F19 STA 100 A01 Discussion 01

Yishan Huang

2019/9/29

Discussion Time: Tuesday 8:00 – 8:50 am, Haring Hall 1204.

Notes: https://github.com/Hahahuo-13316/sta100-a01-fall19

Office Hour: Tuesday 2:00 – 3:00 pm, Earth and Physical Science Building 1317.

Getting Started

Install R

The latest version 3.6.1 of R can be found on its website https://www.r-project.org/. Click on **CRAN** on the left, and choose an appropriate mirror to download the install package. Follow the instruction and complete the installation.

Choose an IDE

There are several IDE (integrated development environment) for R, one of the popular IDE is RStudio. In this discussion I will always use RStudio to present and run the codes.

To install RStudio, go to their website https://rstudio.com/ and download the latest version. After installing that, you could go to preferences – appearance to adjust the font, font size, theme, etc. of IDE.

In this discussion I will also use **R Markdown**, which is a tool in RStudio that could create HTML or PDF documents in which you can put codes, outputs and LaTeX formulas.

When you need some help

You could use the documentation, which could be downloaded on https://cran.r-project.org/manuals.html. Or, just Google it to look for the command you need, then type **?+command** in the console (of your IDE) to know more about a specific command.

```
?sum
?lm
```

R Introduction

Basics

$x^2 + y^2$ You could use # to make comments on your R code:

```
# put # on a line to create a comment
```

Assign values to variables. Notice that '<-' and '=' have the same meaning, but '<-' is recommanded in R. Notice that for commands in console, 'print' could be omitted.

```
x <- 2
x
```

```
## [1] 2
```

```
y = 4
y
```

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

```
print(y)
```

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

Let's go through some basic operations:

```
x + y          #addition
```

```
## [1] 6
```

```
x - y          #subtraction
```

```
## [1] -2
```

```
-x             #negation
```

```
## [1] -2
```

```
x * y          #multiplication
```

```
## [1] 8
```

```
x / y          #division
```

```
## [1] 0.5
```

```
x^y            #powers
```

```
## [1] 16
```

```
log(x)         #natural log
```

```
## [1] 0.6931472
```

```
exp(x)         #exponentiation
```

```
## [1] 7.389056
```

```
sqrt(x)        #square root
```

```
## [1] 1.414214
```

```
x == 4         #equality
```

```
## [1] FALSE
```

```
x >= 4         #greater than or equal
```

```
## [1] FALSE
```

```
x > 4          #greater than
```

```
## [1] FALSE
```

```
x != 4         #not equals
```

```
## [1] TRUE
```

```
!TRUE          #logical negation
```

```
## [1] FALSE
```

Vectors

Use c(...) to create a vector:

```
v <- c(1, 2, 3, 4, 5, 6)
u <- c(7, 10, 15, 30, 40, 45)
u
```

```
## [1]  7 10 15 30 40 45
```

```
v
```

```
## [1] 1 2 3 4 5 6
```

Basic operations for vectors are **element-wise**:

```
u + v
```

```
## [1]  8 12 18 34 45 51
```

```
u - v
```

```
## [1]  6  8 12 26 35 39
```

```
-u
```

```
## [1]  -7 -10 -15 -30 -40 -45
```

```
u * v
```

```
## [1]   7  20  45 120 200 270
```

```
u / v
```

```
## [1] 7.0 5.0 5.0 7.5 8.0 7.5
```

```
u^v
```

```
## [1]          7        100       3375     810000  102400000 8303765625
```

```
log(v)
```

```
## [1] 0.0000000 0.6931472 1.0986123 1.3862944 1.6094379 1.7917595
```

```
exp(v)
```

```
## [1]   2.718282   7.389056  20.085537  54.598150 148.413159 403.428793
```

And also there are operations between vector and scalar:

```
v - 1
```

```
## [1] 0 1 2 3 4 5
```

```
1 / v
```

```
## [1] 1.0000000 0.5000000 0.3333333 0.2500000 0.2000000 0.1666667
```

```
v^2
```

```
## [1]  1  4  9 16 25 36
```

```
2^v
```

```
## [1]  2  4  8 16 32 64
```

Access specific element in a vector using index, or get slices. Notice that **the index begin with 1**.

```
u[1]
```

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

```
u[1:3]
```

```
## [1]  7 10 15
```

Statistical discription of vectors

```
x <- c(1, 1, 1, 2, 2, 2)
sum(x)        #sum
```

```
## [1] 9
```

```
length(x)       #number of entries
```

```
## [1] 6
```

```
mean(x)         #mean
```

```
## [1] 1.5
```

```
sum(x) / length(x) == mean(x)
```

```
## [1] TRUE
```

Random number and random sample

```
runif(10, min = 0, max = 100)
```

```
##  [1] 77.28468 36.25120 20.27240 75.63445 91.69110 58.23028 30.58875
##  [8] 63.18228 29.16440 41.91961
```

```
sample(1:1000, size = 12)
```

```
##  [1] 755 802 291 663  34 436 461 955 427 448 344 712
```

Flow control

There are for and while loops in R:

```
for(i in 1:10) {
  print(i)
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
```

```
## [1] 10
```

```
i <- 1
while(i <= 10) {
  print(i)
  i <- i + 1
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
## [1] 10
```

Important notice: in R, do not use loop if there exists corresponding vectorised methods. For example, it is much slower to run the following code

```
system.time({
  s <- 0
  for(i in 1:123456789) {
    s <- s + i
  }
  s
})
```

```
##    user  system elapsed
##   2.596   0.005   2.611
```

than simply use 'sum'.

```
system.time({sum(1:123456789)})
```

```
##    user  system elapsed
##       0       0       0
```