Session 1: R Basics

Ya-Feng Wen

2020-06-13

Objectives

- Brief introduction of RStudio
- Install packages
- Basic date type and structures
  - Operators
  - Subsetting

Download R and RStudio

- R: http://lib.stat.cmu.edu/R/CRAN/
- RStudio: https://rstudio.com/
- Check R version: `sessionInfo()` or `version`
- Update R: use `installr` package for Windows and `updateR` package for Mac. Detailed instruction can be found here.
  - One benefit of using `installr` is that it will update R and all its package

```
# installing/loading the package:
if(!require(installr)) {
install.packages("installr"); require(installr)} #load / install+load installr

# using the package:
updateR()
```

- Update RStudio: `Help>Check for Updates`

Install packages

- Available packages on R CRAN: https://cran.r-project.org/
- Developed and maintained by RStudio: https://rstudio.com/products/rpackages/
- Install stable release packages from CRAN: `install.packages("the package's name")` Only need to do this once.
- Use the package: `library(the package's name)`. Need for every session.
- Update packages: `update.packages(ask = FALSE)` (only update those on CRAN)
- If packages is not avialble in CRAN, use `devtools::install_github("URL")`
- Update all packages after updating R:

```
## get packages installed
packs = as.data.frame(installed.packages(.libPaths()[1]), stringsAsFactors = F)

## and now re-install install packages using install.packages()
install.packages(packs$Package)
```

Why use RStudio?

- It's interactive. You can test your code (console) and document (script, markdown, slides, etc.) simultaneously.
- It has four panels: Console, Scripts, Environments, Files/Plots/Packages/Help.
- Publish your work to RPubs. Example
- Autocompletion to save time and reduce errors.
- Spell correction
- Execute code directly from R Studio script by using the Ctrl + Enter in Windows or Return in Mac.
- Ctrl + 1 and Ctrl + 2 jump between the script and the console windows.
- Use # signs to comment. (comment as detailed as you can)

Basic Data Types and Data Structures in R

Everything in R is an object. The object can be character, integer, floating point, Boolean etc. Unlike C and java, variables (reserved memory locations to store values) are not declared as some data type. The variables are assigned with R-Objects and the data type of the R-object becomes the data type of the variable.

Commonly used objects:

- **Vectors**: use `vector()` to create a vector. There are 6 data types of these atomic vectors. (When we call a vector atomic, we mean that the vector only holds data of a single data type.) A vector is one-dimensional and each element is of the same type.
- **Lists**: `list()`. A list can contain many different types of elements, like vectors, functions or another list. It can be extremely useful inside functions. Since functions in R can only return a single object, list can "staple" many different results into a single object.
- **Matrices**: `matrix()`. A matrix is a two-dimensional rectangular data set.
- **Arrays**: `array()`. A array can be of any number of dimensions.
- **Factors**: `factor()`. Factors are the r-objects which are created using a vector. It stores the vector along with the distinct values of the elements in the vector as labels.
- **Data Frames**: `data.frame()`. Data frames are tabular data objects.

Six classes of atomic vectors:

- **Logical**: `TRUE`, `FALSE`. You can do math with logical vectors.
- **Numeric**: `2`, `2.5`
- **Integer**: `2L`, `0L`. Produce them using `:` shortcut. For example: `1:0`
- **Complex**: `3+2i`
- **Character**:`"a"`, `"good"`, `"TRUE"`. Character data usually can't go directly into a statistical model1, but factor data can. *Note: Most R models will automatically convert character data to factors. The default reference is chosen alphabetically.*
- **Raw**: `"Hello"` is stored as 48 65 6c 6c 6f (I've never used it.)

Functions to check the features of vectors and other objects:

- `class()`: what kind of object is it? (high level)
- `typeof()`: what is the data type of this object? (low level)
- `length()`: how long is it?
- `attributes()`: does it have any metadata?

Functions to check the attributes of objects:

- `names()`
- `dimnames()`
- `dim()`

Functions for data frame (commonly used):

- `head()` - shows first 6 rows
- `tail()` - shows last 6 rows
- `dim()` - returns the dimensions of data frame (i.e. number of rows and number of columns)
- `nrow()` - number of rows
- `ncol()` - number of columns
- `str()` - structure of data frame - name, type and preview of data in each column
- `names()` or `colnames()` - both show the names attribute for a data frame
- `sapply(dataframe, class)` - shows the class of each column in the data frame

Operators

Assignment operators

- Use `<-` for assignments. (Short cut: Alt + -)
- Use `=` for specify the values of arguments in functions.

