BB503/BB602 - R Training - Week III

Ege Ulgen

R tips

- If you can't figure out how to solve an issue, Google is your friend. e.g., "how to calculate mode r"
- If you need help with the usage of a function, type `?function_name`. e.g. `?quantile`
- If you get an error, and cannot fix it. C/P the error into Google. Someone else most likely had a similar problem
- Some resources for learning the basic syntax of R;
  - Codecademy - https://www.codecademy.com/learn/learn-r
  - RStudio Cloud Primers - https://rstudio.cloud/learn/primers
  - Dataquest - https://www.dataquest.io/course/introduction-to-data-analysis-in-r/
  - R for Data Science - https://r4ds.had.co.nz/index.html
- Interesting read: https://www.dataquest.io/blog/learn-r-for-data-science/
- Some resources you might want to follow:
  - R-Bloggers - https://www.r-bloggers.com/
  - STHDA - http://www.sthda.com/english/
  - Quick-R - https://www.statmethods.net/

R Basics

Conditional Statements

```
### simple conditional (Boolean) statements
3 == 3
```

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

```
3 == 4
```

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

```
3 != 4
```

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

```
3 > 4
```

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

```
3 <= 4
```

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

```
5 %% 3 == 0
```

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

```
5 %% 3 == 2
```

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

```
### multiple conditions
# and
3 == 3 & 3 > 4
```

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

```
# or
3 == 3 | 3 > 4
```

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

```
# > 2 conditions
3 == 3 | (3 > 4 & 5 != 6)
```

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

```
!TRUE
```

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

We can use these conditions to selectively execute commands when these condition(s) are met:

```
# after a complex calculation X becomes 3
x <- 3

if (x == 3) {
    print("x is equal to 3")
    # other commands that should be executed if the statement is true go here
}
```

```
## [1] "x is equal to 3"
```

Using `else if` we can add even more conditions:

```
# after a complex calculation X becomes 5
x <- 5

if (x == 3) {
    print("x is equal to 3")
    # other commands that should be executed if this first statement is true go here
} else if (x > 3) {
    print("x is greater than 3")
    # other commands that should be executed if this second statement is true go here
}
```

```
## [1] "x is greater than 3"
```

Finally, using else we can execute any commands if none of the conditions are met:

```
# after a complex calculation X becomes 1
x <- 1

if (x == 3) {
    print("x is equal to 3")
```

```
    # other commands that should be executed if this first statement is true go here
} else if (x > 3) {
    print("x is greater than 3")
    # other commands that should be executed if this second statement is true go here
} else {
    print("x is less than 3")
    # other commands that should be executed if no condition is true go here
}
```

```
## [1] "x is less than 3"
```

Loops

We may wish to execute certain commands repeatedly until a certain condition is met. For this purpose, we can use a `while` loop:

```
# initialize
current_number <- 1
while(current_number <= 10) {
    # execute any commands
    print(paste("Current #:", current_number))
    # update
    current_number <- current_number + 1
}
```

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

We may wish to execute certain commands repeatedly for a set amount of times. For this purpose, we can use a `for` loop:

```
for (i in 1:10) {
    # execute any commands
    print(paste("Current #:", i))
}
```

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

Putting it all together - The Fizz Buzz Game

Fizz buzz is a group word game for children to teach them about division. Players take turns to count incrementally,

- replacing any number divisible by 3 with the word "fizz"
- replacing any number divisible by 5 with the word "buzz"

```
for (i in 1:100) {
    if (i %% 3 == 0) {
        print("Fizz")
    } else if (i %% 5 == 0) {
        print("Buzz")
    } else {
        print(i)
    }
}
```

```
## [1] 1
## [1] 2
## [1] "Fizz"
## [1] 4
## [1] "Buzz"
## [1] "Fizz"
## [1] 7
## [1] 8
## [1] "Fizz"
## [1] "Buzz"
## [1] 11
## [1] "Fizz"
## [1] 13
## [1] 14
## [1] "Fizz"
## [1] 16
## [1] 17
## [1] "Fizz"
## [1] 19
## [1] "Buzz"
## [1] "Fizz"
## [1] 22
## [1] 23
## [1] "Fizz"
## [1] "Buzz"
## [1] 26
## [1] "Fizz"
## [1] 28
## [1] 29
## [1] "Fizz"
## [1] 31
## [1] 32
## [1] "Fizz"
## [1] 34
## [1] "Buzz"
## [1] "Fizz"
## [1] 37
```

```
## [1] 38
## [1] "Fizz"
## [1] "Buzz"
## [1] 41
## [1] "Fizz"
## [1] 43
## [1] 44
## [1] "Fizz"
## [1] 46
## [1] 47
## [1] "Fizz"
## [1] 49
## [1] "Buzz"
## [1] "Fizz"
## [1] 52
## [1] 53
## [1] "Fizz"
## [1] "Buzz"
## [1] 56
## [1] "Fizz"
## [1] 58
## [1] 59
## [1] "Fizz"
## [1] 61
## [1] 62
## [1] "Fizz"
## [1] 64
## [1] "Buzz"
## [1] "Fizz"
## [1] 67
## [1] 68
## [1] "Fizz"
## [1] "Buzz"
## [1] 71
## [1] "Fizz"
## [1] 73
## [1] 74
## [1] "Fizz"
## [1] 76
## [1] 77
## [1] "Fizz"
## [1] 79
## [1] "Buzz"
## [1] "Fizz"
## [1] 82
## [1] 83
## [1] "Fizz"
## [1] "Buzz"
## [1] 86
## [1] "Fizz"
## [1] 88
## [1] 89
## [1] "Fizz"
## [1] 91
```

```
## [1] 92
## [1] "Fizz"
## [1] 94
## [1] "Buzz"
## [1] "Fizz"
## [1] 97
## [1] 98
## [1] "Fizz"
## [1] "Buzz"
```

