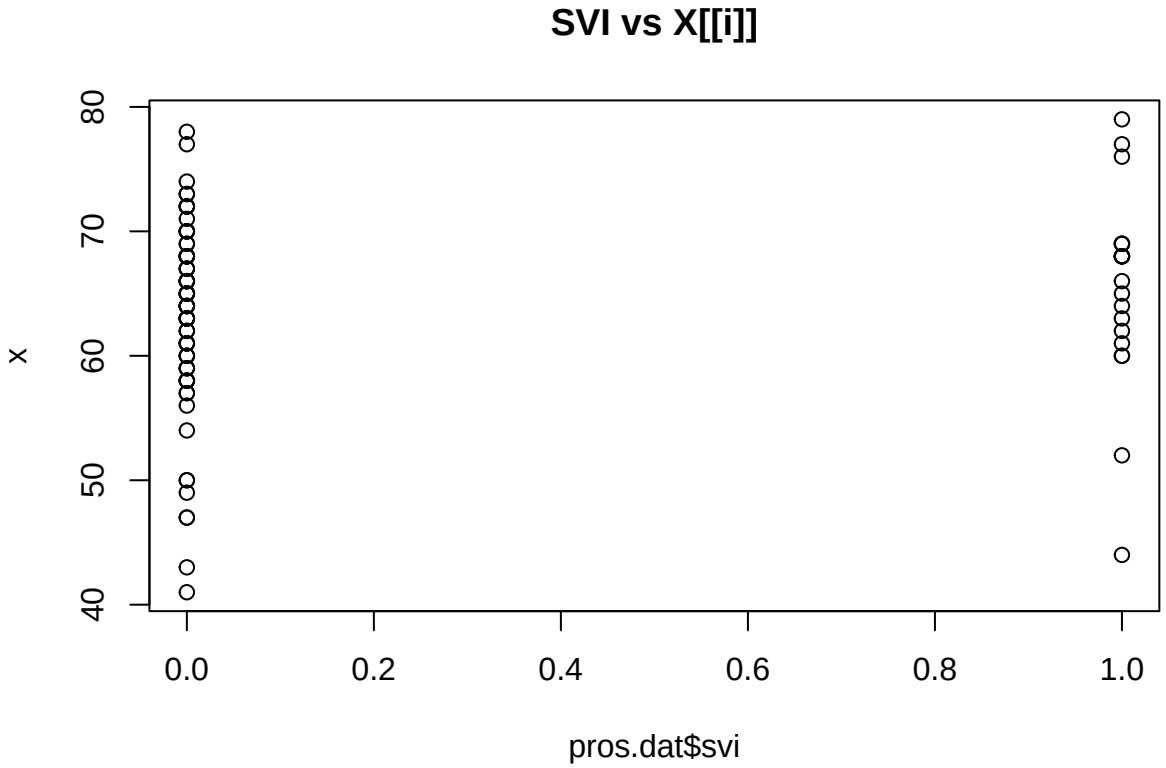
```
##     lcavol    lweight        age       lbph        svi        lcp    gleason
##  1.1786249  0.4284112  7.4451171  1.4508066  0.4139949  1.3982496  0.7221341
##      pgg45       lpsa
## 28.2040346  1.1543291
```

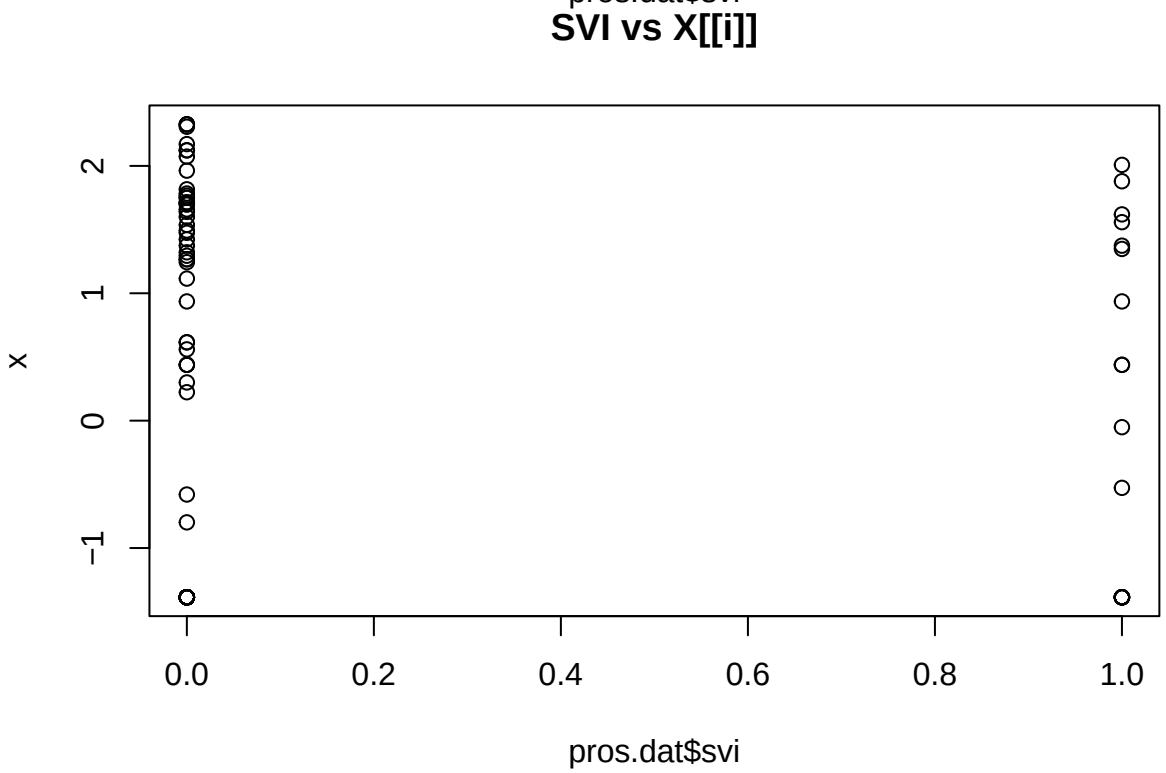
- **2b.** Let's plot each variable against SVI. Using `lapply()`, plot each column, excluding SVI, on the y-axis with SVI on the x-axis. This should require just one line of code.