Arithmetic operators

Operator	Description
`+`	Addition
`-`	Subtraction
`*`	Multiplication
`/`	Division
`^`	Exponent
`%%`	Modulus (Remainder from division)
`%/%`	Integer Division

Examples:

```
# single element vector
x <- 2; y <- 15
x+y
x-y
x*y
y/x
y%%x
y^2

# use `c()` to concatenate or combine element to make vectors
x <- c(1,3,5)
y <- c(2,4,6)
x+y

fruits <- c("apple", "banana", "cherry")
# which word has odd number of letters?
name_lengths <- nchar(fruits) # number of character
odd_letters <- (name_lengths %% 2 == 1)
odd_letters

# which word has first letter of b?
first_letter_b <- (substr(fruits, start = 1, stop = 1) == "b") # substring a char vector
```

```
first_letter_b
```

Vector Recylcing

- When performing an operation on two or more vectors of unequal length, R will recycle elements of the shorter vector(s) to match the longest vector.
- There are no scalars in R. The smallest is a single element vector (can think of a scalar)
- R will issue a warning if the length of the longer vector is not an integral multiple of the shorter vector.

```
x <- c(2,4,6,8)
y <- c(1,3)
x+y # Element of y is recycled to 1,3,1,3
x*2 # Single element  Scalar 1 is recycled to 2,2,2,2
x+c(1,3,5) # Error message: longer object length is not a multiple of shorter object length
```

- Vector-Wise Math: Some functions operate on an entire vector and return *one number* rather than working element-wise:
  - For example: `sum()`,`min()`, `max()`, `mean()`, `sd()`, `var()`

```
a <- 1:10
sum(a); mean(a); sd(a)

# Standardizing Data
z <- (a -mean(a)) / sd(a)
round(z,2)

# Alternatively, use `scale()`
z2 <- scale(a)
round(z2,2)
```

Relational operators

Operator	Description
`<`	Less than
`>`	Greater than
`<=`	Less than or equal to
`>=`	Greater than or equal to
`==`	Equal to
`!=`	Not equal to

Examples:

```
x <- 2; y <- 15
x < y
x > y
x <= 2
y >= 16
y == 15
x != 2
```

```r
fruits <- c("apple", "banana", "cherry")
# which word has 6 or more letters?
name_lengths <- nchar(fruits) # number of character
name_lengths >= 6
mean(name_lengths >= 6) # proportion of name lengths greater than or equal to 6
```

Logical operators

Operator	Description
`!`	Logical NOT
`&`	Element-wise logical AND
`&&`	Logical AND
`|`	Element-wise logical OR
`||`	Logical OR

- Operators `&` and `|` perform element-wise operation producing result having length of the longer operand.
- Operators `&&` and `||` examines only the first element of the operands resulting into a single length logical vector.
- Zero is considered `FALSE` and non-zero numbers are taken as `TRUE`.

Examples:

```r
x <- c(TRUE,FALSE,0,6)
y <- c(FALSE,TRUE,FALSE,TRUE)
!x
x&y
x&&y
x|y
x||y
```

Available data sets in R

Use `data()` to view all available dataset. Call the data set by its name directly. For example: `ChickWeight`,`iris`

Subsetting Vectors

- Passing a single index or vector of entries to **keep**:

```r
fruits[c(1,3)]
```

- Passing a single index or vector of entries to **drop**:

```r
fruits[-c(1,3)]
```

- Passing a logical vector (`TRUE`=keep, `FALSE`=drop):

```r
fruits[odd_letters|first_letter_b]

color = c("red", "yellow", "red") # create a color vector corresponds to fruits
fruits[color != "yellow"] # != is "not equal to"
```

- `%in%` avoids typing a lot of logical ORs (`|`):

```
fruits %in% c(fruits[color=="red"])
```

- `which()` returns the indices of `TRUE`s in a logical vector:

```
which(fruits %in% c(fruits[color=="red"]))
```

Practice

1. Create a variable called 'x' and store the number from 0 to 5. Check the features of x.

```
x <- 0:5
# alternatively you can use seq(). Use `? function` to look up the function
x <- seq(5) # does not contain 0
x <- seq(0,5)
x
class(x); typeof(x); length(x); attributes(x); str(x)

# convert to numeric
y <- as.numeric(x)
y
class(y); typeof(y)

# convert to character
z <- as.character(x)
z
class(z); typeof(z)

# convert to logical
l <- as.logical(x)
l
class(l); typeof(l)
```

2. Create an empty numeric vector with 6 elements and store it as 'a'. Then

- update 'a' to 6 consecutive even numbers start from 2.
- name this vector using consecutive letters start from a

```
a <- vector("numeric", length=6) # create an empty numeric vector
a <- numeric(6) # alternatively

a <- c(2,4,6,8,10,12)
names(a) <- letters[1:6]
typeof(a); str(a)

a <- seq(from = 2, to = 12, by = 2)
a <- seq(2,12,2) # alternative
length(a); str(a)
```

