Introduction to R

Session 2 – Data subsetting

Statistical Consulting Centre

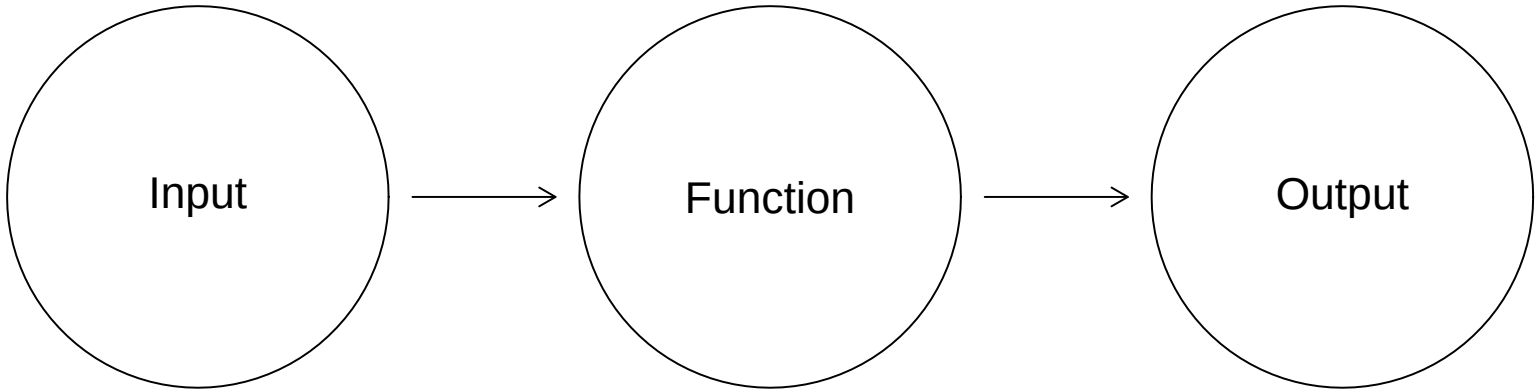
consulting@stat.auckland.ac.nz
The Department of Statistics
The University of Auckland

19 July, 2017

SCIENCE
DEPARTMENT OF STATISTICS

Functions

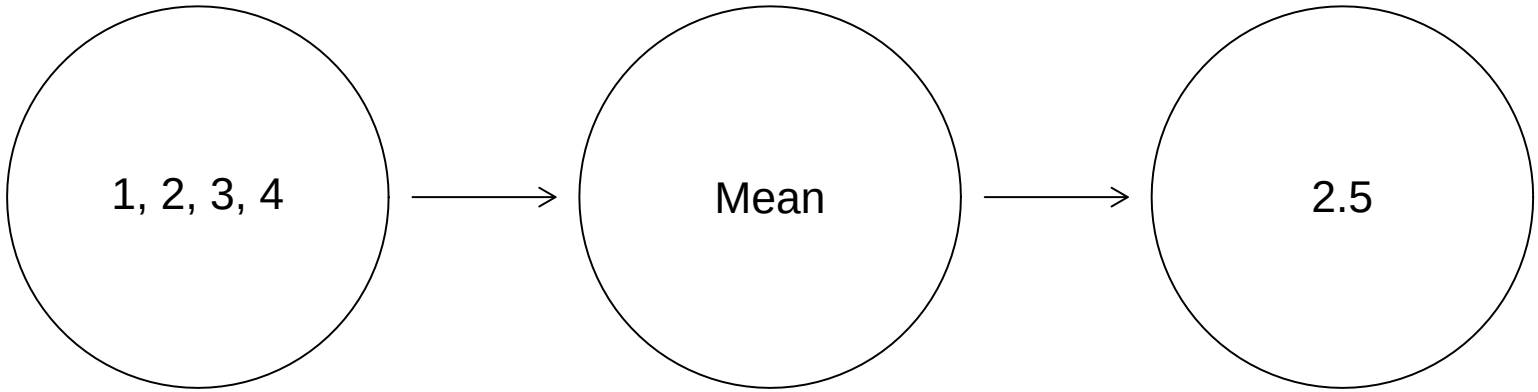
A function is a relationship between a set of inputs (arguments) and a set of outputs. E.g., the function is fed some information on which it operates, the results of which are the output.



This is an essential building block for the R package.

Functions

We have seen many functions, e.g. `log`, `mean`, `table`, `with`, etc.



