Introduction to Statistical Programming with R

Discussion Section A, B

TA: Feng Gao (Pronunciation in English: Fung Gao)
Email: f.gao@ucla.edu

Office Hour

Wednesday 8:30 a.m. - 9:30 a.m.

Friday 8:30 a.m. - 9:30 a.m.

Introduction

Goal of the Class:

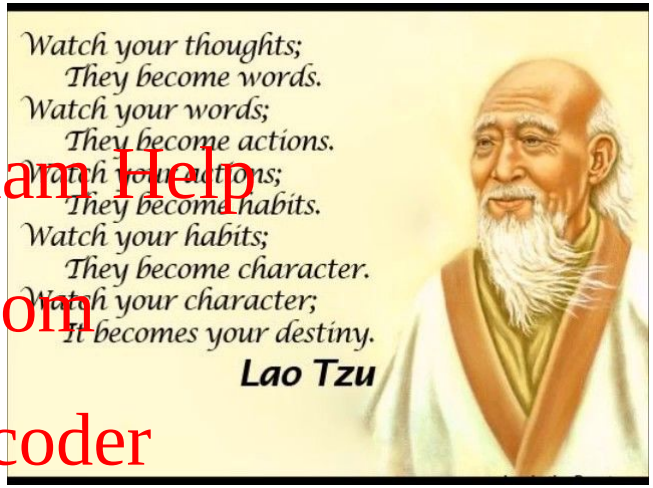
Learn basic R programming ideas and skills for further study in Statistics.

Goal of the Discussion and Office Hour:

Resolve the questions.

Note:

- I am here to help you instead of torturing you.
- Think beyond the code itself.



Give a man a fish, and you feed him for a day. Teach a man to fish, and you feed him for a lifetime. - Lao Tzu

Programming Language: An Analogy

Could you please .....?

Codes

Install R and RStudio

R:

https://www.r-project.org/

RStudio:

https://rstudio.com/

Install the Required Packages

Link:

https://github.com/elmstedt/UCLAstats20/

Command to run in RStudio Console:

```
install.packages("devtools")
library(devtools)
install_github("elmstedt/UCLAstats20")
```

Get Access to the Interactive Notes

Every time you relaunch R:


```
library(UCLAStats20)
notes(#)
```

Code Style

https://style.tidyverse.org/index.html

10/13 Discussion

Vectors in R

TA: Feng Gao

Email: f.gao@ucla.edu

Clean Up

- Any questions for the materials in the last homework?
- Any suggestions?
- My suggestions:
  - Try to code and run it first. Don't just keep the idea in your mind. Write down the code first and see what will happen.
  - Make sure you leverage all the available materials, e.g. CCLE, interactive notes, Campuswire.
  - Ask question after you tried. Don't be hand-waving.

Vectors in R

Everything in R is vector!

Differences between Vector and Set

Vector:

Order.
Each element is not necessary to be unique.

Set:

No order.
Elements have to be unique.

Create Vectors and Mode Hierarchy

Create a Vector:

```
c()
```


Mode Hierarchy:


```
logical < numeric <
character
```

More on Mode Hierarchy

- Combining logical and numeric vectors will result in a numeric vector.
- Combining numeric and character vectors will result in a character vector.
- Combining logical and character vectors will result in a character vector.
- Combining logical, numeric, and character vectors will result in a character vector.

Some Examples

```
fib <- c(1, 1, 2, 3, 5, 8, 13)
parks <- c("Leslie", "April", "Ron", "Tom", "Donna",
"Jerry")
true_dat <- c(TRUE, FALSE, TRUE, T, F)
```


- mode(c(fib, parks))
- mode(c(fib, true_dat))
- mode(c(parks, true_dat))
- mode(c(fib, parks, true_dat))

Sequences and Repeated Patterns

Create a sequence:

```
seq(from, to, by)
seq(from, to, length.out)
```


Repeat:


```
rep(x, times = 1, length.out = NA, each = 1)
rep(c(1,2,3), 2)
```

The Colon :

Functions the same as Seq():

```
-2:5 # same as seq(-2, 5)
pi:10 # same as seq(pi, 10)
```


What are the differences?:

```
1:n - 1
1:(n - 1)
```

The seq_len() and seq_along() Function

seq_len():

The seq_len() function inputs a single length.out argument and generates the sequence of integers

seq_along():

The seq_along() function inputs a single along.with argument and generates the sequence of integers 1, 2, ..., length(along.with). In conclusion, it returns index(s).