3. Guess the type of following vector: `b <- c(2.2, "a")`, `c <- c(FALSE, 1)`, `d <- c("a", FALSE)`

```
b <- c(2.2, "a")
typeof(b)

c <- c(FALSE, 1)
typeof(c)
```

```
d <- c("a", FALSE)
typeof(d)

# R will convert all the elements to the same type. This is "coercion". When R converts the mode of sto
# This may cause unwanted behavior.
```

4. Create a 2x2 identity matrix and store it to 'I'. Create another 2x3 matrix with first row of 1,3,5 and second row of 2,4,6 and store it to 'M'. What is the element of 'M' at 2,3 position?

```
I <- matrix(
  c(1,0,0,1),
  nrow=2,
  ncol=2
)

I <- diag(2) # alternative way

M <- matrix(1:6, nrow=2, ncol=3)
# alternative way
M <- 1:6
dim(M) <- c(2,3)
M
# alternative way, using `rbinc()` and `cbind()`
x <- seq(1,5,2)
y <- seq(2,6,2)
rbind(x,y)

# Use `byrow` argument to specify how the matrix is filled
matrix(1:6, nrow=2, ncol=3, byrow=FALSE)

# Using single sqaure brackets to access to the element. It is always "row" then "column".
M[2.3]
```

5. Create a list called list1 containing 4 elements: 1) an integer vector of 1,2,3; 2) an character "Pharmacometrix"; 3) a function `sin`; 4) dataset: iris. What is the object type of each element?

```
list1 <- list(c(1L,2L,3L), "Pharmacometrix", sin, iris)
list1
str(list1)

a <- list1[1]
# a + 3
class(list1[1])

# The content of elements of a list can be retrieved by using double square brackets
b <- list1[[1]]
b + 3
class(list1[[1]])

# Name each element is the list
names(list1) <- c("integer vector", "character", "functions", "dataset")
list1

# alternatively, you can add the name while creating the list
```

```r
list2 <- list(`integer vector` = c(1L,2L,3L),
              character = "Pharmacometrix",
              functions = sin,
              dataset = iris)
list2

# After adding a name for each element in the list, you can access by using `$` notation
chr1 <- list1$character
chr1
```

6. Create a dataframe for 10 subjects with the following columns and values:
   - ID: 1 to 10
   - SEX: randomly select “male”, “female”, NA. Make this column a factor.
   - AGE: follow a normal distribution with mean 60 and standard deviation of 10

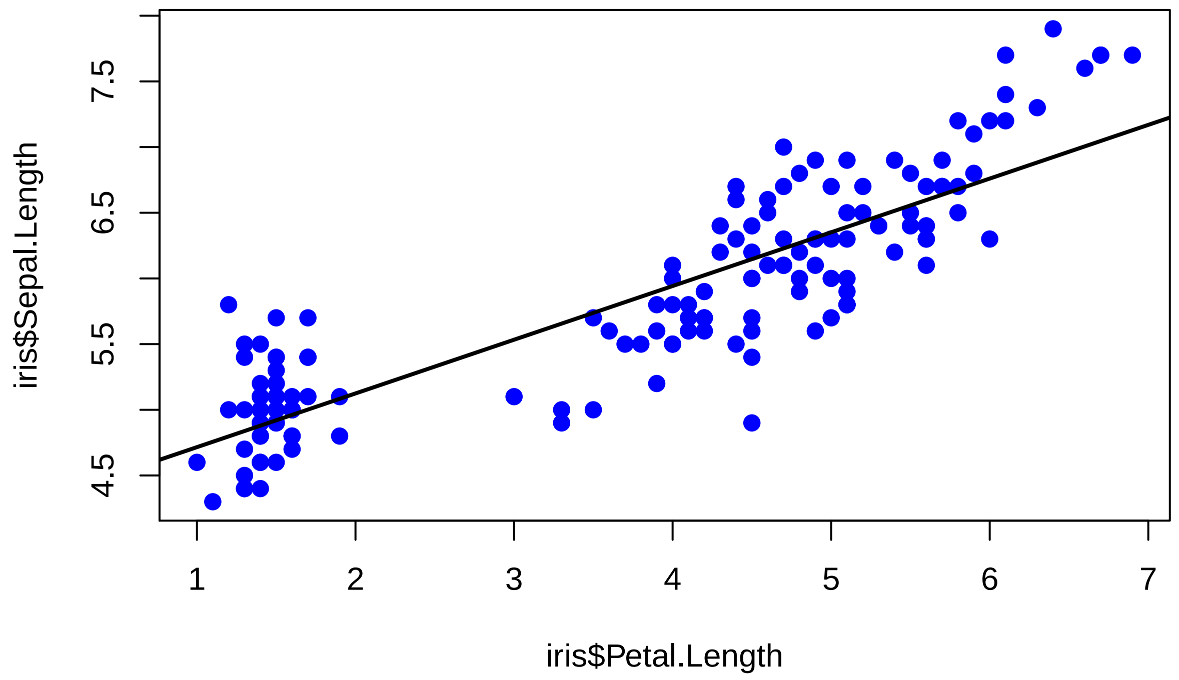
What is the object type for each column? How many missing values?

```r
set.seed(1) # to create a reproducible result
df <- data.frame(
  ID = 1:10,
  SEX = as.factor(sample(c("male", "female", NA), size=10, replace=TRUE)), # create SEX as a factor
  AGE = round(rnorm(10, mean = 60, sd = 10),0) # round the age to a whole number
)

# Check the object type of this dataframe
str(df); dim(df); nrow(df); ncol(df); names(df)

# Check if there is any missing values
anyNA(df)

# Check which is missing
is.na(df)
df$SEX %in% NA # check vector is NA

# Count how many missing values
sum(as.numeric(is.na(df)))

# Using apply function, same as using a loop (will cover in the next session)
sapply(df, function(x) sum(is.na(x)))

apply(df, MARGIN = 2, function(x) sum(is.na(x)))
# count the NAs row-wise (case by case)
apply(df, MARGIN = 1, function(x) sum(is.na(x)))
```