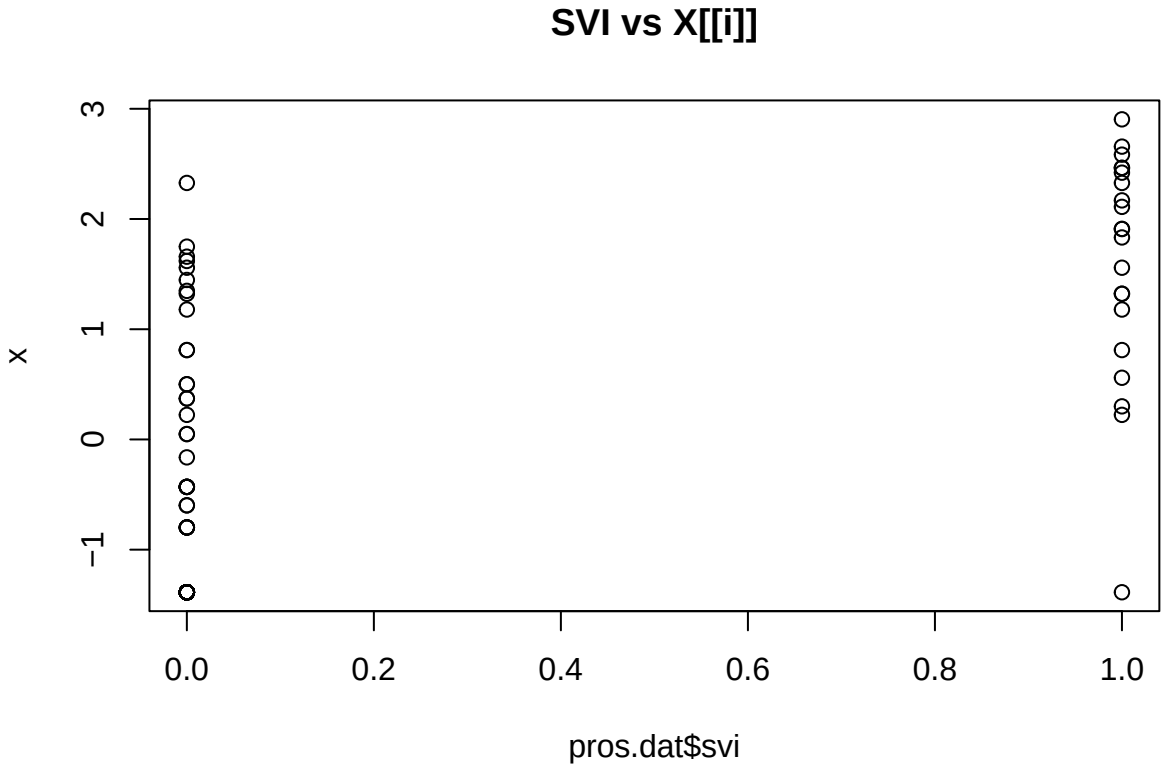
```
# YOUR CODE GOES HERE
lapply(pros.dat[ , !names(pros.dat) %in% "svi"],
       function(x) plot(pros.dat$svi,
                        x,
                        main = paste("SVI vs",
                                     deparse(substitute(x)))))
```