Function rep()

The rep() function creates a vector of repeated values. The first argument, generically called x, is the vector of values we want to repeat. The second argument is the times argument that specifies how many times we want to repeat the values in the x vector.

What are the returns and why?:


```
rep(seq(2, 20, by = 2), 2)
rep(seq(2, 20, by = 2), rep(2, 10))
```

Extracting and Assigning Vector Elements

Extracting, what the the returns?:


```
running_times[5]
running_times[c(3, 7)]
running_times[4:8]
running_times[-4]
running_times[-c(1, 5)]
running_times[c(1:4)]
running_times[c(-1, 3)]
running_times[1.9]
running_times[-1.9]
running_times[0.5]
```

Extracting and Assigning Vector Elements

Assigning, what the the returns?:

```
running_times[8] <- 43
running_times[9:10] <- c(42, 37)
bad[1:2] <- c(14, 8)
```

Vector Arithmetic

What are the values?:

```
x <- c(1, 3, 5)
y <- c(2, 4, 3)
x + y
x * y
x^y
```

Vector Arithmetic

Recycling:

When applying arithmetic operations to two vectors of different lengths, R will automatically recycle, or repeat, the shorter vector until it is long enough to match the longer vector.


```
c(1, 3, 5) + c(5, 7, 0, 2, 9, 11)
c(1, 3, 5) + 5
c(1, 3, 5) + c(5, 7, 0, 2, 9)
```

Vectorization

Suppose we have a function that we want to apply to all elements of a vector. In many cases, functions in R are vectorized, which means that applying a function to a vector will automatically apply the function to each individual element in the vector.

The vapply() Function

Usage

```
vapply(.x, .f, fun_value, ..., use_names = TRUE)
```


Arguments

.x	A vector.
.f	A function to be applied.
fun_value	A (generalized) vector; a template for the return value from .f.

Built-in Functions

- `sum(x)` computes the sum of the values of `x`
- `mean(x)` computes the mean of `x`
- `sd(x)` computes the standard deviation of `x`
- `var(x)` computes the variance of `x`
- `median(x)` computes the median of `x`
- `IQR(x)` computes the interquartile range of `x`
- `min(x)` computes the minimum value of `x`
- `max(x)` computes the maximum value of `x`
- `range(x)` computes the range (difference between the min and max) of `x`
- `diff(x)` computes consecutive differences of `x`
- `cumsum(x)` computes the cumulative sum of `x`
- `sort(x)` orders the values of `x` (increasing order by default)
- `fivenum(x)` computes the five-number summary of `x`
- `summary(x)` computes a few summary statistics of `x`

Special Values

NA:

```
running_times <- c(running_times[1:5], NA, running_times[6:10])
mean(running_times)
```

NULL:

```
nada <- NULL
mode(nada)
```

NaN:

```
0 / 0
```

Inf:

```
1 / 0
```

10/20 Discussion

Logic Expression & Control Flow

TA: Feng Gao

Email: f.gao@ucla.edu

Clean Up

- Any questions for the materials in the last homework?
- Any suggestions?
- I don’t have the solutions either.
- My suggestions
  - Don’t focus on interpreting the hints. It is not helpful beyond the HW.
  - Please concentrate on the problem itself.
  - Using the learned materials in the class as the building blocks.

Relational Operators

Binary Operators that returns TRUE or FALSE

A list of the relational operators in R is below:

- < : Less than
- > : Greater than
- <= : Less than or equal to
- >= : Greater than or equal to
- == : Equal to
- != : Not equal to

Cautions on == and !=


```
49 * (4 / 49) == 4 # Is 49 * (4 / 49) exactly equal to 4?
```

Relational Operators: Vectorization


```
c(3, 8) >= 3
c(1, 4, 9) == 9
c(3, 8) < c(1, 4)
c(1, 4, 9, 3, 8) > c(5, 6, 7)
```

The any(), all(), and identical() Functions

```
any()
```

```
all()
```

```
identical(c(1,2,3,4), 1:4)
```

Special Values

NA: Not Available

```
c(7, NA, 4) > 8
```

NULL: Nothing

```
c(TRUE, FALSE) > NULL
c(1,2) > NULL
```

NaN: Not a Number (implies an illegal math expression)

```
c(1, 4, 9) <= NaN
```

Inf: everything larger than a large enough threshold

```
c(1, 4, Inf) < Inf
exp(1000) == Inf
```

Logical Indexing

Expression:

```
logical_index <- (running_times %% 2) == 0
```


Extracting elements:


```
running_times[running_times > 40]
```


which() function: returns indices whose corresponding logical expression is TRUE

```
which(running_times >= 50)
```