Working with functions

- Functions can be user-defined, i.e., you can write your own.
- Output is the last line of the function. You can use `return()` to specifiy the output.
- Here is a function calculates the standard error of the mean (SEM).

```
mystder <- function(x) {
    mysd <- sd(x, na.rm = TRUE)  # Calc std. deviation
    n <- length(x)  # Calc sample size
    mysd/sqrt(n)  # Definition of SEM
}
mystder(Patient.df$Height)
```

```
## [1] 0.02999245
```

- A set of user-defined functions can be bundled together into an R package.

Getting data into R

- Base R includes only functions which read data sets saved in simple file formats, e.g. csv, txt, tab delimited, etc.
- What if your data was saved in another format, e.g. Excel?
- The readxl package for R contains functions that may help, https://cran.r-project.org/web/packages/readxl/index.html

```
library(readxl)
excel <- read_excel("data.xlsx", sheet = 1)
```

Getting data into R

- What if your data was saved in another format, e.g. Excel, STATA, SPSS, or SAS spreadsheets?
- The haven package for R contains functions that may help, https://cran.r-project.org/web/packages/haven/index.html

```
library(haven)
stata <- read_dta("data.dta")
spss <- read_sav("data.sav")
sas <- read_sas("data.sas7bdat")
sasxport <- read_xpt("data.xpt")
```

However, it is always the easiest and safest to read data into R from a csv file.

Packages

- Currently, the CRAN package repository features 10,098 available packages (4 Jul. 2017). There are about 13,169 CRAN, BioConductor and Github packages in total.
- To install packages from the `R` GUI, click on `Packages` → `Install Package(s) ...` → `New Zealand (or whatever region you are located)` → `Package name`
- Or, you can type `install.packages(`*`package name`*`)`, e.g. `install.packages("haven")`.
- After the installation, use `library("`*`package name`*`")` to load it into `R`.

Note: Installation is performed only once; however, it must be loaded (i.e. use the command `library("`*`package name`*`")`) in *every* `R` session.

Bioconductor Packages

- To install core packages, type the following in an R command window:

```
source("https://bioconductor.org/biocLite.R")
biocLite()
```

- Install specific packages, e.g., "GenomicFeatures" and "AnnotationDbi", with

```
biocLite(c("GenomicFeatures", "AnnotationDbi"))
```

Note: Installation is performed only once; however, it must be loaded (i.e. use the command `library("package name")`) in *every* R session.

`which` individual does Smoke?

Let's use R's powerful subsetting capabilities to select those cases for which the value of Smoke is 1 (i.e. Yes).

```
index <- which(Patient.df$Smoke == 1)
index
```

```
##   [1]   6   7   8  12  15  16  18
##   [8]  19  20  21  22  23  28  33
##  [15]  35  38  45  47  48  52  53
##  [22]  55  56  57  58  64  70  75
##  [29]  79  80  81  89  90  91  93
##  [36]  99 108 109 110 116 118 119
##  [43] 122 123 124 127 136 137 144
##  [50] 146 148 154 155 162 172 174
##  [57] 180 182 187 188 189 191 192
##  [64] 195 198 204 205 210 213 221
##  [71] 225 229 230 232 233 234 237
```

How many are there?

```
# Use length() to count the number of elements in
# 'index'.
length(index)
```

```
## [1] 4371
```

- `Patient.df$Smoke == 1` gives a vector of True/False for every observations.
- `which()` gives a vector of the position of `TRUE`.
- `length()` tells us how many elements there are in the vector.

How many are there?

Another way is to sum up how many `TRUE`s in the logical vector `Patient.df$Smoke == 1`

```
sum(Patient.df$Smoke == 1)
```

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

How many are there?

Another way is to sum up how many `TRUE`s in the logical vector `Patient.df$Smoke == 1`

```
sum(Patient.df$Smoke == 1, na.rm = TRUE)
```

```
## [1] 4371
```

Who are they?

```
# Use square brackets to extract IDs corresponding
# to the cases numbers contained in 'index'
index <- which(Patient.df$Smoke == 1)
Patient.df$Patient.ID[index]
```

```
##   [1]   19   34   44   51   55   56   67
##   [8]   70   71   72   73   74   90  110
##  [15]  115  120  129  139  140  154  155
##  [22]  159  161  164  165  180  190  210
##  [29]  217  218  219  242  249  251  255
##  [36]  275  297  298  303  317  331  335
##  [43]  343  344  346  353  376  377  395
##  [50]  400  403  427  429  445  470  474
##  [57]  485  488  503  507  509  519  520
##  [64]  524  528  537  539  550  554  570
##  [71]  578  586  588  592  593  595  604
##  [78]  606  618  620  639  644  646  650
```