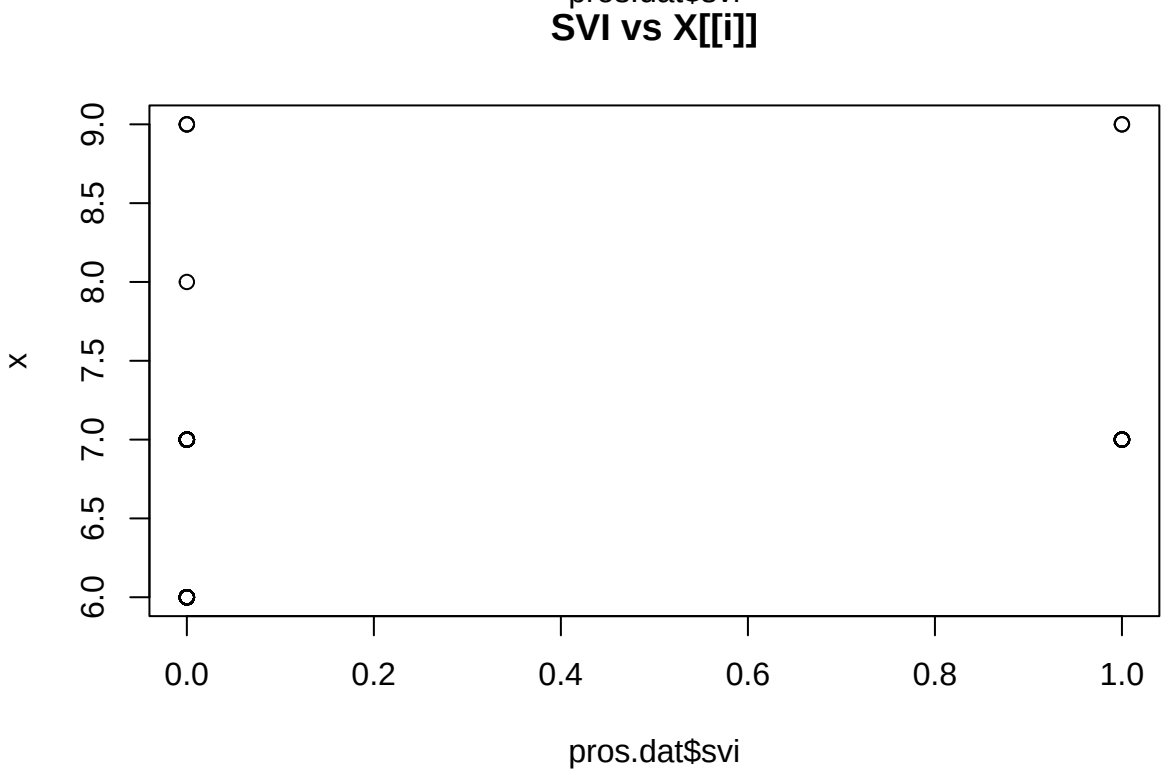

SVI vs X[[i]]
x
−1
0
1
2
3
4
0.0
0.2
0.4
0.6
0.8
1.0
pros.dat$svi

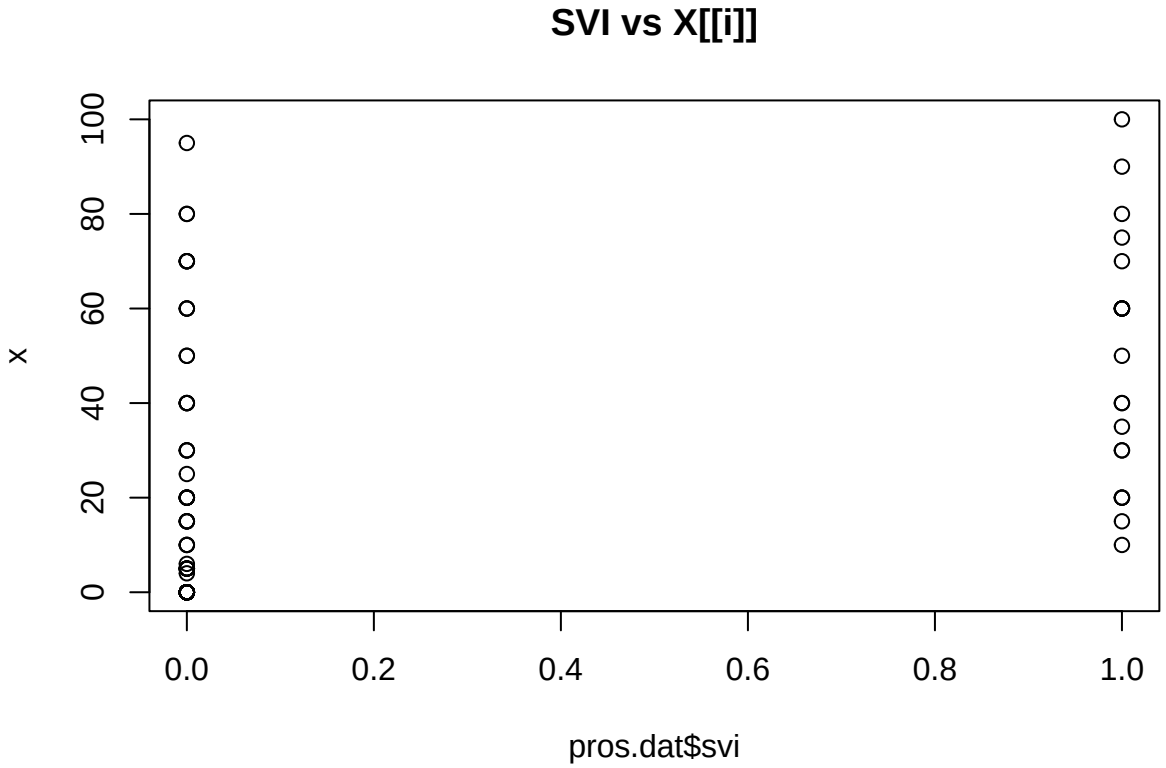

SVI vs X[[i]]
x
2.5
3.0
3.5
4.0
4.5
0.0
0.2
0.4
0.6
0.8
1.0
pros.dat$svi

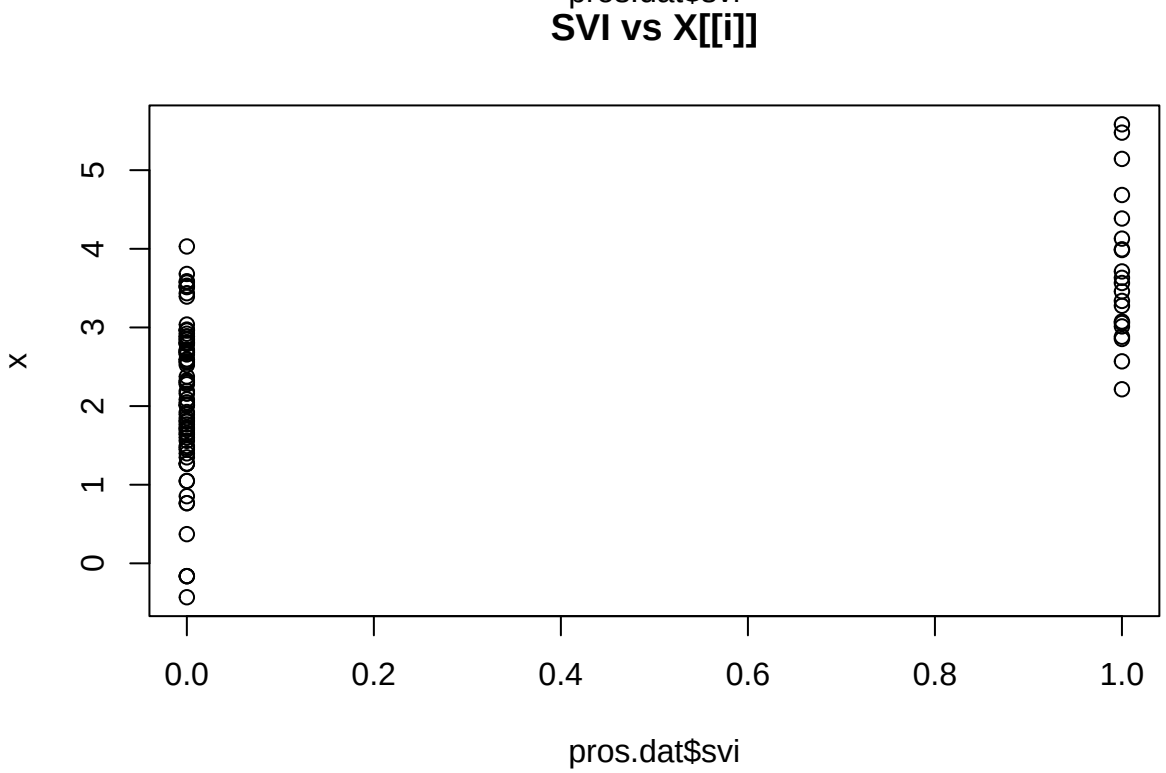

SVI vs X[[i]]
x
40
50
60
70
80
0.0
0.2
0.4
0.6
0.8
1.0
pros.dat$svi


SVI vs X[[i]]
x
−1
0
1
2
0.0
0.2
0.4
0.6
0.8
1.0
pros.dat$svi


SVI vs X[[i]]
x
−1
0
1
2
3
0.0
0.2
0.4
0.6
0.8
1.0
pros.dat$svi


SVI vs X[[i]]
x
6.0
6.5
7.0
7.5
8.0
8.5
9.0
0.0
0.2
0.4
0.6
0.8
1.0
pros.dat$svi






```
## $lcavol
## NULL
##
## $lweight
## NULL
##
```

```
## $age
## NULL
##
## $lbph
## NULL
##
## $lcp
## NULL
##
## $gleason
## NULL
##
## $pgg45
## NULL
##
## $lpsa
## NULL
```

- **2c.** Now, use `lapply()` to perform t-tests for each variable in the data set, between SVI and non-SVI groups. To be precise, you will perform a t-test for each variable excluding the SVI variable itself. For convenience, we've defined a function `t.test.by.ind()` below, which takes a numeric variable `x`, and then an indicator variable `ind` (of 0s and 1s) that defines the groups. Run this function on the columns of `pros.dat`, excluding the SVI column itself, and save the result as `tests`. What kind of data structure is `tests`? Print it to the console.

```
t.test.by.ind = function(x, ind) {
  stopifnot(all(ind %in% c(0, 1)))
  return(t.test(x[ind == 0], x[ind == 1]))
}
```

```
# YOUR CODE GOES HERE
tests <-
  lapply(pros.dat[ , !names(pros.dat) %in% "svi"],
         function(x) t.test.by.ind(x,
                                   pros.dat$svi))
cat("the data structure is: ",
    class(tests))
```

```
## the data structure is:  list
```

```
print(tests)
```

```
## $lcavol
##
##  Welch Two Sample t-test
##
## data:  x[ind == 0] and x[ind == 1]
## t = -8.0351, df = 51.172, p-value = 1.251e-10
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -1.917326 -1.150810
## sample estimates:
## mean of x mean of y
##  1.017892  2.551959
##
##
```