Boolean Operators

&: AND

```
some_nums > 3 & some_nums < 7
```

|: OR

```
some_nums < 3 | some_nums > 7
```

!: NOT

```
!(some_nums < 3)
```

The && and || Operators


```
x <- -5
x < 0 | is.na(sqrt(x) > 2)
x < 0 || is.na(sqrt(x) > 2)
x < 0 && is.na(sqrt(x) > 2)
x < 0 & is.na(sqrt(x) > 2)
```


From left to right. Not necessary to compute all of them

Control Flow

Loop:

```
for(iteratable){
}
While(condition){
}
repeat{
}
```


Condition Statement:

```
if(condition){

}
else{

}
```

Some Questions

Can you implement:

```
max() and min() with if{}else{}
```


When to use:


```
for(iteratable){
}
While(condition){
}
repeat{
}
```

10/27 Discussion

Review Section

TA: Feng Gao

Email: f.gao@ucla.edu

Chapter 1 Key Points:

Basic Arithmetic: `+, -, *, /, ^`

Modular Arithmetic: `%%, %/%`

Order of Operations: See HW3, Question 6

Object Assignment: What is a legal name of an object?; Masking; Built-in Objects

Function: Syntax; What will a function return by default?; How to create/call a function?; local environment, global environment, parent environment, mask?

HW1 Intermediate Questions

Chapter 2 Key Points:

Vector: Basic data types; Mode hierarchy and Coercion; Length; `c()` function

Import functions: `seq()` function, `rep()` function, `seq_len()` function, `seq_along()` function.

Extract, Assign Vector Elements: Negative index; Multiple indices; Fractional index; Blank index;

Vector Arithmetic: Recycling; Vectorization; `vapply()` function

Built-in Functions: Chapter 2 notes, Section 6.

Special Values: See discussion slides; Floating point representation

HW2 Intermediate Questions

Chapter 3 Key Points:

Relational Operators: `<, <=, >, >=, ==, !=`; See notes for detail.

Vectorized; Recycling; `any()`, `all`, `identical()` function; Special values

Logical Indexing: Syntax; Extract elements; `which()` function

Boolean Operators: `&, |, !, &&, ||`; Differences between `&, | and &&, ||`

Chapter 4 Key Points

Loop:

Syntax of Loops. for, while, repeat.

How to write loops?

When to use?

HW3 Intermediate Questions

Chapter 5 Key Points:

Matrix: `matrix()` function; Matrix indexing; Dimension;

`cbind()` function; `rbind()` function

Naming row and columns: `colnames()`, `rownames()` function; `dimnames()` function

Extracting data from 2-D matrix: Single element; Entire row; Entire column; Remove element, row, column

Logical Indices

Entrywise Arithmetic; Matrix Arithmetic

Diagonal matrix:diag`()` function; Inverse matrix `solve()` function; `apply()` function

Tips: How to write/implement a function

1. Clarify the INPUT and the OUTPUT.
2. Think through the algorithm.

   2.1. Any Constraints?

   2.2. What is the Time Complexity? (How many times of loops)

   2.3. Logic

   2.4 Corner Cases

3. Design TEST CASES (Very important).

11/10 Discussion

Data Frames and Lists

TA: Feng Gao

Email: f.gao@ucla.edu

Questions about Midterm Exam

- Any concerns on the midterm exam?
- What do you think of the difficulty?
- If you want to ask a specific question on the midterm, please make an appointment with me.

Some Functions

```
library()

data()

head()

tail()
```

What is the output?

```
head(1:10)

head(1:10, n = 5)

head(1:10, n = -5)

tail(1:10, n = -3)
```

Data Frames: Definition

A data frame is a two-dimensional array of values.

Q: What makes it different from a matrix?

A: Each column of a data frame can be of a different type.

Create Data Frame

```
parks_df <- data.frame(

        "Name" = c("Leslie", "Ron", "April"), "Height" = c(62, 71, 66),
        "Weight" = c(115, 201, 119), "Income" = c(4000, NA, 2000) )

parks_mat <- cbind(c(62, 71, 66), c(115, 201, 119), c(4000, NA, 2000))

rownames(parks_mat) <- c("Leslie", "Ron", "April")

colnames(parks_mat) <- c("Height", "Weight", "Income")

data.frame(parks_mat)
```

Data Frame: Basic Functions

```
dim()           dim(parks_df)

rownames()      rownames(parks_df)

colnames()      colnames(parks_df)

dimnames()      dimnames(parks_df)

rbind()         rbind(parks_df, list("Leon", 74, 194, 5000))

cbind()         cbind(parks_df, "Age" = c(34, 49, 20))
```

