R for Bioinformatics

Fundamentals of R: Part I

Md. Jubayer Hossain
CHIRAL Bangladesh
Founder & Executive Director

Last Updated on March, 2024

Agenda

- Introduction
- Background
- Basic Terms
- Getting Started With R
- Data Types in R
- Operators in R
- Questions?

Introduction

About the Course

- What do you hope to get out of the course?
  - R for Bioinformatics is an introductory course designed to equip learners with essential skills in using the R programming language for bioinformatics research.
- Why do you want to use R?
  - The course focuses on the practical application of R in genomic data manipulation, statistical analysis, data visualization, and reproducible research.

Learning Objectives

- `Data Manipulation:` Learn how to import, clean, and transform data in R for research purposes.
- `Data Visualization:` Master techniques for creating effective data visualizations in R to communicate research findings visually.
- `Statistical Analysis:` Develop skills in conducting statistical analysis using R for hypothesis testing, regression analysis, and other statistical tests.
- `Reproducible Research:` Implement principles of reproducible research using R to document and organize code, data, and analysis for replicability.

Course Platforms

- Website: https://omicscenter.github.io/RforResearch/
- Github: https://github.com/omicscenter/RforResearch
- Materials will be uploaded the night before class.
- Please check regularly official **Telegram group** for this course.

> We are constantly trying to improve content! Please refresh/download materials before class.

Course Format

- Lecture with live coding (possibly “Interactive”)
- Lab/Practical experience
- Two 5 min breaks each session - timing may vary

Assignment Policy

- 20% of your grade will be determined by an assignment during normal class hours.
- 50% of your grade will come from a 5-10 page report that explores in further detail one of the research areas in the class syllabus
- We expect that students will use original data collection, whether quantitative or qualitative analysis, to answer the questions posed.
- 30% of your grade will be determined by your attendance and participation in class.

Required Textbooks

The following books purchased and are available at the online book store. We have also a placed a copy of each on reserve at our Telegram group.

- Data Analysis for the Life Sciences with R, by Rafael Irizarry
- Introduction to Data Science, by Rafael Irizarry

Acknowledgements

Questions

- Please add any questions to the public Zoom chat.
- Coordinators will monitor the chat
- We'll also have time for questions at the break and at the end

Background

What is R?

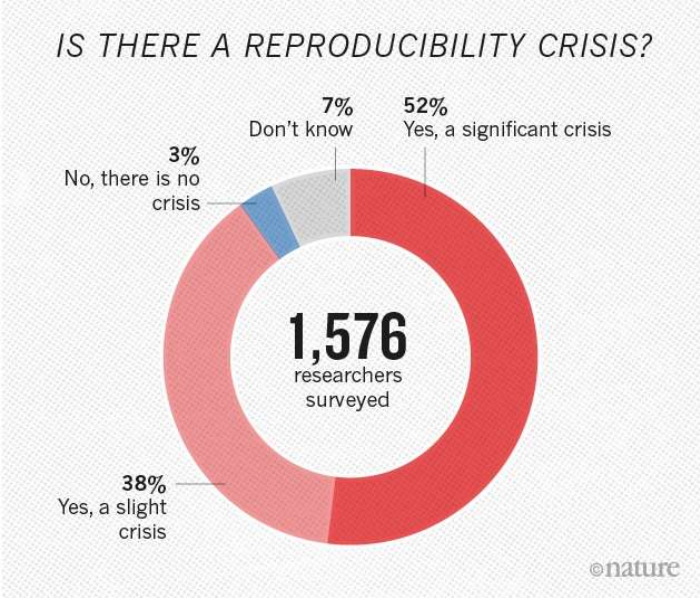
- R is a dialect of S(R is an implementation of the S programming language).
- R was created by Ross Ihaka and Robert Gentleman at the University of Auckland, New Zealand, and is developed by the R Development Core Team.
- R is a programming language and environment commonly used in statistical computing, data analytics and scientific research.
- R is a programming language and free software environment for statistical computing and graphics supported by the R Foundation for Statistical Computing.
- The R language is widely used among statisticians and data miners for developing statistical software and data analysis.

Why R?