```
## $lweight
##
##  Welch Two Sample t-test
##
## data:  x[ind == 0] and x[ind == 1]
## t = -1.8266, df = 42.949, p-value = 0.07472
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.33833495  0.01674335
## sample estimates:
## mean of x mean of y
##  3.594131  3.754927
##
##
## $age
##
##  Welch Two Sample t-test
##
## data:  x[ind == 0] and x[ind == 1]
## t = -1.1069, df = 30.212, p-value = 0.2771
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -6.018547  1.786718
## sample estimates:
## mean of x mean of y
##  63.40789  65.52381
##
##
## $lbph
##
##  Welch Two Sample t-test
##
## data:  x[ind == 0] and x[ind == 1]
## t = 0.88281, df = 34.337, p-value = 0.3835
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.3914341  0.9930934
## sample estimates:
##  mean of x  mean of y
##  0.1654837 -0.1353460
##
##
## $lcp
##
##  Welch Two Sample t-test
##
## data:  x[ind == 0] and x[ind == 1]
## t = -8.8355, df = 31.754, p-value = 4.58e-10
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -2.797674 -1.749133
## sample estimates:
##  mean of x  mean of y
## -0.6715458  1.6018579
```

```
##
##
## $gleason
##
##  Welch Two Sample t-test
##
## data:  x[ind == 0] and x[ind == 1]
## t = -3.6194, df = 36.843, p-value = 0.0008816
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.8718223 -0.2459721
## sample estimates:
## mean of x mean of y
##  6.631579  7.190476
##
##
## $pgg45
##
##  Welch Two Sample t-test
##
## data:  x[ind == 0] and x[ind == 1]
## t = -4.9418, df = 31.288, p-value = 2.482e-05
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -44.04052 -18.31537
## sample estimates:
## mean of x mean of y
##  17.63158  48.80952
##
##
## $lpsa
##
##  Welch Two Sample t-test
##
## data:  x[ind == 0] and x[ind == 1]
## t = -6.8578, df = 33.027, p-value = 7.879e-08
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -2.047129 -1.110409
## sample estimates:
## mean of x mean of y
##  2.136592  3.715360
```

