MiCM Workshop Series

R - Beyond the Basics

Efficient Coding

- Yi Lian

- March 6, 2020

Link to workshop material https://github.com/ly129/MiCM2020 (https://github.com/ly129/MiCM2020)

Outline

1. **An overview of efficiency**
    - General rules
    - R-specific rules
    - R object types (if necessary)
    - Record runtime of your code
2. **Efficient coding**
    - Powerful functions in R

    ```
    - aggregate(), by(), apply() family
    - ifelse(), cut() and split()
    ```

    - Write our own functions in R

    ```
    - function()
    ```

    - Examples
        - Categorization, conditional operations, etc..
3. **Exercises**

Important note! There are MANY advanced and powerful packages that do different things. There are too many and they are too diverse to be covered in this workshop.

Here is a list of some awesome packages. https://awesome-r.com/ (https://awesome-r.com/)

1.1 General rules

1.2 R-specific rules

1.3 R data types and structures

1.3.1 R data types

```
- numeric
    - integer
    - double precision (default)
- logical
- character
- factor
- ...
```

In [1]:

```
# double
class(5); is.double(5)
```

'numeric'

TRUE

In [2]:

```
# integer
class(5L); is.double(5L)
```

'integer'

FALSE

In [3]:

```
# How precise is double precision?
options(digits = 22) # show more digits in output
print(1/3)
options(digits = 7) # back to the default
```

```
[1] 0.3333333333333333148296
```

In [4]:

```
object.size(rep(5, 10))
object.size(rep(5L, 10))
```

```
176 bytes
```

```
96 bytes
```

In [5]:

```
# logical
class(TRUE); class(F)
```

'logical'

'logical'

In [6]:

```
# character
class("TRUE")
```

'character'

In [7]:

```
# Not important for this workshop
fac <- as.factor(c(1, 5, 11, 3))
fac
```

1 5 11 3

▶ **Levels**:

In [8]:
```
class(fac)
```

'factor'

In [9]:
```
# R has an algorithm to decide the order of the levels
fac.ch <- as.factor(c("B", "a", "1", "ab", "b", "A"))
fac.ch
```

B a 1 ab b A

▶ **Levels**:

1.3.2 R data structures

```
- Scalar *
- Vector
- Matrix
- Array
- List
- Data frame
- ...
```

In [10]:
```
# Scalar - a vector of length 1
myscalar <- 5
myscalar
```

5

In [11]:
```
class(myscalar)
```

'numeric'

In [12]:
```
# Vector
myvector <- c(1, 1, 2, 3, 5, 8)
myvector
```

1 1 2 3 5 8

In [13]:
```
class(myvector)
```

'numeric'

In [14]:
```
# Matrix - a 2d array
mymatrix <- matrix(c(1, 1, 2, 3, 5, 8), nrow = 2, byrow = FALSE)
mymatrix
```

A matrix:
2 × 3 of
type dbl

1	2	5
1	3	8

In [15]:
```
class(mymatrix)
```

'matrix'

In [16]:
```
str(mymatrix)
```

```
 num [1:2, 1:3] 1 1 2 3 5 8
```

In [17]:
```
# Array - not important for this workshop
myarray <- array(c(1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144), dim = c(2, 2, 3))
print(myarray) # print() is not needed if run in R or Rstudio.
```

```
, , 1

     [,1] [,2]
[1,]    1    2
[2,]    1    3

, , 2

     [,1] [,2]
[1,]    5   13
[2,]    8   21

, , 3

     [,1] [,2]
[1,]   34   89
[2,]   55  144
```

In [18]:
```
class(myarray)
```

'array'

In [19]:
```
# List - very important for the workshop
mylist <- list(Title = "R Beyond the Basics",
               Duration = c(2, 2),
               sections = as.factor(c(1, 2, 3, 4)),
               Date = as.Date("2020-03-06"),
               Lunch_provided = FALSE,
               Feedbacks = c("Amazing!", "Great workshop!", "Yi is the best!", "Wow!")
)
print(mylist) # No need for print if running in R or Rstudio
```

```
$Title
[1] "R Beyond the Basics"

$Duration
[1] 2 2

$sections
[1] 1 2 3 4
Levels: 1 2 3 4

$Date
[1] "2020-03-06"

$Lunch_provided
[1] FALSE

$Feedbacks
[1] "Amazing!"        "Great workshop!" "Yi is the best!" "Wow!"
```

In [20]:
```
class(mylist)
```

'list'

In [21]:
```
# Access data stored in lists
mylist$Title
```

'R Beyond the Basics'

In [22]:
```
# or
mylist[[6]]
```

'Amazing!' 'Great workshop!' 'Yi is the best!' 'Wow!'

In [23]:
```
# Further
mylist$Duration[1]
mylist[[6]][2]
```

2

'Great workshop!'

In [24]:
```
# Elements in lists can have different data types
lapply(mylist, class) # We will talk about lapply() later
```

$Title
'character'
$Duration
'numeric'
$sections
'factor'
$Date
'Date'
$Lunch_provided
'logical'
$Feedbacks
'character'

In [25]:
```
# Elements in list can have different lengths
lapply(mylist, length)
```

$Title
1
$Duration
2
$sections
4
$Date
1
$Lunch_provided
1
$Feedbacks
4

In [26]:
```
# Data frames - most commonly used for analyses
head(mtcars)
```

A data.frame: 6 × 11

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
Mazda RX4	21.0	6	160	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160	110	3.90	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108	93	3.85	2.320	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258	110	3.08	3.215	19.44	1	0	3	1
Hornet Sportabout	18.7	8	360	175	3.15	3.440	17.02	0	0	3	2
Valiant	18.1	6	225	105	2.76	3.460	20.22	1	0	3	1

In [27]:
```
# Access a column (variable) in data frames
mtcars$mpg
```

21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8 16.4 17.3 15.2 10.4 10.4 14.7 32.4 30.4 33.9 21.5 15.5 15.2 13.3 19.2 27.3 26 30.4 15.8 19.7 15 21.4

1.4 Time your program in R

Illustrations of R rules for efficiency.

```
- proc.time(), system.time()
- microbenchmark
```

1.4.1 Vectorized operation vs. loop

Example Calculate the square root of 1 to 1,000,000 using vectorized operation vs. using a for loop.