Subsetting

Square brackets [] are used to extract subsets of data.

```
# First element only
Patient.df$Height[1]
```

```
## [1] 70.4
```

```
# All but the first element
Patient.df$Height[-1]
```

```
##      [1] 63.9 61.8 69.8   NA 70.2 62.6 64.4 64.3
##      [9] 67.6 59.5 71.1 70.2 61.0 68.0 73.4 61.1
##     [17] 63.8 67.8 61.7 71.1 68.4 65.7 61.4 59.8
##     [25] 61.8 60.3 66.1 67.9 64.1 59.1 70.0 62.4
##     [33] 63.7 65.1 64.3 65.4 69.7 67.5 68.0 71.1
##     [41] 63.6 63.2 64.6 74.8 61.4 64.5 65.2 71.9
##     [49] 67.4 66.5 65.7 68.5 60.0 67.2 69.4 65.8
```

Subsetting

Square brackets [] are used to extract subsets of data.

```
# Elements 3 through 8
Patient.df$Height[3:8]
```

```
## [1] 61.8 69.8   NA 70.2 62.6 64.4
```

```
# Elements 3 and 8
Patient.df$Height[c(3, 8)]
```

```
## [1] 61.8 64.4
```

More on subsetting

Subsetting two-dimensional arrays, such as data frames, requires the use of *two indices.*

```
# First row or record
Patient.df[1, ]
```

```
##   Patient.ID Age  Sex Race Weight Height Smoke
## 1          3  21 Male    1  179.5   70.4    NA
```

More on subsetting

For data frames, you need to use two indices.

```
# Second column or variable
Patient.df[, 2]
```

```
##      [1] 21 32 48 35 48 44 42 24 67 56 82 44 50 36
##     [15] 48 32 66 70 63 37 60 42 58 80 80 23 83 28
##     [29] 90 86 27 72 34 21 45 84 36 28 69 63 31 25
##     [43] 41 33 39 55 72 32 27 55 78 65 57 26 31 31
##     [57] 49 61 65 42 48 80 22 43 72 34 59 43 22 73
##     [71] 66 69 79 85 48 32 73 32 28 62 38 56 48 22
##     [85] 21 66 89 61 55 52 51 66 33 54 24 83 31 42
##     [99] 24 47 35 21 59 64 41 29 63 33 53 60 76 31
##    [113] 20 77 27 40 30 78 43 24 80 66 48 66 89 42
##    [127] 47 46 22 73 46 38 42 81 77 65 24 21 36 73
##    [141] 81 20 39 38 61 30 76 58 81 67 26 53 51 33
##    [155] 58 85 35 30 85 72 61 38 29 26 45 34 61 70
```

More on subsetting

For data frames, you need to use two indices.

```
# Some rows and columns
Patient.df[2:5, 4:7]
```

```
##   Race Weight Height Smoke
## 2    1     NA   63.9    NA
## 3    1  149.7   61.8     2
## 4    1  203.5   69.8    NA
## 5    1  155.3     NA     2
```

More on subsetting

For data frames, you need to use two indices.

```
# Rows by number, columns by name
Patient.df[1:10, c("Race", "Smoke")]
```

```
##    Race Smoke
## 1     1    NA
## 2     1    NA
## 3     1     2
## 4     1    NA
## 5     1     2
## 6     2     1
## 7     2     1
## 8     1     1
## 9     2    NA
## 10    1     2
```

Missing values

-R reserves the object NA (Not Available) for elements of a vector that are missing or unavailable. - Use of `is.na()` to search for missing values requires that they are recorded as NA. - na will not do because R is case sensitive!

```
sum(is.na(Patient.df$Smoke))
```

```
## [1] 8404
```

Missing values

The default option of `table()` ignores `NA`s when constructing frequency tables. Now that all occurrences of `"NA, refused"` have been replaced with `NA`, missing values will no longer be shown in frequency tables constructed using `table()`.

```
table(Patient.df$Smoke)
```

```
##
##    1    2
## 4371 4255
```

Missing values

If you still want to see how many `NA`s in the frequency tables, you can change the `useNA` arguement to `"always"` in `table()`.

```
table(Patient.df$Smoke, useNA = "always")
```

```
##
##    1    2 <NA>
## 4371 4255 8404
```

Data Cleaning

The `ifelse()` function provides a quick way to convert the `Smoke` variable in `Patient.df` from number to words, i.e.