For a variation of the game, any number divisible by both 3 and 5 is replaced with the word "fizzbuzz"

```
for (i in 1:100) {
    if (i %% 3 == 0 & i %% 5 == 0) {
        print("FizzBuzz")
    } else if (i %% 3 == 0) {
        print("Fizz")
    } else if (i %% 5 == 0) {
        print("Buzz")
    } else {
        print(i)
    }
}
```

```
## [1] 1
## [1] 2
## [1] "Fizz"
## [1] 4
## [1] "Buzz"
## [1] "Fizz"
## [1] 7
## [1] 8
## [1] "Fizz"
## [1] "Buzz"
## [1] 11
## [1] "Fizz"
## [1] 13
## [1] 14
## [1] "FizzBuzz"
## [1] 16
## [1] 17
## [1] "Fizz"
## [1] 19
## [1] "Buzz"
## [1] "Fizz"
## [1] 22
## [1] 23
## [1] "Fizz"
## [1] "Buzz"
## [1] 26
## [1] "Fizz"
## [1] 28
## [1] 29
## [1] "FizzBuzz"
## [1] 31
```

```
## [1] 32
## [1] "Fizz"
## [1] 34
## [1] "Buzz"
## [1] "Fizz"
## [1] 37
## [1] 38
## [1] "Fizz"
## [1] "Buzz"
## [1] 41
## [1] "Fizz"
## [1] 43
## [1] 44
## [1] "FizzBuzz"
## [1] 46
## [1] 47
## [1] "Fizz"
## [1] 49
## [1] "Buzz"
## [1] "Fizz"
## [1] 52
## [1] 53
## [1] "Fizz"
## [1] "Buzz"
## [1] 56
## [1] "Fizz"
## [1] 58
## [1] 59
## [1] "FizzBuzz"
## [1] 61
## [1] 62
## [1] "Fizz"
## [1] 64
## [1] "Buzz"
## [1] "Fizz"
## [1] 67
## [1] 68
## [1] "Fizz"
## [1] "Buzz"
## [1] 71
## [1] "Fizz"
## [1] 73
## [1] 74
## [1] "FizzBuzz"
## [1] 76
## [1] 77
## [1] "Fizz"
## [1] 79
## [1] "Buzz"
## [1] "Fizz"
## [1] 82
## [1] 83
## [1] "Fizz"
## [1] "Buzz"
```

```
## [1] 86
## [1] "Fizz"
## [1] 88
## [1] 89
## [1] "FizzBuzz"
## [1] 91
## [1] 92
## [1] "Fizz"
## [1] 94
## [1] "Buzz"
## [1] "Fizz"
## [1] 97
## [1] 98
## [1] "Fizz"
## [1] "Buzz"
```

Functions

Functions are modules of code that accomplish a specific task. They take in data, process it, and return a result.