In [28]:
```
# Vectorized operation
# system.time(operation)  returns the time needed to run the 'operation'
t <- system.time( x1 <- sqrt(1:1000000) )
head(x1)
```

1 1.4142135623731 1.73205080756888 2 2.23606797749979 2.44948974278318

In [29]:
```
# For loop with memory pre-allocation
x2 <- rep(NA, 1000000)
t0 <- proc.time()
for (i in 1:1000000) {
    x2[i] <- sqrt(i)
}
t1 <- proc.time()

identical(x1, x2) # Check whether results are the same
```

TRUE

In [30]:
```
# For loop without memory pre-allocation
x3 <- NULL
t2 <- proc.time()
for (i in 1:1000000) {
    x3[i] <- sqrt(i)
}
t3 <- proc.time()
```

In [31]:
```
# As we can see, R is not very fast with loops.
t; t1 - t0; t3 - t2
# ?proc.time
```

```
   user  system elapsed 
  0.006   0.005   0.011 

   user  system elapsed 
  0.066   0.004   0.071 

   user  system elapsed 
  0.289   0.065   0.355 
```

Take-home message

- Use vectorized operations rather than loops for speed in R.
- Loops are more intuitive though.
- Balance between
  - speed
  - your need for speed
  - your level of comfortableness with linear algebra
  - your level of laziness
  - your typing speed
  - ...
- Based on what you are doing
  - dealing with big dataset and expensive calculations?
  - running the code only once or potentially many many times?

1.4.2 Use established functions

Example Calculate the square root using sqrt( ) vs. our own implementation.

In [32]:
```
# microbenchmark runs the code multiple times and take a summary
# Use well-developped R function
library(microbenchmark)
result <- microbenchmark(sqrt(500),
                         500^0.5,
                         unit = "ns", times = 1000
                        )
summary(result)
# Result in nanoseconds
```

A data.frame: 2 × 8

expr	min	lq	mean	median	uq	max	neval
<fct>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
sqrt(500)	81	91	126.869	98	103	17784	1000
500^0.5	155	164	196.509	171	177	8721	1000

In summary, keep the rules in mind, know what you want to do, test your program, time your program.

2. Efficient coding

R has many powerful and useful functions that we can use to achieve efficient coding and computing.

2.1 Powerful functions in R

Let's play with some data.

In [33]:
```
data <- read.csv("https://raw.githubusercontent.com/ly129/MiCM2020/master/sample.csv", header =
TRUE)
head(data, 8)
```

A data.frame: 8 × 8

X	Sex	Wr.Hnd	NW.Hnd	Pulse	Smoke	Height	Age
<int>	<fct>	<dbl>	<dbl>	<int>	<fct>	<dbl>	<dbl>
1	Male	21.4	21.0	63	Never	180.00	19.000
2	Male	19.5	19.4	79	Never	165.00	18.083
3	Female	16.3	16.2	44	Regul	152.40	23.500
4	Female	15.9	16.5	99	Never	167.64	17.333
5	Male	19.3	19.4	55	Never	180.34	19.833
6	Male	18.5	18.5	48	Never	167.00	22.333
7	Female	17.5	17.0	85	Heavy	163.00	17.667
8	Male	19.8	20.0	NA	Never	180.00	17.417

In [34]:
```
summary(data)
```

```
       X              Sex          Wr.Hnd          NW.Hnd          Pulse
 Min.   :  1.00   Female:47   Min.   :13.00   Min.   :12.50   Min.   : 40.00
 1st Qu.: 25.75   Male  :53   1st Qu.:17.50   1st Qu.:17.45   1st Qu.: 50.25
 Median : 50.50               Median :18.50   Median :18.50   Median : 71.50
 Mean   : 50.50               Mean   :18.43   Mean   :18.39   Mean   : 69.90
 3rd Qu.: 75.25               3rd Qu.:19.50   3rd Qu.:19.52   3rd Qu.: 84.75
 Max.   :100.00               Max.   :23.20   Max.   :23.30   Max.   :104.00
                                                              NA's   :6
   Smoke        Height           Age
 Heavy: 6   Min.   :152.0   Min.   :16.92
 Never:79   1st Qu.:166.4   1st Qu.:17.58
 Occas: 5   Median :170.2   Median :18.46
 Regul:10   Mean   :171.8   Mean   :20.97
            3rd Qu.:179.1   3rd Qu.:20.21
            Max.   :200.0   Max.   :73.00
            NA's   :13
```

a1. Calculate the mean writing hand span of all individuals

```
mean(x, trim = 0, na.rm = FALSE, ...)
```

In [35]:
```
mean(data$Wr.Hnd)
```

18.43

a2. Calculate the mean height of all individuals, exclude the missing values

In [36]:
```
mean(data$Height)
```

<NA>

In [37]:
```
mean(data$Height, na.rm = T)
```

171.784597701149

a3. Calculate the mean of all continuous variables

```
apply(X, MARGIN, FUN, ...)
```

In [38]:
```
# Choose the continuous variables
names(data)
cts <- c("Wr.Hnd", "NW.Hnd", "Pulse", "Height", "Age")
cts.data <- data[, cts]
head(cts.data)
```

'X' 'Sex' 'Wr.Hnd' 'NW.Hnd' 'Pulse' 'Smoke' 'Height' 'Age'

A data.frame: 6 × 5

Wr.Hnd	NW.Hnd	Pulse	Height	Age
<dbl>	<dbl>	<int>	<dbl>	<dbl>
21.4	21.0	63	180.00	19.000
19.5	19.4	79	165.00	18.083
16.3	16.2	44	152.40	23.500
15.9	16.5	99	167.64	17.333
19.3	19.4	55	180.34	19.833
18.5	18.5	48	167.00	22.333

In [39]:
```
# Calculate the mean
apply(X = cts.data, MARGIN = 2, FUN = mean)
```

Wr.Hnd	18.43
NW.Hnd	18.391
Pulse	<NA>
Height	<NA>
Age	20.96503

In [40]:
```
apply(cts.data, MARGIN = 2, FUN = mean, na.rm = TRUE)
```

Wr.Hnd	18.43
NW.Hnd	18.391
Pulse	69.9042553191489
Height	171.784597701149
Age	20.96503

b1. Calculate the count/proportion of females and males

```
table(...,
  exclude = if (useNA == "no") c(NA, NaN),
  useNA = c("no", "ifany", "always"),
  dnn = list.names(...), deparse.level = 1)

prop.table()
```