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

- `test`: a logical test.
- `yes`, what happens if the test is True.
- `no`, what happens if the test is False.

Data Cleaning

```
Patient.df$Smoke.group <-
  with(Patient.df, ifelse(Smoke == 1, "Yes", "No"))

table(Patient.df$Smoke.group)
```

```
##
##   No  Yes
## 4255 4371
```

Data Cleaning

How about `Race`? One `ifelse()` inside another `ifelse()`:

```
Patient.df$Race.group <-
  with(Patient.df,
       ifelse(Race == 1, "Caucasian",
              ifelse(Race == 2, "African", "Other")))
table(Patient.df$Race.group)
```

```
##
##   African Caucasian     Other
##      4860     11612       553
```

Data Cleaning

```
head(Patient.df)
```

```
##   Patient.ID Age    Sex Race Weight Height Smoke
## 1          3  21   Male    1  179.5   70.4    NA
## 2          4  32 Female    1     NA   63.9    NA
## 3          9  48 Female    1  149.7   61.8     2
## 4         10  35   Male    1  203.5   69.8    NA
## 5         11  48   Male    1  155.3     NA     2
## 6         19  44   Male    2  189.6   70.2     1
##   Smoke.group Race.group
## 1        <NA>  Caucasian
## 2        <NA>  Caucasian
## 3          No  Caucasian
## 4        <NA>  Caucasian
## 5          No  Caucasian
## 6         Yes    African
```

Data Cleaning

```
str(Patient.df)
```

```
## 'data.frame':    17030 obs. of  9 variables:
##  $ Patient.ID : int  3 4 9 10 11 19 34 44 45 48 ...
##  $ Age        : int  21 32 48 35 48 44 42 24 67 56 ...
##  $ Sex        : chr  "Male" "Female" "Female" "Male" ...
##  $ Race       : int  1 1 1 1 1 2 2 1 2 1 ...
##  $ Weight     : num  180 NA 150 204 155 ...
##  $ Height     : num  70.4 63.9 61.8 69.8 NA 70.2 62.6 64.4
##  $ Smoke      : int  NA NA 2 NA 2 1 1 1 NA 2 ...
##  $ Smoke.group: chr  NA NA "No" NA ...
##  $ Race.group : chr  "Caucasian" "Caucasian" "Caucasian" "C
```

Data Cleaning

Now we have created new variables, `Smoke.group` and `Race.group`, the next step is to remove the `Smoke` and `Race` variables in the `Patient.df`.

```
names(Patient.df)
```

```
## [1] "Patient.ID"   "Age"          "Sex"
## [4] "Race"         "Weight"       "Height"
## [7] "Smoke"        "Smoke.group"  "Race.group"
```

Data Cleaning

Now we have created new variables, `Smoke.group` and `Race.group`, the next step is to remove the `Smoke` and `Race` variables in the `Patient.df`.

```
Smoke.index <- which(names(Patient.df) == "Smoke")
Smoke.index
```

```
## [1] 7
```

```
Race.index <- which(names(Patient.df) == "Race")
Race.index
```

```
## [1] 4
```

```
names(Patient.df[, -c(Smoke.index, Race.index)])
```

```
## [1] "Patient.ID"   "Age"          "Sex"
## [4] "Weight"       "Height"       "Smoke.group"
## [7] "Race.group"
```

Data Cleaning with %in% operator

```
names(Patient.df)
```

```
## [1] "Patient.ID"   "Age"          "Sex"
## [4] "Race"         "Weight"       "Height"
## [7] "Smoke"        "Smoke.group"  "Race.group"
```

```
names(Patient.df) %in% c("Smoke", "Race")
```

```
## [1] FALSE FALSE FALSE  TRUE FALSE FALSE  TRUE
## [8] FALSE FALSE
```

```
names(Patient.df[, !names(Patient.df) %in% c("Smoke",
    "Race")])
```

```
## [1] "Patient.ID"   "Age"          "Sex"
## [4] "Weight"       "Height"       "Smoke.group"
## [7] "Race.group"
```

Data Cleaning with %in% operator

```
Patient.df <- Patient.df[, !names(Patient.df) %in%
    c("Smoke", "Race")]
```

Your turn

First to calculate `BMI`:

$$\texttt{BMI} = \frac{\texttt{Weight(kg)}}{\texttt{Height(m)}^2} = \frac{\texttt{Weight(pounds)}}{\texttt{Height(inches)}^2} \times 703$$

Then, create a `BMI.group` for `normal`, `overweight` and `obese`, where,

- overweight is a BMI greater than or equal to 25; and
- obesity is a BMI greater than or equal to 30.