- **2d.** Using `lapply()` again, extract the p-values from the `tests` object you created in the last question, with just a single line of code. Hint: first, take a look at the first element of `tests`, what kind of object is it, and how is the p-value stored? Second, run the command `` `[[`(pros.dat, "lcavol") `` in your console—what does this do? Now use what you've learned to extract p-values from the `tests` object.

```
# YOUR CODE GOES HERE
 # extract p-value
p_values_list <-
  lapply(tests,
         function(x) x$p.value)
cat("The p-values are: ",
    unlist(p_values_list), "\n")
```

```
## The p-values are:  1.25104e-10 0.07472088 0.2770533 0.3834772 4.579752e-10 0.0008816293 2.482255e-05
```

Rio Olympics data set

Now we're going to examine data from the 2016 Summer Olympics in Rio de Janeiro, taken from https://github.com/flother/rio2016 (complete data on the 2020 Summer Olympics in Tokyo doesn't appear to be available yet). Below we read in the data and store it as `rio`.

```
rio = read.csv("rio.csv")
```

Q3. More practice with data frames and apply

- **3a.** What kind of object is `rio`? What are its dimensions and columns names of `rio`? What does each row represent? Is there any missing data?

```
# YOUR CODE GOES HERE
cat("the type of 'rio' is:",
    class(rio), "\n")
```

```
## the type of 'rio' is: data.frame
```

```
cat("the dimesion is :",
    dim(rio), "\n") # return the dimensions
```

```
## the dimesion is : 11538 12
```

```
cat("the column names are:",
    colnames(rio), "\n")# column names
```

```
## the column names are: id name nationality sex date_of_birth height weight sport gold silver bronze ir
```

- **3b.** Use `rio` to answer the following questions. How many athletes competed in the 2016 Summer Olympics? How many countries were represented? What were these countries, and how many athletes competed for each one? Which country brought the most athletes, and how many was this? Hint: for a factor variable `f`, you can use `table(f)` see how many elements in `f` are in each level of the factor.

```
# YOUR CODE GOES HERE
cat("the number of atheletes:",
    nrow(rio), "\n")
```

```
## the number of atheletes: 11538
```

```
cat("the number of contries:",
    nrow(rio$nationality), "\n")
```

```
## the number of contries:
```

```
atheletes_per_nation <- table(rio$nationality)
cat("Atheletes per country", "\n")
```

```
## Atheletes per country
```

```
cat(atheletes_per_nation)
```

```
## 3 6 68 5 26 9 223 32 7 4 431 71 56 30 7 11 9 108 6 8 2 11 3 124 12 12 485 34 3 50 5 6 6 321 5 10 2 4
```

```
max_atheletes_nation <-
  names(atheletes_per_nation)[which.max(atheletes_per_nation)]
max_atheletes_num <-
  max(atheletes_per_nation) # the max value
```

```r
cat("contries with the most atheletes:",
    max_atheletes_nation, "with",
    max_atheletes_num, "atheletes.\n")
```

```
## contries with the most atheletes: USA with 567 atheletes.
```

- **3c.** Please count the medal numbers of three types for each sport and identify which sports have different medal numbers of the three types.

```r
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
# the number of each kind of medals for each sport
medals_count <- rio %>%
  group_by(sport) %>%
  summarise(
    gold = sum(gold, na.rm = TRUE),
    silver = sum(silver, na.rm = TRUE),
    bronze = sum(bronze, na.rm = TRUE)
  )

print(medals_count)
```

```
## # A tibble: 28 x 4
##    sport       gold silver bronze
##    <chr>      <int>  <int>  <int>
##  1 aquatics     120    112    106
##  2 archery        8      8      8
##  3 athletics     66     64     62
##  4 badminton      8      8      8
##  5 basketball    24     24     24
##  6 boxing        13     13     26
##  7 canoe         27     27     28
##  8 cycling       27     28     29
##  9 equestrian    15     15     15
## 10 fencing       21     22     22
## # i 18 more rows
```

```r
# the sports that have different numbers in 3 kinds of medals
diff_medals_sports <- medals_count %>%
  filter(gold != silver | silver != bronze | gold != bronze)

print(diff_medals_sports)
```

```
## # A tibble: 13 x 4
##    sport         gold silver bronze
##    <chr>        <int>  <int>  <int>
##  1 aquatics       120    112    106
```

```
##  2 athletics       66     64     62
##  3 boxing          13     13     26
##  4 canoe           27     27     28
##  5 cycling         27     28     29
##  6 fencing         21     22     22
##  7 handball        29     30     29
##  8 hockey          34     32     33
##  9 judo            14     14     28
## 10 rugby sevens    25     24     25
## 11 taekwondo        8      8     16
## 12 volleyball      28     27     28
## 13 wrestling       18     18     36
```

- **3d.** Create a column called `total` which adds the number of gold, silver, and bronze medals for each athlete, and add this column to `rio`. Which athlete had the most number of medals and how many was this? Gold medals? Silver medals? In the case of ties, here, display all the relevant athletes.

```
# YOUR CODE GOES HERE
rio$total <- rio$gold + rio$silver + rio$bronze
max_medals <- max(rio$total, na.rm = TRUE)

athletes_with_max_medals <- rio %>%
  filter(total == max_medals)

cat("Athletes with the most medals:", "\n")
```

```
## Athletes with the most medals:
```

```
print(athletes_with_max_medals)
```

```
##          id           name nationality  sex date_of_birth height weight
## 1 491565031 Michael Phelps         USA male    1985-06-30   1.94     90
##      sport gold silver bronze
## 1 aquatics    5      1      0
##
## 1 The USA's Michael Phelps has claimed 22 Olympic medals from three editions, 18 of which were gold,
##   total
## 1     6
```

```
# show all information
cat("Gold medals:",
    max(athletes_with_max_medals$gold,
        na.rm = TRUE), "\n")
```

```
## Gold medals: 5
```

```
cat("Silver medals:",
    max(athletes_with_max_medals$silver,
        na.rm = TRUE), "\n")
```

```
## Silver medals: 1
```

```
cat("Bronze medals:",
    max(athletes_with_max_medals$bronze,
        na.rm = TRUE), "\n")
```

```
## Bronze medals: 0
```