In [41]:
```
sex.tab <- table(data$Sex)
sex.tab
```

```
Female   Male
    47     53
```

In [42]:
```
prop.table(sex.tab)
```

```
Female   Male
  0.47   0.53
```

b2. Calculate the count in each Smoke group

In [43]:
```
smoke.tab <- table(data$Smoke)
smoke.tab
```

```
Heavy Never Occas Regul
    6    79     5    10
```

b3. Calculate the count of males and females in each Smoke group

In [44]:
```
table(data$Sex, data$Smoke)
```

```
         Heavy Never Occas Regul
  Female     3    40     3     1
  Male       3    39     2     9
```

In [45]:
```
# I prefer this...
table(data[, c("Sex", "Smoke")])
```

```
        Smoke
Sex      Heavy Never Occas Regul
  Female     3    40     3     1
  Male       3    39     2     9
```

c1. Calculate the standard deviation of writing hand span of females

```
aggregate()
tapply()
by()
```

In [46]:
```
# aggregate() syntax 1
aggregate(x = data$Wr.Hnd, by = list(data$Sex), FUN = sd)
```

A data.frame: 2 × 2

Group.1	x
<fct>	<dbl>
Female	1.519908
Male	1.712066

In [47]:
```
# aggregate() syntax 2
aggregate(Wr.Hnd~Sex, data = data, FUN = sd)
```

A data.frame: 2 × 2

Sex	Wr.Hnd
<fct>	<dbl>
Female	1.519908
Male	1.712066

In [48]:
```
# by()
by(data = data$Wr.Hnd, INDICES = list(data$Sex), FUN = sd)
```

```
: Female
[1] 1.519908
------------------------------------------------------------
: Male
[1] 1.712066
```

In [49]:
```
# tapply()
tapply(X = data$Wr.Hnd, INDEX = list(data$Sex), FUN = sd)
```

Female	1.51990797715501
Male	1.71206552443005

In [50]:
```
# Return a list using tapply()
tapply(X = data$Wr.Hnd,
       INDEX = list(data$Sex),
       FUN = sd,
       simplify = F)
```

$Female
1.51990797715501
$Male
1.71206552443005

aggregate( ), by( ) and tapply( ) are all connected. They give different types of output.

c2. Calculate the standard deviation of writing hand span of all different Sex-Smoke groups

In [51]:
```
# Syntax 1
aggregate(x = data$Wr.Hnd,
          by = list(data$Sex, data$Smoke),
          FUN = sd)
```

A data.frame: 8 × 3

Group.1	Group.2	x
<fct>	<fct>	<dbl>
Female	Heavy	0.2309401
Male	Heavy	4.8569538
Female	Never	1.5762663
Male	Never	1.3857770
Female	Occas	1.9000000
Male	Occas	2.2627417
Female	Regul	NA
Male	Regul	1.6537835

In [52]:
```
# Syntax 2
aggregate(Wr.Hnd~Sex+Smoke, data = data, FUN = sd)
```

A data.frame: 8 × 3

Sex	Smoke	Wr.Hnd
<fct>	<fct>	<dbl>
Female	Heavy	0.2309401
Male	Heavy	4.8569538
Female	Never	1.5762663
Male	Never	1.3857770
Female	Occas	1.9000000
Male	Occas	2.2627417
Female	Regul	NA
Male	Regul	1.6537835

c3. Calculate the standard deviation of writing hand and non-writing hand span of all Sex-Smoke groups

```
cbind()
```

In [53]:
```
# Syntax 1
aggregate(x = cbind(wh = data$Wr.Hnd, nwh = data$NW.Hnd),
          by = list(sex = data$Sex, smoke = data$Smoke),
          FUN = sd)
```

A data.frame: 8 × 4

sex	smoke	wh	nwh
<fct>	<fct>	<dbl>	<dbl>
Female	Heavy	0.2309401	0.2886751
Male	Heavy	4.8569538	3.9828800
Female	Never	1.5762663	1.6625899
Male	Never	1.3857770	1.3760875
Female	Occas	1.9000000	1.3796135
Male	Occas	2.2627417	1.0606602
Female	Regul	NA	NA
Male	Regul	1.6537835	1.3991069

In [54]:
```
# Syntax 2
aggregate(cbind(Wr.Hnd, NW.Hnd)~Sex+Smoke, data = data, FUN = sd)
```

A data.frame: 8 × 4

Sex	Smoke	Wr.Hnd	NW.Hnd
<fct>	<fct>	<dbl>	<dbl>
Female	Heavy	0.2309401	0.2886751
Male	Heavy	4.8569538	3.9828800
Female	Never	1.5762663	1.6625899
Male	Never	1.3857770	1.3760875
Female	Occas	1.9000000	1.3796135
Male	Occas	2.2627417	1.0606602
Female	Regul	NA	NA
Male	Regul	1.6537835	1.3991069

Let's try to figure out what aggregate( ) is doing

```
print()
```

In [55]:
```
aggregate(Wr.Hnd~Sex+Smoke, data = data, FUN = print)
```

```
[1] 17.5 17.5 17.1
[1] 14.0 23.2 21.3
 [1] 15.9 13.0 18.5 17.5 18.6 16.0 13.0 19.6 17.5 19.5 19.5 16.4 17.2 19.4 17.0
[16] 18.0 16.9 16.5 17.0 17.6 16.5 18.8 17.7 15.5 18.0 17.6 19.5 19.0 17.5 19.0
[31] 18.5 15.0 16.0 18.5 17.5 18.0 19.0 17.5 17.6 18.7
 [1] 21.4 19.5 19.3 18.5 19.8 22.0 20.0 18.0 21.0 18.9 18.1 16.0 18.8 18.5 17.8
[16] 21.0 18.5 19.1 21.0 19.0 21.5 20.8 18.9 18.5 19.2 17.7 17.5 18.0 18.5 19.2
[31] 21.5 17.5 19.5 17.0 18.2 18.0 19.5 19.5 20.5
[1] 19.1 15.4 16.5
[1] 22.2 19.0
[1] 16.3
[1] 18.5 19.5 19.7 18.0 17.0 22.5 20.5 20.0 21.0
```