The exponentiation function $f(x, y) = x^y$ can be coded in R as:

```
pow <- function(x, y) {
    output <- x ^ y
    return(output)
}
```

E.g., $f(2, 3) = 2^3 = 8$:

```
pow(2, 3)
```

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

Here,

- `pow` is the name of the function, and it is arbitrary
- `function(...)` is for defining any function in R
- `x` and `y` are input names (arguments), and names are arbitrary
- `output` is the object that stores the result (not created globally)
- `return` is the function for outputting the result

We may define default values for the arguments:

```
pow2 <- function(x, y = 2) {
    output <- x ^ y
    return(output)
}
pow2(3)
```

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

```
pow2(3, 3)
```

```
## [1] 27
```

Another Solution for Fizz-Buzz

```
fizz_buzz <- function(number) {
    if (number %% 3 == 0 & number %% 5 == 0) {
        return("FizzBuzz")
    }

    if (number %% 3 == 0) {
        return("Fizz")
    }

    if (number %% 5 == 0) {
        return("Buzz")
    }
    return(number)
}

fizz_buzz(2)
```

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

```
fizz_buzz(3)
```

```
## [1] "Fizz"
```

```
fizz_buzz(5)
```

```
## [1] "Buzz"
```

```
fizz_buzz(15)
```

```
## [1] "FizzBuzz"
```

```
sapply(1:100, fizz_buzz)
```

```
##   [1] "1"        "2"        "Fizz"     "4"        "Buzz"     "Fizz"
##   [7] "7"        "8"        "Fizz"     "Buzz"     "11"       "Fizz"
##  [13] "13"       "14"       "FizzBuzz" "16"       "17"       "Fizz"
##  [19] "19"       "Buzz"     "Fizz"     "22"       "23"       "Fizz"
##  [25] "Buzz"     "26"       "Fizz"     "28"       "29"       "FizzBuzz"
##  [31] "31"       "32"       "Fizz"     "34"       "Buzz"     "Fizz"
##  [37] "37"       "38"       "Fizz"     "Buzz"     "41"       "Fizz"
##  [43] "43"       "44"       "FizzBuzz" "46"       "47"       "Fizz"
##  [49] "49"       "Buzz"     "Fizz"     "52"       "53"       "Fizz"
##  [55] "Buzz"     "56"       "Fizz"     "58"       "59"       "FizzBuzz"
##  [61] "61"       "62"       "Fizz"     "64"       "Buzz"     "Fizz"
##  [67] "67"       "68"       "Fizz"     "Buzz"     "71"       "Fizz"
##  [73] "73"       "74"       "FizzBuzz" "76"       "77"       "Fizz"
##  [79] "79"       "Buzz"     "Fizz"     "82"       "83"       "Fizz"
##  [85] "Buzz"     "86"       "Fizz"     "88"       "89"       "FizzBuzz"
##  [91] "91"       "92"       "Fizz"     "94"       "Buzz"     "Fizz"
##  [97] "97"       "98"       "Fizz"     "Buzz"
```

```
?sapply
```

The apply family of functions

```
mat <- matrix(rnorm(30), nrow = 5, ncol = 6)
sample_types <- c("Tumor", "Tumor", "Tumor", "Control", "Control")

?apply
# apply mean to each row
apply(mat, 1, mean)
```

```
## [1] -0.65008 -0.22373  0.64550  0.76677  0.67384
```

```
# apply sd to each column
apply(mat, 2, sd)
```

```
## [1] 0.99292 0.99008 1.13726 0.68687 0.76416 1.25498
```

```
# Apply a function over a list or vector
sapply(mat[1, ], function(x) x^3)
```

```
## [1] -1.8078965  0.3124814 -8.6741200 -0.0023680  0.0012509 -2.1002694
```

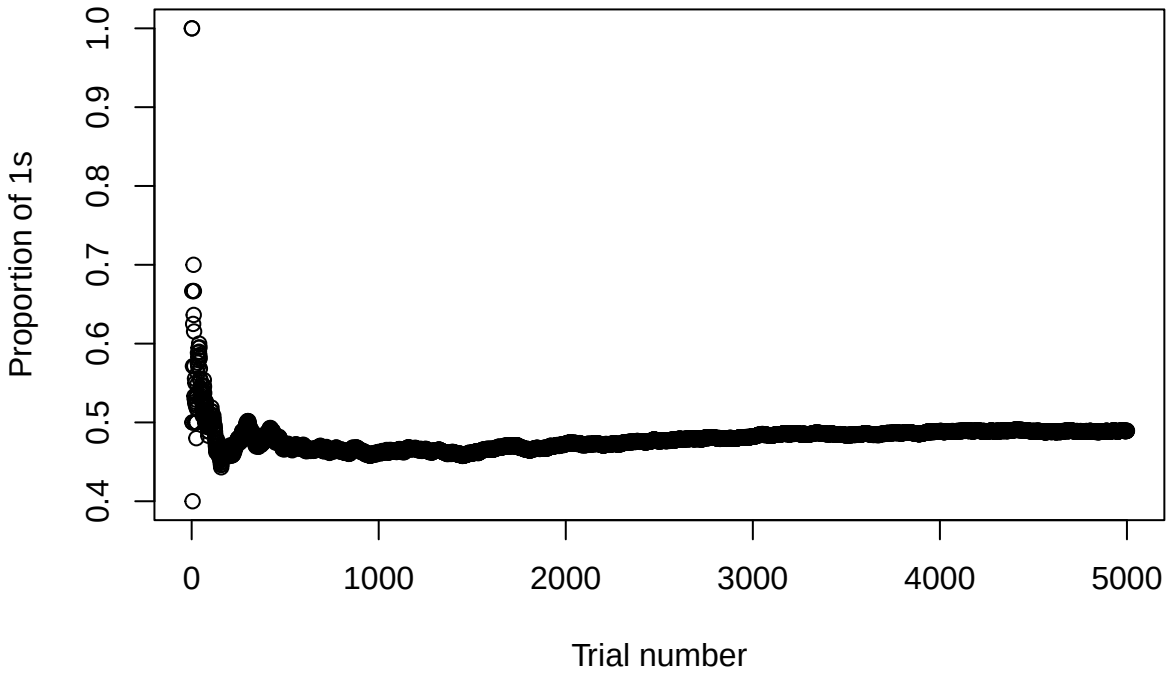
```
# Apply a function over a ragged array
# mean values of 1st variable per sample type
tapply(mat[, 1], sample_types, mean)
```

```
##  Control    Tumor
##  0.71216 -0.43408
```