Some Tips for Accessing Model Output

We want to fit a linear regression model to predict iris sepal length by using its petal length. First, we take a look at the relationship between mpg and cyl:

```r
plot(iris$Sepal.Length~iris$Petal.Length, pch = 16, cex = 1.2, col = "blue")
abline(lm(Sepal.Length ~ Petal.Length, data=iris), lwd = 2)
```



We fit the model:

```
regress_model <- lm(Sepal.Length ~ Petal.Length, data=iris)
class(regress_model); typeof(regress_model); attributes(regress_model)

regress_model$coefficients
regress_model[["coefficients"]]
petal_beta <- regress_model[["coefficients"]][["Petal.Length"]]
petal_beta

# retrieve standard errrors
summary(regress_model)
summary(regress_model)[["coefficients"]]
petal_SE <- summary(regress_model)[["coefficients"]]["Petal.Length","Std. Error"]
petal_SE

# calculate 95% CI
petal_CI <- petal_beta + c(-qnorm(0.975), qnorm(0.975))*petal_SE
names(petal_CI) <- c("lower", "upper")
petal_CI
```

Then generate report directly from the model without copy and paste the model values.

```
# In a Markdown document:
A 1 unit increase in petal length is associated with a `r round(petal_beta, 2)` unit increase in septal
                                    `r round(petal_CI["upper"],2)`))
```

A 1 unit increase in petal length is associated with a 0.41 unit increase in septal length (95% CI: (0.37, 0.45))

Other Practices: `swirl`

Interactive R tutorials for beginner, intermediate and even advanced learners. More details can be found here

To set up `swirl`:

1. `install.packages("swirl")`
2. `library(swirl)`
3. `swirl()`
4. Choose `R Programming`, pick a tutorial, and follow directions
5. To get out of `swirl`, type `bye()` in the middle of a lesson, or `0` in the menus

Further details

Resources:

1. R Tutorial https://www.tutorialspoint.com/r/index.htm
2. R for Data Science by Grlemund and Wickham (2017), online at https://r4ds.had.co.nz/
3. R cheetsheets https://rstudio.com/resources/cheatsheets/
4. Carpentry Programming with R https://swcarpentry.github.io/r-novice-inflammation/

Session Info:

```
sessionInfo()
```

```
## R version 3.6.1 (2019-07-05)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 18363)
##
## Matrix products: default
##
## locale:
## [1] LC_COLLATE=English_United States.1252
## [2] LC_CTYPE=English_United States.1252
## [3] LC_MONETARY=English_United States.1252
## [4] LC_NUMERIC=C
## [5] LC_TIME=English_United States.1252
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## loaded via a namespace (and not attached):
##  [1] compiler_3.6.1  magrittr_1.5    tools_3.6.1     htmltools_0.4.0
##  [5] yaml_2.2.1      Rcpp_1.0.4.6    stringi_1.4.6   rmarkdown_1.16
##  [9] knitr_1.28      stringr_1.4.0   xfun_0.14       digest_0.6.25
## [13] rlang_0.4.6     evaluate_0.14
```