A data.frame: 8 × 3

Sex	Smoke	Wr.Hnd
<fct>	<fct>	<list>
Female	Heavy	17.5, 17.5, 17.1
Male	Heavy	14.0, 23.2, 21.3
Female	Never	15.9, 13.0, 18.5, 17.5, 18.6, 16.0, 13.0, 19.6, 17.5, 19.5, 19.5, 16.4, 17.2, 19.4, 17.0, 18.0, 16.9, 16.5, 17.0, 17.6, 16.5, 18.8, 17.7, 15.5, 18.0, 17.6, 19.5, 19.0, 17.5, 19.0, 18.5, 15.0, 16.0, 18.5, 17.5, 18.0, 19.0, 17.5, 17.6, 18.7
Male	Never	21.4, 19.5, 19.3, 18.5, 19.8, 22.0, 20.0, 18.0, 21.0, 18.9, 18.1, 16.0, 18.8, 18.5, 17.8, 21.0, 18.5, 19.1, 21.0, 19.0, 21.5, 20.8, 18.9, 18.5, 19.2, 17.7, 17.5, 18.0, 18.5, 19.2, 21.5, 17.5, 19.5, 17.0, 18.2, 18.0, 19.5, 19.5, 20.5
Female	Occas	19.1, 15.4, 16.5
Male	Occas	22.2, 19.0
Female	Regul	16.3
Male	Regul	18.5, 19.5, 19.7, 18.0, 17.0, 22.5, 20.5, 20.0, 21.0

Exercise.

1. Repeat b1-b3 using aggregate( )

In [56]:
```
aggregate(Wr.Hnd~Sex+Smoke, data = data, FUN = length)
```

A data.frame: 8 × 3

Sex	Smoke	Wr.Hnd
<fct>	<fct>	<int>
Female	Heavy	3
Male	Heavy	3
Female	Never	40
Male	Never	39
Female	Occas	3
Male	Occas	2
Female	Regul	1
Male	Regul	9

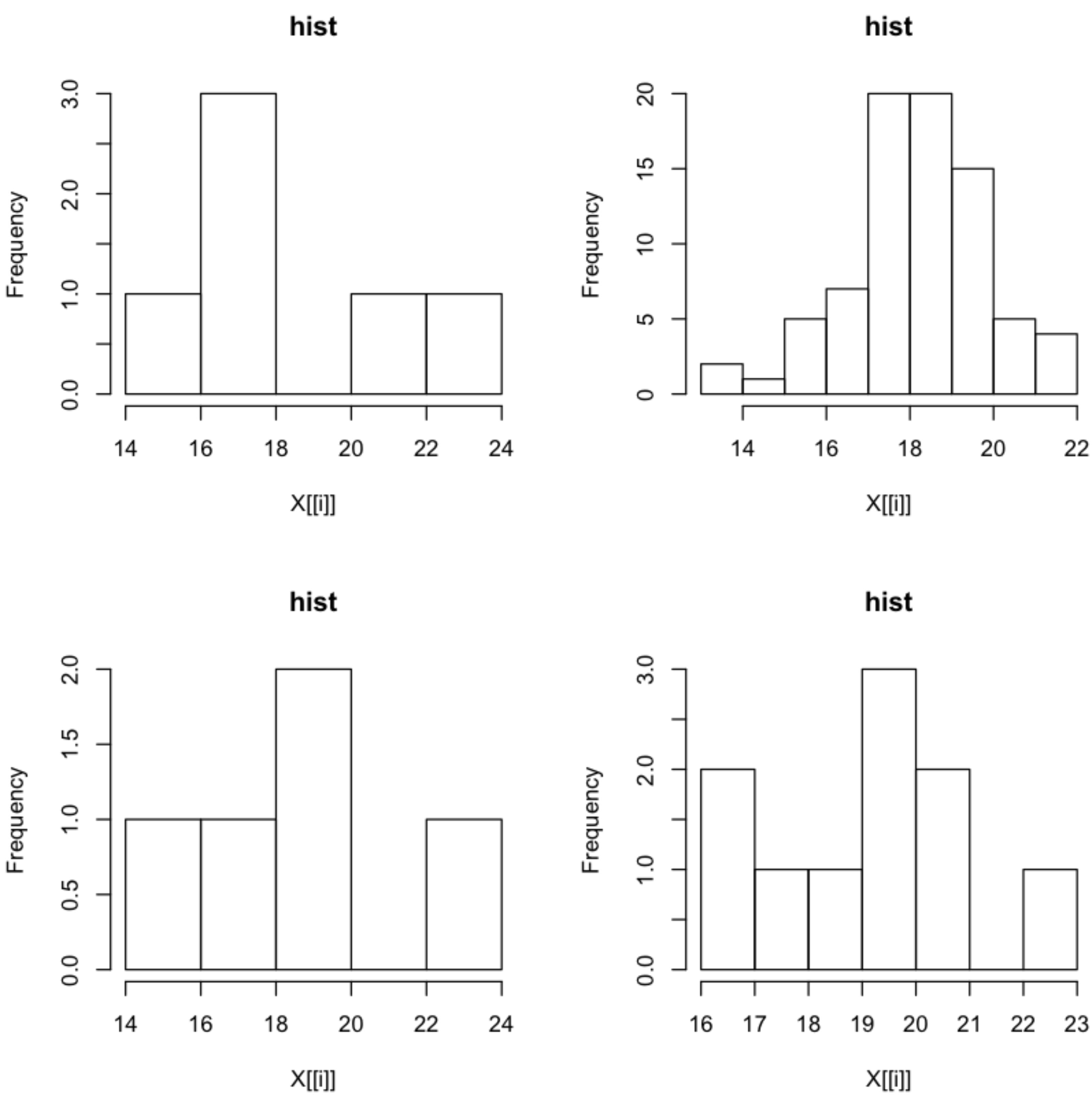
1. Make histograms of writing hand span for all four Smoke groups using aggregate( )

```
hist()
```

In [57]:
```
par(mfrow = c(2,2))
aggregate(Wr.Hnd~Smoke, data = data, FUN = hist, main = "hist")
```