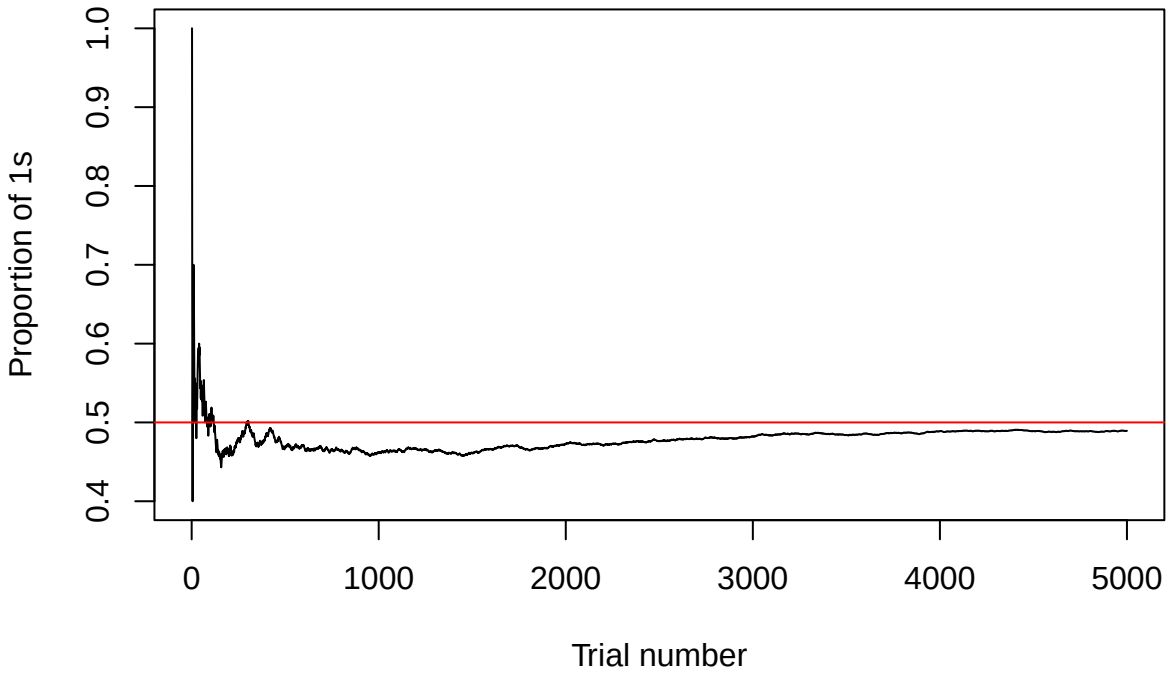
Probability

Basic Simulation - Flipping a fair coin

```
# by this you get the same answer all the time
set.seed(321)

# number of trials
n <- 5000

# tossing coins
results <- c() # to store results
average_at_i <- c() # to store averages (will approximate 0.5)
for (i in 1:n) {
    current_res <- sample(c(0, 1), 1)
    results <- c(results, current_res)
    average_at_i <- c(average_at_i, mean(results))
}

# plot resulting averages by each trial
plot(average_at_i, xlab = "Trial number", ylab = "Proportion of 1s")
```




```
plot(average_at_i, xlab = "Trial number", ylab = "Proportion of 1s", type = "l")
abline(h = 0.5, col = "red")
```


Proportion of 1s
0.4
0.5
0.6
0.7
0.8
0.9
1.0
0
1000
2000
3000
4000
5000
Trial number