Your turn

To calculate BMI:

$$\text{BMI} = \frac{\text{Weight(kg)}}{\text{Height(m)}^2} = \frac{\text{Weight(pounds)}}{\text{Height(inches)}^2} \times 703$$

```
Patient.df$BMI <- (Patient.df$Weight/Patient.df$Height^2)*703
mean(Patient.df$BMI, na.rm = TRUE)
```

```
## [1] 27.03084
```

Or,

```
# Calculate BMI using Weight and Height
Patient.df$BMI <- with(Patient.df, (Weight/Height^2)*703)
mean(Patient.df$BMI, na.rm = TRUE)
```

```
## [1] 27.03084
```

Your turn

```
Patient.df$BMI.group <-
  ifelse(Patient.df$BMI >= 30, "obese",
         ifelse(Patient.df$BMI >= 25, "overweight",
          "normal"))
table(Patient.df$BMI.group)
```

```
##
##     normal      obese overweight
##       6916       4185       5866
```

```
table(Patient.df$BMI.group, useNA = "always")
```

```
##
##     normal      obese overweight       <NA>
##       6916       4185       5866         63
```

Subsetting in calculations

```
with(Patient.df, table(Race.group, BMI.group))
```

```
##              BMI.group
## Race.group    normal obese overweight
##   African       1858  1439       1546
##   Caucasian     4774  2627       4168
##   Other          282   119        149
```

Subsetting in calculations

Produce the last table with Race groups of `African` and `Caucasian`, and BMI groups of `obese` and `overweight`

```
exclude.rows <- with(Patient.df, which(Race.group ==
    "Other" | BMI.group == "Normal"))

head(Patient.df[-exclude.rows, c("Race.group", "BMI.group")])
```

```
##   Race.group  BMI.group
## 1  Caucasian overweight
## 2  Caucasian       <NA>
## 3  Caucasian overweight
## 4  Caucasian overweight
## 5  Caucasian       <NA>
## 6    African overweight
```

Subsetting in calculations

```
R.B.tab <- with(Patient.df[-exclude.rows,],
                table(Race.group, BMI.group))
R.B.tab
```

```
##              BMI.group
## Race.group  normal obese overweight
##   African     1858  1439       1546
##   Caucasian   4774  2627       4168
```

Subsetting in calculations

Convert counts to percentages rounded to 1 decimal place.

```
round(prop.table(R.B.tab) * 100, 1)
```

```
##              BMI.group
## Race.group    normal obese overweight
##   African       11.3   8.8        9.4
##   Caucasian     29.1  16.0       25.4
```

Easier way

```
R.B.tab1 <- with(Patient.df, table(Race.group, BMI.group))
dim(R.B.tab1)
```

```
## [1] 3 3
```

```
row.names(R.B.tab1)
```

```
## [1] "African"   "Caucasian" "Other"
```

```
colnames(R.B.tab1)
```

```
## [1] "normal"     "obese"      "overweight"
```

Easier way

```
colnames(R.B.tab1)
```

```
## [1] "normal"     "obese"      "overweight"
```

```
which(row.names(R.B.tab1) == "Other")
```

```
## [1] 3
```

```
which(colnames(R.B.tab1) == "Normal")
```

```
## integer(0)
```

Easier way

```
R.B.tab1[-3, -1]
```

```
##              BMI.group
## Race.group    obese overweight
##   African      1439       1546
##   Caucasian    2627       4168
```

```
round(prop.table(R.B.tab1[-3, -1]) * 100, 1)
```

```
##              BMI.group
## Race.group    obese overweight
##   African      14.7       15.8
##   Caucasian    26.9       42.6
```

Subsetting in calculations

```
# Mean BMI level of females
with(Patient.df, mean(BMI[Sex == "Female"], na.rm = TRUE))
```

```
## [1] 27.45053
```

```
# Mean BMI level of females and Caucasian
with(Patient.df,
     mean(BMI[Sex == "Male" &
                  Race.group == "Caucasian"],
          na.rm = TRUE))
```

```
## [1] 26.64311
```

Summary

- Making R functions
- Installing and loading R pacakges
- Subsetting vectors and datasets
- `ifelse()` function