- **3e.** Using `tapply()`, calculate the total medal count for each country. Save the result as `total.by.nat`,

and print it to the console. Which country had the most number of medals, and how many was this? How many countries had zero medals?

```
# YOUR CODE GOES HERE
total.by.nat <-
  tapply(rio$gold + rio$silver + rio$bronze,
         rio$nationality,
         sum,
         na.rm = TRUE)

print(total.by.nat)
```

```
## AFG ALB ALG AND ANG ANT ARG ARM ARU ASA AUS AUT AZE BAH BAN BAR BDI BEL BEN BER
##   0   0   2   0   0   0  22   4   0   0  82   2  18   6   0   0   1  21   0   0
## BHU BIH BIZ BLR BOL BOT BRA BRN BRU BUL BUR CAF CAM CAN CAY CGO CHA CHI CHN CIV
##   0   0   0  12   0   0  51   2   0   7   0   0   0  69   0   0   0   0 113   2
## CMR COD COK COL COM CPV CRC CRO CUB CYP CZE DEN DJI DMA DOM ECU EGY ERI ESA ESP
##   0   0   0   8   0   0   0  24  11   0  15  41   0   0   1   0   3   0   0  45
## EST ETH FIJ FIN FRA FSM GAB GAM GBR GBS GEO GEQ GER GHA GRE GRN GUA GUI GUM GUY
##   4   8  13   1  95   0   0   0 145   0   7   0 160   0   7   1   0   0   0   0
## HAI HKG HON HUN INA IND IOA IRI IRL IRQ ISL ISR ISV ITA IVB JAM JOR JPN KAZ KEN
##   0   0   0  22   4   2   2   8   3   0   0   2   0  72   0  30   1  65  17  13
## KGZ KIR KOR KOS KSA LAO LAT LBA LBR LCA LES LIB LIE LTU LUX MAD MAR MAS MAW MDA
##   0   0  26   1   0   0   0   0   0   0   0   0   0   7   0   0   1   8   0   1
## MDV MEX MGL MHL MKD MLI MLT MNE MON MOZ MRI MTN MYA NAM NCA NED NEP NGR NIG NOR
##   0   5   2   0   0   0   0   0   0   0   0   0   0   0   0  47   0  18   1  19
## NRU NZL OMA PAK PAN PAR PER PHI PLE PLW PNG POL POR PRK PUR QAT ROT ROU RSA RUS
##   0  36   0   0   0   0   0   1   0   0   0  16   1   7   1   1   0  17  23 115
## RWA SAM SEN SEY SIN SKN SLE SLO SMR SOL SOM SRB SRI SSD STP SUD SUI SUR SVK SWE
##   0   0   0   0   1   0   0   4   0   0   0  53   0   0   0   0  11   0   8  28
## SWZ SYR TAN TGA THA TJK TKM TLS TOG TPE TTO TUN TUR TUV UAE UGA UKR URU USA UZB
##   0   0   0   0   6   1   0   0   0   5   1   3   8   0   1   0  15   0 264  13
## VAN VEN VIE VIN YEM ZAM ZIM
##   0   3   2   0   0   0   0
```

```
# country with the max number of medals
max_medals_country <-
  names(total.by.nat)[which.max(total.by.nat)]
max_medals_num <-
  max(total.by.nat)

cat("The country with the most medals:",
    max_medals_country, "with",
    max_medals_num, "medals.\n")
```

```
## The country with the most medals: USA with 264 medals.
```

```
# countries with 0 medals
countries_with_zero_medals <-
  sum(total.by.nat == 0)

cat("Number of countries with zero medals:",
    countries_with_zero_medals, "\n")
```

```
## Number of countries with zero medals: 120
```

- **3f.** Among the countries that had zero gold medals, which had the most athletes, and how many

athletes was this?

```
# YOUR CODE GOES HERE
# countries with 0 gold medals
no_gold_countries <- rio %>%
  group_by(nationality) %>%
  summarise(total_gold = sum(gold, na.rm = TRUE)) %>%
  filter(total_gold == 0)

athletes_per_no_gold <-
  tapply(rio$total,
         rio$nationality,
         sum,
         na.rm = TRUE)
athletes_in_no_gold_countries <-
  athletes_per_no_gold[names(athletes_per_no_gold)
                       %in%
                         no_gold_countries$nationality]

max_athletes_country <-
  names(athletes_in_no_gold_countries)[which.max
                                       (athletes_in_no_gold_countries)]
max_athletes_count <-
  max(athletes_in_no_gold_countries)

cat("The country with the most athletes but no gold medals is:",
    max_athletes_country,
    "with",
    max_athletes_count,
    "athletes.\n")
```

```
## The country with the most athletes but no gold medals is: NOR with 19 athletes.
```

Q4. Young and old folks

- **4a.** The variable `date_of_birth` contains strings of the date of birth of each athlete. Use the `substr()` function to extract the year of birth for each athlete, and then create a new numeric variable called `age`, equal to 2016 - (the year of birth). (Here we are ignoring days and months for simplicity.)

```
# YOUR CODE GOES HERE
```