A data.frame: 4 × 2

Smoke	Wr.Hnd
<fct>	<list[,6]>
Heavy	14, 16, 18, 20, 22, 24, 1, 3, 0, 1, 1, 0.08333333, 0.25000000, 0.00000000, 0.08333333, 0.08333333, 15, 17, 19, 21, 23, X[[i]], TRUE
Never	13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 2, 1, 5, 7, 20, 20, 15, 5, 4, 0.02531646, 0.01265823, 0.06329114, 0.08860759, 0.25316456, 0.25316456, 0.18987342, 0.06329114, 0.05063291, 13.5, 14.5, 15.5, 16.5, 17.5, 18.5, 19.5, 20.5, 21.5, X[[i]], TRUE
Occas	14, 16, 18, 20, 22, 24, 1, 1, 2, 0, 1, 0.1, 0.1, 0.2, 0.0, 0.1, 15, 17, 19, 21, 23, X[[i]], TRUE
Regul	16, 17, 18, 19, 20, 21, 22, 23, 2, 1, 1, 3, 2, 0, 1, 0.2, 0.1, 0.1, 0.3, 0.2, 0.0, 0.1, 16.5, 17.5, 18.5, 19.5, 20.5, 21.5, 22.5, X[[i]], TRUE



d1. Categorize 'Age' - make a new binary variable 'Adult'

```
ifelse(test, yes, no)
```

In [58]:
```
vec <- 1:5
vec

ifelse(vec>3, yes = "big", no = "small")
```

1 2 3 4 5

'small' 'small' 'small' 'big' 'big'

In [59]:
```
data$Adult <- ifelse(data$Age>=18, yes = "Yes", no = "No")
head(data)
```

A data.frame: 6 × 9

X	Sex	Wr.Hnd	NW.Hnd	Pulse	Smoke	Height	Age	Adult
<int>	<fct>	<dbl>	<dbl>	<int>	<fct>	<dbl>	<dbl>	<chr>
1	Male	21.4	21.0	63	Never	180.00	19.000	Yes
2	Male	19.5	19.4	79	Never	165.00	18.083	Yes
3	Female	16.3	16.2	44	Regul	152.40	23.500	Yes
4	Female	15.9	16.5	99	Never	167.64	17.333	No
5	Male	19.3	19.4	55	Never	180.34	19.833	Yes
6	Male	18.5	18.5	48	Never	167.00	22.333	Yes

R has if (test) {opt1} else {opt2}, what is the advantage of ifelse( )?

In [60]:
```
if (data$Age >= 18) {
    data$Adult2 = "Yes"
} else {
    data$Adult2 = "No"
}
head(data)
```

```
Warning message in if (data$Age >= 18) {:
“the condition has length > 1 and only the first element will be used”
```

A data.frame: 6 × 10

X	Sex	Wr.Hnd	NW.Hnd	Pulse	Smoke	Height	Age	Adult	Adult2
<int>	<fct>	<dbl>	<dbl>	<int>	<fct>	<dbl>	<dbl>	<chr>	<chr>
1	Male	21.4	21.0	63	Never	180.00	19.000	Yes	Yes
2	Male	19.5	19.4	79	Never	165.00	18.083	Yes	Yes
3	Female	16.3	16.2	44	Regul	152.40	23.500	Yes	Yes
4	Female	15.9	16.5	99	Never	167.64	17.333	No	Yes
5	Male	19.3	19.4	55	Never	180.34	19.833	Yes	Yes
6	Male	18.5	18.5	48	Never	167.00	22.333	Yes	Yes

In [61]:
```
# Delete Adult2
data <- subset(data, select=-c(Adult2))
```

ifelse( ) is vectorized!!!

d2. Categorize 'Wr.Hnd' into 5 groups - make a new categorical variable with 5 levels

```
1. =< 16: TP/XS
2. 16~18 (16,18]: P/S
3. 18~20 (18,20]: M/M
4. 20~22 (20,22]: G/L
5. >  22: TG/XL
```

Can we still use ifelse( )?

```
cut(x, breaks, labels = NULL, right = TRUE, ...)
```

In [62]:

```
cut.points <- c(0, 16, 18, 20, 22, Inf)

data$Hn.Grp <- cut(data$Wr.Hnd, breaks = cut.points, right = T)

head(data)
# labels as default
```

A data.frame: 6 × 10

X	Sex	Wr.Hnd	NW.Hnd	Pulse	Smoke	Height	Age	Adult	Hn.Grp
<int>	<fct>	<dbl>	<dbl>	<int>	<fct>	<dbl>	<dbl>	<chr>	<fct>
1	Male	21.4	21.0	63	Never	180.00	19.000	Yes	(20,22]
2	Male	19.5	19.4	79	Never	165.00	18.083	Yes	(18,20]
3	Female	16.3	16.2	44	Regul	152.40	23.500	Yes	(16,18]
4	Female	15.9	16.5	99	Never	167.64	17.333	No	(0,16]
5	Male	19.3	19.4	55	Never	180.34	19.833	Yes	(18,20]
6	Male	18.5	18.5	48	Never	167.00	22.333	Yes	(18,20]

In [63]:

```
# Set labels to false
data$Hn.Grp <- cut(data$Wr.Hnd, breaks = cut.points,
                   right = T, labels = FALSE)
head(data)
```

A data.frame: 6 × 10

X	Sex	Wr.Hnd	NW.Hnd	Pulse	Smoke	Height	Age	Adult	Hn.Grp
<int>	<fct>	<dbl>	<dbl>	<int>	<fct>	<dbl>	<dbl>	<chr>	<int>
1	Male	21.4	21.0	63	Never	180.00	19.000	Yes	4
2	Male	19.5	19.4	79	Never	165.00	18.083	Yes	3
3	Female	16.3	16.2	44	Regul	152.40	23.500	Yes	2
4	Female	15.9	16.5	99	Never	167.64	17.333	No	1
5	Male	19.3	19.4	55	Never	180.34	19.833	Yes	3
6	Male	18.5	18.5	48	Never	167.00	22.333	Yes	3

In [64]:
```
# Customized labels
custom.label <- c("TP/XS", "P/S", "M/M", "G/L", "TG/XL")
data$Hn.Grp <- cut(data$Wr.Hnd, breaks = cut.points,
                   right = T, labels = custom.label)
head(data)
```