Data Frame: Extracting Data

Single Bracket

```
parks_df[1, ]

parks_df[, -1]

parks_df[-2, 3]

parks_df[, "Name"]

parks_df[c(FALSE, FALSE, TRUE), "Income"]

parks_df[, "Name", drop = FALSE]
```

Data Frame: Extracting Data

Double Brackets

```
parks_df[[1]]
```

```
parks_df[["Height"]]
```

```
parks_df[[3]][1]
```

Data Frame: Extracting Data

$ Operator

```
parks_df$Height
```

```
parks_df$Age <- c(34, 49, 20)
```

Data Frame: Extracting Data

with() function: construct a local environment

```
with(parks_df, Height)

with(parks_df, Weight > 110)

with(parks_df, mean(Height))
```

Modes and Classes

Q: Why data in a data frame can be of different types?

A: Because they are stored as different lists. Each column is a separate vector.

Lists: Definitions

A list is an ordered collection of objects.

```
hL <- list(

1:10,

matrix(1:6, nrow = 2, ncol = 3),

parks_df,

list(1:5, matrix(1:9, nrow = 3, ncol = 3))

)
```

Lists: Functions

```
length(L)
```

```
names(L) <- c("Vector", "Matrix", "Data.Frame", "List")
```

```
list("Vector" = 1:10, "Matrix" = matrix(1:6, nrow = 2, ncol = 3))
```

Lists: Extracting Data

Brackets:

```
L[[1]]

L[1]

L[[2]][, 1]

L[[-1]]

L[[c(4, 2)]]  L[[4]][[2]]

L[[c(4, 2, 3)]]
```

Lists: Extracting Data

$ Operator

```
L$Vector
```

```
L$Matrix
```

```
L$List
```

```
L$Function <- mean
```

```
L$Function(L$Vector)
```

Vectorized Functions for Data Frames and Lists

Str() and Summary()

```
str(trees)
```

```
summary(trees$Volume)
```

```
summary(trees)
```

Vectorized Functions for Data Frames and Lists

Family of apply()

`apply()`

`lapply()`

`sapply()`

`vapply()`

11/24 Discussion

String Manipulation

TA: Feng Gao

Email: f.gao@ucla.edu

Basic Definition

What is the main difference between string and character vector?

```
s <- "I am string"

typeof(s)

length(s)
```

Some Functions

paste() function

```
paste(..., sep = " ", collapse = NULL)

paste("I", "am", "a", "string")

paste("I", "am", "a", "string", sep = ", ")

paste("UC", c("Berkeley", "Los Angeles", "Irvine"), sep = "")

paste("UC", c("Berkeley", "Los Angeles", "Irvine"), sep = "", collapse =
", and ")
```

Some Functions

```
s <- “I am a string.”

print(s)

print(s, quote = FALSE)

noquote(s)
```

Some Functions

```
s <- cat("U", "C", "L", "A")

s

format((1:5)/3, digit = 2)

format((1:5)/3, digit = 2, scientific = TRUE)

format(c("Pawnee", "rules", "Eagleton", "drools"), width = 10, justify =
"left")

format(c("Pawnee", "rules", "Eagleton", "drools"), width = 3, justify =
"left")

format(c("Pawnee", "rules", "Eagleton", "drools"), width = 7, justify =
"left")
```

Some Functions

```
s <- “I am a string.”

nchar(s)

tolower(s)

toupper(s)

casefold(s, upper = FALSE)

casefold(s, upper = TRUE)
```

Some Functions

```
school <- “UCLA”

chartr(old = “LA”, new = “SD”, school)

chartr(old = “LA”, new = “??”, school)
```

Some Functions

```
school <- “UCLA”

substr(school, start = 1, end = 4)

substr(school, start = 1, end = 5)
```

Some Functions

```
color <- “Red, Green, Blue”

c <- strsplit(color)

c <- strsplit(color, split = “, ”)

typeof(strsplit(color, split = “, ”))

unlist(c)
```

Some Functions

```
s <- "ucla"

s %in% "u"

ucla <- c("u", "c", "l", "a")

ucla %in% "u"
```

Some Functions

```
s <- “ucla”

grep(pattern = “uc”, s)

schools <- c(“ucla”, “ucberkeley”)

grepl(pattern = “uc”, schools)

gsub(pattern = “uc”, replacement = “University of California, ”,
schools)
```

Discussion

How to validate an UCLA email address?

E.g. f.gao@ucla.edu is valid. stats20@ucla.edu is valid.