- **4b.** Answer the same questions as in the last part, but now only among athletes who won a gold medal.

```
# YOUR CODE GOES HERE
```

- **4c.** Using a single call to `tapply()`, answer: how old are the youngest and oldest athletes, for each sport?

```
# YOUR CODE GOES HERE
```

- **4d.** You should see that your output from `tapply()` in the last part is a list, which is not particularly convenient. Convert this list into a matrix that has one row for each sport, and two columns that display the ages of the youngest and oldest athletes in that sport. The first 3 rows should look like this:

```
                Youngest Oldest
athletics             14     41
archery               17     44
athletics             16     47
```

You'll notice that we set the row names according to the sports, and we also set appropriate column names. Hint: `unlist()` will unravel all the values in a list; and `matrix()`, as you've seen before, can be used to create a matrix from a vector of values. After you've converted the results to a matrix, print it to the console (and make sure its first 3 rows match those displayed above).

```
# YOUR CODE GOES HERE
```

- **4e.** Determine the *names* of the youngest and oldest athletes in each sport, along with their ages (so your result should have 4 columns), without using any explicit iteration. In the case of ties, just return one relevant athlete name. (For this part, you can use another package, such as `plyr` or `dplyr` if you want to.)

```
# YOUR CODE GOES HERE
```

Q5. Sport by sport

- **5a.** Create a new data frame called `sports`, which we'll populate with information about each sporting event at the Summer Olympics. Initially, define `sports` to contain a single variable called `sport` which contains the names of the sporting events in alphabetical order. Then, add a column called `n_participants` which contains the number of participants in each sport. Use one of the apply functions to determine the number of gold medals given out for each sport, and add this as a column called `n_gold`. Using your newly created `sports` data frame, calculate the ratio of the number of gold medals to participants for each sport. Which sport has the highest ratio? Which has the lowest?

```
# YOUR CODE GOES HERE
```

- **5b.** Use one of the apply functions to compute the average weight of the participants in each sport, and add this as a column to `sports` called `ave_weight`. Important: there are missing weights in the data set coded as `NA`, but your column `ave_weight` should ignore these, i.e., it should be itself free of `NA` values. You will have to pass an additional argument to your apply call in order to achieve this. Hint: look at the help file for the `mean()` function; what argument can you set to ignore `NA` values? Once computed, display the average weights along with corresponding sport names, in decreasing order of average weight.

```
# YOUR CODE GOES HERE
```

- **5c.** As in the last part, compute the average weight of athletes in each sport, but now separately for men and women. You should therefore add two new columns, called `ave_weight_men` and `ave_weight_women`, to `sports`. Once computed, display the average weights along with corresponding sports, for men and women, each list sorted in decreasing order of average weight. Are the orderings roughly similar?

```
# YOUR CODE GOES HERE
```

- **5d.** Use one of the apply functions to compute the proportion of women among participating athletes in each sport. Use these proportions to recompute the average weight (over all athletes in each sport) from the `ave_weight_men` and `average_weight_women` columns, and define a new column `ave_weight2` accordingly. Does `ave_weight2` differ from `ave_weight`? It should. Explain why. Then show how to recompute the average weight from `ave_weight_men` and `average_weight_women` in a way that exactly recreates `average_weight`.

```
# YOUR CODE GOES HERE
```

```
library(purrr)
library(dplyr)
library(repurrrsive)
```

Game of Thrones data set

We also install the `repurrrsive` package which has the Game of Thrones data set that we'll use for the first couple of questions. Since this may be the first time installing packages for some of you, we'll show you how. If you already have these packages installed, then you can of course skip this part. Note: *do not remove* ***`eval=FALSE`*** *from the above code chunk*, just run the lines below in your console. Otherwise, you will repeatedly install the package once you compile this R markdown file.

```
install.packages("repurrrsive")
```

Then, we load the required packages

```
library(purrr)
library(dplyr)
library(repurrrsive)
```

Below we inspect a data set on the 30 characters from Game of Thrones from the `repurrrsive` package. It's stored in a list called `got_chars`, which is automatically loaded into your R session when you load the `repurrrsive` package.

```
class(got_chars)
```

```
## [1] "list"
```

```
length(got_chars)
```

```
## [1] 30
```

```
names(got_chars[[1]])
```

```
##  [1] "url"         "id"          "name"        "gender"      "culture"
##  [6] "born"        "died"        "alive"       "titles"      "aliases"
## [11] "father"      "mother"      "spouse"      "allegiances" "books"
## [16] "povBooks"    "tvSeries"    "playedBy"
```

```
got_chars[[1]]$name
```

```
## [1] "Theon Greyjoy"
```

```
got_chars[[1]]$aliases
```

```
## [1] "Prince of Fools" "Theon Turncloak" "Reek"            "Theon Kinslayer"
```

Q6. Warming up with map

- Using the map functions from the `purrr` package, extract the names of the characters in `got_chars` so that you produce a character vector of length 30. Do this four different ways: (i) using `map()`, defining a custom function on-the-fly, and casting the resulting list into an appropriate data structure; (ii) using one of the `map_***()` functions, but still defining a custom function on-the-fly; (iii) using one of the `map_***()` functions, and using one of `` `[`() `` or `` `[[`() `` functions, as well as an additional argument; (iv) using one of the `map_***()` functions, and passing a string instead of a function (relying on its ability to define an appropriate extractor accordingly).