A data.frame: 6 × 10

X	Sex	Wr.Hnd	NW.Hnd	Pulse	Smoke	Height	Age	Adult	Hn.Grp
<int>	<fct>	<dbl>	<dbl>	<int>	<fct>	<dbl>	<dbl>	<chr>	<fct>
1	Male	21.4	21.0	63	Never	180.00	19.000	Yes	G/L
2	Male	19.5	19.4	79	Never	165.00	18.083	Yes	M/M
3	Female	16.3	16.2	44	Regul	152.40	23.500	Yes	P/S
4	Female	15.9	16.5	99	Never	167.64	17.333	No	TP/XS
5	Male	19.3	19.4	55	Never	180.34	19.833	Yes	M/M
6	Male	18.5	18.5	48	Never	167.00	22.333	Yes	M/M

e1. Calculate the mean Wr.Hnd span of each Hn.Grp

In [65]:
```
aggregate(Wr.Hnd~Hn.Grp, data = data, FUN = mean)
```

A data.frame: 5 × 2

Hn.Grp	Wr.Hnd
<fct>	<dbl>
TP/XS	14.98000
P/S	17.37941
M/M	19.04634
G/L	21.12500
TG/XL	22.63333

e2. Calcuate the mean Wr.Hnd span of each Hnd.group without using aggregate, by, tapply

```
split(x, f, ...)
lapply(X, FUN, ...)
sapply(X, FUN, ..., simplify = TRUE)
```

In [66]:
```
# cut.points <- c(0, 16, 18, 20, 22, Inf)
num <- 1:10
let <- sample(letters[1:3], size = 10, replace = T)
cbind(num, let)
split(num, let)
```

A matrix:
10 × 2 of
type chr

num	let
1	b
2	c
3	a
4	b
5	b
6	a
7	b
8	a
9	b
10	c

$a
3 6 8

$b
1 4 5 7 9

$c
2 10

In [67]:
```
wr.hn.grp <- split(x = data$Wr.Hnd, f = data$Hn.Grp)
wr.hn.grp
```

$`TP/XS`
15.9 13 16 13 14 16 15.5 15.4 15 16

$`P/S`
16.3 17.5 17.5 18 17.5 16.4 17.2 17 17.8 18 18 17 16.9 16.5 17 17.6 16.5 17.5 17.7 17.1 18 17.6 17.5 17.7 17.5 17.5 18 18 16.5 17.5 17.5 17.6 17 18

$`M/M`
19.5 19.3 18.5 19.8 18.5 20 18.6 18.5 19.1 19.6 19.5 19.5 18.9 18.1 19.7 18.8 19.5 18.5 19.4 18.5 19.1 18.8 20 19 19.5 19 18.9 19 18.5 18.5 19.2 18.5 19 18.5 19.2 19.5 18.7 18.2 19.5 19 19.5

$`G/L`
21.4 22 21 21 20.5 21 21.5 20.8 21.3 21.5 21 20.5

$`TG/XL`
22.2 23.2 22.5

In [68]:
```
# lapply
la <- lapply(wr.hn.grp, FUN = mean)
la
```

$`TP/XS`
14.98
$`P/S`
17.3794117647059
$`M/M`
19.0463414634146
$`G/L`
21.125
$`TG/XL`
22.6333333333333

In [69]:
```
# sapply
sapply(wr.hn.grp, FUN = mean, simplify = T)
```

TP/XS	14.98
P/S	17.3794117647059
M/M	19.0463414634146
G/L	21.125
TG/XL	22.6333333333333

In [70]:
```
sapply(wr.hn.grp, FUN = mean, simplify = F)
```

$`TP/XS`
14.98
$`P/S`
17.3794117647059
$`M/M`
19.0463414634146
$`G/L`
21.125
$`TG/XL`
22.6333333333333

In [71]:
```
# vapply *
# Safer than sapply(), and a little bit faster
# because FUN.VALUE has to be specified that length and type should match

va <- vapply(wr.hn.grp, summary, FUN.VALUE = c("Min." = numeric(1),
                                                "1st Qu." = numeric(1),
                                                "Median" = numeric(1),
                                                "Mean" = numeric(1),
                                                "3rd Qu." = numeric(1),
                                                "Max." = numeric(1)))
va
```

A matrix: 6 × 5 of type dbl

	TP/XS	P/S	M/M	G/L	TG/XL
Min.	13.000	16.30000	18.10000	20.500	22.20000
1st Qu.	14.250	17.00000	18.50000	20.950	22.35000
Median	15.450	17.50000	19.00000	21.000	22.50000
Mean	14.980	17.37941	19.04634	21.125	22.63333
3rd Qu.	15.975	17.70000	19.50000	21.425	22.85000
Max.	16.000	18.00000	20.00000	22.000	23.20000

f. Calculate the 95% sample confidence intervals of Wr.Hnd in each Smoke group.

One variable for lower bound and one variable for upper bound.

$$CI = \bar{x} \pm t_{n-1,0.025} \times \sqrt{\frac{s^2}{n}}$$

where $\bar{x}$ is the sample mean and s^2 is the sample variance.

In [72]:
```
# aggregate(Wr.Hnd~Smoke, data = data, FUN = ...)
# tapply(X = data$Wr.Hnd, INDEX = list(data$Smoke), FUN = ...)
```

Unfortunately, I do not know any function in R that does this calculation.

But we know how to do it step by step.

In [73]:
```
sample.means <- aggregate(Wr.Hnd~Smoke, data = data, FUN = mean)[,2]
sample.var <- aggregate(Wr.Hnd~Smoke, data = data, FUN = var)[,2]
n <- aggregate(Wr.Hnd~Smoke, data = data, FUN = length)[,2]
t <- qt(p = 0.025, df = n - 1, lower.tail = FALSE)

# sample.means; sample.var

lb <- sample.means - t * sqrt(sample.var / n); lb
ub <- sample.means + t * sqrt(sample.var / n); ub

# How many times did we aggregate according to the group? Can on aggregate only once?
```

14.9809186861126   17.9392216758363   15.1612075858683   17.9536416263331

21.8857479805541   18.698753007708   21.7187924141317   20.6463583736669

Or, we can make our own function and integrate it into aggregate( ), by( ), or tapply( ) !!!

2.2 Write our own functions in R

A function takes in some arguments and gives some outputs

Arguments include

```
- inputs
- options
```

In [74]:
```
# The structure
func_name <- function(argument){
    statement
}
```

Example 1. Make a function for $f(x) = 2x$

In [75]:
```
# Build the function
times2 <- function(x) {
    fx = 2 * x
    return(fx)
}
# Use the function
times2(x = 5)
# or
times2(3)
```

10

6

Example 2. Make a function to calculate the integer division of a by b, return the integer part and the modulus.