  Store each of the results in a different vector and check that they are all identical.

```
# YOUR CODE GOES HERE
```

Q7. Cultural studies

- **7a.** Using `map_dfr()`, create a data frame of dimension 30 x 5, whose columns represent, for each Game of Thrones character, their name, birth date, death date, gender, and culture. Store it as `got_df` and print the last 5 rows to the console.

```
# YOUR CODE GOES HERE
```

- **7b.** Using `got_df`, show that you can compute whether each character is alive or not, and compare this to what is stored in `got_chars`, demonstrating that the two ways of checking whether each character is alive lead to equal results.

```
# YOUR CODE GOES HERE
```

- **7c.** Using `filter()`, print the subset of the rows of `got_df` that correspond to Ironborn characters. Then print the subset that correspond to female Northmen.

```
# YOUR CODE GOES HERE
```

- **7d.** Create a matrix of dimension (number of cultures) x 2 that counts how many women and men there are in each culture appearing in `got_df`. Print the results to the console. Hint: what happens if you pass `table()` two arguments?

```
# YOUR CODE GOES HERE
```

- **7e.** Using `group_by()` and `summarize()` on `got_df`, compute how many characters in each culture have died. Which culture—aside from the unknown category represented by `""`—has the most deaths?

```
# YOUR CODE GOES HERE
```

Rio Olympics data set

We continue to analyze the `rio` data set using the `dplyr` package.

```
rio = read.csv("rio.csv")
```

Q8. Practice with grouping and summarizing

- **8a.** Using `group_by()` and `summarize()`, compute how many athletes competed for each country in the `rio` data frame? Print the results for the first 10 countries to the console. Building off your here answer, use an additional call to `filter()` to compute which country had more than 400 athletes and how many those was. Hint: consider using `n()` from the `dplyr` package for the first part here.

```
# YOUR CODE GOES HERE
```

- **8b.** Using `group_by()`, `summarize()`, and `filter()`, compute which country had more than 100 medals and many those were.

```
# YOUR CODE GOES HERE
```

- **8c.** Using `group_by()`, `summarize()`, and `filter()`, compute which country—among those with zero total medals—had the most number of athletes. Hint: you will need to modify your `summarize()` command to compute the number of athletes; and you might need two calls to `filter()`.

```
# YOUR CODE GOES HERE
```

- **8d.** Using—yes, you guessed it—`group_by()`, `summarize()`, and `filter()`, compute the average weight of athletes in each sport, separately for men and women, and report the two sport with the highest average weights (one for each of men and women). Hint: `group_by()` can accept more than one grouping variable. Also, consider using `na.rm=TRUE` as an additional argument to certain arithmetic summary functions so that they will not be thrown off by `NA` or `NaN` values.

```
# YOUR CODE GOES HERE
```

Fastest 100m sprint times

Below, we read two data sets of the 1000 fastest times ever recorded for the 100m sprint, in men's and women's track. We scraped this data from http://www.alltime-athletics.com/m_100ok.htm and http://www.alltime-athletics.com/w_100ok.htm, in early September 2021.

```
sprint.m.df = read.table(
  file="sprint.m.txt",
  sep="\t", quote="", header=TRUE)
sprint.w.df = read.table(
  file="sprint.w.txt",
  sep="\t", quote="", header=TRUE)
```

Q9. More practice with data frame computations

- **9a.** Confirm that both `sprint.m.df` and `sprint.w.df` are data frames. Delete the `Rank` column from each data frame, then display the first and last 5 rows of each.

```
# YOUR CODE GOES HERE
```

- **9b.** Recompute the ranks for the men's data set from the `Time` column and add them back as a `Rank` column to `sprint.m.df`. Do the same for the women's data set. After adding back the rank columns, print out the first 10 rows of each data frame, but only the `Time`, `Name`, `Date`, and `Rank` columns. Hint: consider using `rank()`.

```
# YOUR CODE GOES HERE
```

- **9c.** Using base R functions, compute, for each country, the number of sprint times from this country that appear in the men's data set. Call the result `sprint.m.counts`. Do the same for the women's data set, and call the result `sprint.w.counts`. What are the 5 most represented countries, for the men, and for the women?

```
# YOUR CODE GOES HERE
```

- **9d.** Repeat the same calculations as in last part but using `dplyr` functions, and print out again the 5 most represented countries for men and women. (No need to save new variables.) Hint: consider using `arrange()` from the `dplyr` library.

```
# YOUR CODE GOES HERE
```

- **9e.** Are there any countries that are represented by women but not by men, and if so, what are they? Vice versa, represented by men and not women? Hint: consider using the `%in%` operator.

```
# YOUR CODE GOES HERE)
```

Q10. Practice with grouping

- **10a.** Using `dplyr` functions, compute, for each country, the fastest time among athletes who come from that country. Do this for each of the men's and women's data sets, and display the first 10 rows of the result.

```
# YOUR CODE GOES HERE
```

- **10b.** With the most minor modification to your code possible, do the same computations as in the last part, but now display the first 10 results ordered by increasing time. Hint: recall `arrange()`.

```
# YOUR CODE GOES HERE
```

- **10c.** Rewrite your solution in the last part using base R. Hint: `tapply()` gives probably the easiest route here. Note: your code here shouldn't be too much more complicated than your code in the last part.

```
# YOUR CODE GOES HERE
```

- **10d.** Using `dplyr` functions, compute, for each country, the quadruple: name, city, country, and time, corresponding to the athlete with the fastest time among athletes from that country. Do this for each of the men's and women's data sets, and display the first 10 rows of the result, ordered by increasing time. If there are ties, then show all the results that correspond to the fastest time. Hint: consider using `select()` from the `dplyr` library.

```
# YOUR CODE GOES HERE
```

- **10e.** Rewrite your solution in the last part using base R. Hint: there are various routes to go; one strategy is to use `split()`, followed by `lapply()` with a custom function call, and then `rbind()` to get things in a data frame form. Note: your code here will probably be more complicated, or at least less intuitive, than your code in the last part.

```
# YOUR CODE GOES HERE
```