In [76]:
```
# R has operators that do this
9 %/% 2
9 %% 2
```

4

1

```
 floor( ) takes a single numeric argument x and returns a numeric vector containing the largest int
egers not greater than the corresponding elements of x.
```

In [77]:
```
int.div <- function(a, b){
    int <- floor(a/b)
    mod <- a - int*b
    return(list(integer = int, modulus = mod))
}
```

In [78]:
```
# class(result)
# Recall: how do we access the modulus?
result <- int.div(21, 4)
result
```

$integer
5
$modulus
1

In [79]:
```
result$modulus
```

1

In [80]:
```
int.div <- function(a, b){
    int <- a%/%b
    mod <- a%%b
    return(cat(a, "%%", b, ": \n integer =", int,"\n ------------------", " \n modulus =", mod,
"\n"))
}
int.div(21,4)
```

```
21 %% 4 :
 integer = 5
 ------------------
 modulus = 1
```

In [81]:
```
int.div <- function(a, b){
    int <- a%/%b
    mod <- a%%b
    return(c(numerator = a, denom = b))
}
int.div(21, 4)
```

numerator	21
denom	4

Example 3. Make the simplest canadian AI chatbot

A function can return something other than an R object, say some voice.

In [82]:
```
# No need to worry about the details here.
# Just want to show that functions do not always have to return() something.
AIcanadian <- function(who, reply_to) {
    system(paste("say -v", who, "Sorry!"))
}
# AIcanadian("Alex", "Sorry I stepped on your foot.")
```

In [83]:
```
# Train my chatbot - AlphaGo style.
# I'll let Alex and Victoria talk to each other.
# MacOS has their voices recorded.
# chat_log <- rep(NA, 8)
# for (i in 1:8) {
#     if (i == 1) {
#         chat_log[1] <- "Sorry I stepped on your foot."
#         system("say -v Victoria Sorry, I stepped on your foot.")
#     } else {
#         if (i %% 2 == 0)
#             chat_log[i] <- AIcanadian("Alex", chat_log[i - 1])
#         else
#             chat_log[i] <- AIcanadian("Victoria", chat_log[i - 1])
#     }
# }
# chat_log
```

Example 4. Check one summary statistic by Smoke group of our 'data' data.

Function arguments can be basically anything, say another function.

In [84]:
```
data_summary <- function(func) {
    data <- read.csv("https://raw.githubusercontent.com/ly129/MiCM2020/master/sample.csv", head
er = TRUE)
    by(data = data$Wr.Hnd, INDICES = list(data$Smoke), FUN = func)
}
data_summary(var)
```

```
: Heavy
[1] 10.82267
------------------------------------------------------------
: Never
[1] 2.874635
------------------------------------------------------------
: Occas
[1] 6.973
------------------------------------------------------------
: Regul
[1] 3.542222
```

Example 5. Default argument value & stop execution

In [85]:
```
a_times_2_unless_you_want.something.else.but.I.refuse.3 <- function(a, b=2){
    if (b == 3) {
        stop("I refuse 3!")
    }
    if (b == 4){
        warning("4 sucks too")
    }
    a*b
}
```

In [86]:
```
a_times_2_unless_you_want.something.else.but.I.refuse.3(a = 5)
```

10

In [87]:
```
a_times_2_unless_you_want.something.else.but.I.refuse.3(a = 5, b = 4)
```

```
Warning message in a_times_2_unless_you_want.something.else.but.I.refuse.3(a = 5, :
“4 sucks too”
```

20

In [88]:
```
# a_times_2_unless_you_want.something.else.but.I.refuse.3(a = 5, b = 3)
```

Exercise:

1. Make a function to calculate sample confidence intervals (2.1 f)

In [89]:
```
sample.ci <- function(x){
    mean <- mean(x)
    var <- var(x)
    n <- length(x)
    t <- qt(p = .025, df = n - 1, lower.tail = FALSE)
    lb <- mean - t * sqrt(var / n); lb
    ub <- mean + t * sqrt(var / n); ub
    return(c(lower = lb, upper = ub))
}
```

In [90]:
```
sample.ci(c(1453,45,14,51,235,123,4,123,412))
```

lower	-80.8877593881401
upper	627.554426054807

1. Use the function in 1 with aggregate(), by() or apply() to calculate the sample confidence intervals (2.1 f)

In [91]:
```
aggregate(Wr.Hnd~Smoke, data = data, FUN = sample.ci)
```

A data.frame: 4 × 2

Smoke	Wr.Hnd
<fct>	<dbl[,2]>
Heavy	14.98092, 21.88575
Never	17.93922, 18.69875
Occas	15.16121, 21.71879
Regul	17.95364, 20.64636

3. Exercises

A fake dataset is generated. Results should make no biological sense.

In [92]:
```
set.seed(20200306)
N <- 200
height <- round(rnorm(n = N, mean = 180, sd = 10)) # in centimeter
weight <- round(rnorm(n = N, mean = 80, sd = 10)) # in kilograms
age <- round(rnorm(n = N, mean = 50, sd = 10))
treatment <- sample(c(TRUE, FALSE), size = N, replace = T, prob = c(0.3,0.7))
HF <- sample(c(TRUE, FALSE), size = N, replace = T, prob = c(0.1,0.9))

fake <- data.frame(height, weight, age, treatment, HF)
head(fake)
```

A data.frame: 6 × 5

height	weight	age	treatment	HF
<dbl>	<dbl>	<dbl>	<lgl>	<lgl>
186	92	60	FALSE	FALSE
155	74	58	FALSE	TRUE
182	79	62	FALSE	FALSE
178	101	54	FALSE	FALSE
182	72	54	FALSE	FALSE
159	66	41	FALSE	TRUE

1. (Vectorized operation) Calculate BMI for every individual

$$\text{BMI} = \text{weight}(kg)/\text{height}(m)^2$$

In [93]:
```
names(fake)
fake$BMI <- fake$weight/(fake$height/100)^2
head(fake)
```

'height' 'weight' 'age' 'treatment' 'HF'

A data.frame: 6 × 6

height	weight	age	treatment	HF	BMI
<dbl>	<dbl>	<dbl>	<lgl>	<lgl>	<dbl>
186	92	60	FALSE	FALSE	26.59267
155	74	58	FALSE	TRUE	30.80125
182	79	62	FALSE	FALSE	23.84978
178	101	54	FALSE	FALSE	31.87729
182	72	54	FALSE	FALSE	21.73651
159	66	41	FALSE	TRUE	26.10656

2. (Categorization) BMI Categories:

```
- Underweight = <18.5
- Normal weight = 18.5–24.9
- Overweight = 25–29.9
- Obesity = BMI of 30 or greater
```

In [94]:
```
cut.pts <- c(-Inf, 18.5, 25, 30, Inf)
labs <- c("Underweight", "Normal weight", "Overweight", "Obesity")
fake$BMI.cat <- cut(fake$BMI, breaks = cut.pts, labels = labs, right = F)
head(fake)
```

A data.frame: 6 × 7

height	weight	age	treatment	HF	BMI	BMI.cat
<dbl>	<dbl>	<dbl>	<lgl>	<lgl>	<dbl>	<fct>
186	92	60	FALSE	FALSE	26.59267	Overweight
155	74	58	FALSE	TRUE	30.80125	Obesity
182	79	62	FALSE	FALSE	23.84978	Normal weight
178	101	54	FALSE	FALSE	31.87729	Obesity
182	72	54	FALSE	FALSE	21.73651	Normal weight
159	66	41	FALSE	TRUE	26.10656	Overweight

3. (*apply) Mean BMI of each BMI group

In [95]:
```
# aggregate() or tapply()
aggregate(BMI~BMI.cat, data = fake, FUN = mean)
```

A data.frame: 4 × 2

BMI.cat	BMI
<fct>	<dbl>
Underweight	16.17253
Normal weight	22.09822
Overweight	26.94884
Obesity	32.79935

In [96]:
```
# split() and lapply()
BMI.grp <- split(fake$BMI, f = fake$BMI.cat)
lapply(BMI.grp, FUN = mean)
```

$Underweight
16.1725259238271
$`Normal weight`
22.0982178055369
$Overweight
26.9488351034313
$Obesity
32.7993482108602

4. (Aggregation) Proportion with heart failure in each BMI-treatment group

In [97]:
```
# Trick:
FALSE + TRUE + TRUE
F + F + T + T
```

2

2

In [98]:
```
aggregate(HF~BMI.cat+treatment, data = fake, FUN = sum)
```

A data.frame: 8 × 3

BMI.cat	treatment	HF
<fct>	<lgl>	<int>
Underweight	FALSE	0
Normal weight	FALSE	9
Overweight	FALSE	3
Obesity	FALSE	4
Underweight	TRUE	1
Normal weight	TRUE	5
Overweight	TRUE	2
Obesity	TRUE	0

5. Write a function that allows user to specify

```
- a dataset
- the (binary) treatment variable
- the (binary) outcome variable
```

and return a cross-tabulation (a 2x2 table).

In [99]:
```
tab2by2 <- function(data, treatment, outcome){
    sub <- data[, c(treatment, outcome)]
    return(table(sub))
}
```

In [100]:
```
tab2by2(fake, treatment = "treatment", outcome = "HF")
```

```
         HF
treatment FALSE TRUE
    FALSE   130   16
    TRUE     46    8
```

5 Pro. The function should be able to check whether the treatment/outcome variables are binary or not. Continuous variables will be dichotomized based on a user-defined threshold.

In [101]:
```
tab2by2.pro <- function(data, treatment, outcome, treatment.threshold, outcome.threshold){
    tx <- data[, treatment]
    rx <- data[, outcome]

    if (length(table(tx))>2) {
        if (missing(treatment.threshold)) {
            stop("Non-binary treatment. Please provide a threshold.")
        } else {
            binary.treatment <- ifelse(tx<=treatment.threshold,
                                       yes = paste("<=", treatment.threshold),
                                       no = paste(">", treatment.threshold))
        }
    } else {
        binary.treatment <- tx
    }

    if (length(table(rx))>2) {
        if (missing(outcome.threshold)) {
            stop("Non-binary outcome. Please provide a threshold.")
        } else {
            binary.outcome <- ifelse(rx<=outcome.threshold,
                                     yes = paste("<=", outcome.threshold),
                                     no = paste(">", outcome.threshold))
        }
    } else {
        binary.outcome <- rx
    }


    return(table(treatment = binary.treatment, outcome = binary.outcome))
}
```

In [102]:
```
tab2by2.pro(fake, treatment = "age", outcome = "BMI")
```

```
Error in tab2by2.pro(fake, treatment = "age", outcome = "BMI"): Non-binary treatment. Please p
rovide a threshold.
Traceback:

1. tab2by2.pro(fake, treatment = "age", outcome = "BMI")
2. stop("Non-binary treatment. Please provide a threshold.")   # at line 7 of file <text>
```

In [103]:
```
tab2by2.pro(fake, treatment = "age", outcome = "BMI", treatment.threshold = 50)
```

```
Error in tab2by2.pro(fake, treatment = "age", outcome = "BMI", treatment.threshold = 50): Non-
binary outcome. Please provide a threshold.
Traceback:

1. tab2by2.pro(fake, treatment = "age", outcome = "BMI", treatment.threshold = 50)
2. stop("Non-binary outcome. Please provide a threshold.")   # at line 19 of file <text>
```

In [104]:
```
tab2by2.pro(fake, treatment = "age", outcome = "BMI", treatment.threshold = 50, outcome.thresho
ld = 20)
```

```
         outcome
treatment <= 20 > 20
    <= 50     6   93
    > 50     11   90
```

In [105]:
```
tab2by2.pro(fake, treatment = "age", outcome = "HF")
```

```
Error in tab2by2.pro(fake, treatment = "age", outcome = "HF"): Non-binary treatment. Please pr
ovide a threshold.
Traceback:

1. tab2by2.pro(fake, treatment = "age", outcome = "HF")
2. stop("Non-binary treatment. Please provide a threshold.")   # at line 7 of file <text>
```

In [106]:
```
# HF is binary, so it is OK if "outcome.threshold" is missing.
tab2by2.pro(fake, treatment = "age", outcome = "HF", treatment.threshold = 50)
```

```
         outcome
treatment FALSE TRUE
    <= 50    93    6
    > 50     83   18
```

6. Specific task in your own research

In [ ]:
```
```