- Free (open source)
- High level language designed for statistical computing
- Powerful and flexible - especially for data wrangling and visualization
- Extensive add-on software (packages)
- R is popular – and increasing in popularity.
- R runs on all platforms.(Windows, Linux and Mac)
- R is being used by the biggest tech giants(google, facebook, microsoft, twitter)
- Strong community

Reproducibility Crisis

- Quality of medical research is often low
- **Low quality code** in medical research part of the problem
- Low quality code is more **likely to contain errors**
- Reproducibility is often **cumbersome** and **time-consuming**



Basic Terms

Variable and Sample

- **Variable**: something measured or counted that is a characteristic about a sample
  - *Examples:* temperature, length, count, color, category
- **Sample**: individuals that you have data about.
  - *Examples:* people, houses, viruses etc.

Object

Object - an object is something that can be worked with in R - can be lots of different things!

- a matrix of numbers
- a plot
- a function

... many more

Columns and Rows

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa

- Sample/Observations = Row
- Variable = Column
- Data objects that looks like this is often called a **data frame**.
- Fancier versions from the tidyverse are called **tibbles** (more on that soon!).

Function

- **Function** - a function is a particular piece of code that allows you to do something in R. You can write your own, use functions that come directly from installing R, or use functions from additional packages.
- A function might help you add numbers together, create a plot, or organize your data. More on that soon!

Argument/Parameter

Argument/Parameter - what you pass to a function

```
1 sum(1, 20234)
```

[1] 20235

- Can be data like the number 1 or 20234
- Can be options about how you want the function to work such as `digits`

```
1 round(0.627, digits = 2)
```

[1] 0.63

```
1 round(0.627, digits = 1)
```

[1] 0.6

Package

- **Package** - a package in R is a bundle or "package" of code (and or possibly data) that can be loaded together for easy repeated use or for **sharing** with others.
- Packages are sort of analogous to a software application like Microsoft Word on your computer. Your operating system allows you to use it, just like having R installed (and other required packages) allows you to use packages.

More on Functions and Packages

- When you download R, it has a “base” set of functions/packages (**base R**)
- You can install additional packages for your uses from CRAN or GitHub
- These additional packages are written by RStudio or R users/developers (like us)

Using Packages

- Not all packages available on CRAN or GitHub are trustworthy
- RStudio (the company) makes a lot of great packages
- Who wrote it? **Hadley Wickham** is a major authority on R (Employee and Developer at RStudio)
- How to trust an R package
- Install packages: `install.packages("package_name")`
- Use packages: `library(package_name)`

Tidyverse Ecosystem and Base R

- We will mostly show you how to use tidyverse packages and functions.
- This is a newer set of packages designed for data science that can make your code more **intuitive** as compared to the original older Base R.
- **Tidyverse advantages**:
  - **consistent structure** - making it easier to learn how to use different packages
  - particularly good for **wrangling** (manipulating, cleaning, joining) data
  - more flexible for **visualizing** data
- Packages for the tidyverse are managed by a team of respected data scientists at RStudio.

Installation and Use

- Install all the packages in the tidyverse by running `install.packages("tidyverse")`.
- Run `library(tidyverse)` to load the core tidyverse and make it available in your current R session.
- Learn more about the tidyverse package at https://tidyverse.tidyverse.org.

Core tidyverse

- The core tidyverse includes the packages that you're likely to use in everyday data analyses.
- As of tidyverse 1.3.0, the following packages are included in the core tidyverse.
- The tidyverse also includes many other packages with more specialised usage.
- They are not loaded automatically with `library(tidyverse)`, so you'll need to load each one with its own call to `library()`.

Data Import

- As well as readr, for reading flat files, the tidyverse package installs a number of other packages for reading data:
  - DBI for relational databases. (Maintained by Kirill Müller.) You'll need to pair DBI with a database specific backends like RSQLite, RMariaDB, RPostgres, or odbc. Learn more at https://db.rstudio.com.
- readxl for `.xls` and `.xlsx` sheets.

Data Wrangling

In addition to tidyr, and dplyr, there are five packages (including stringr and forcats) which are designed to work with specific types of data:

- lubridate for dates and date-times.
- hms for time-of-day values.
- blob for storing blob (binary) data.

Program

In addition to purrr, which provides very consistent and natural methods for iterating on R objects, there are two additional tidyverse packages that help with general programming challenges:

- magrittr provides the pipe, %>% used throughout the tidyverse. It also provide a number of more specialised piping operators (like %$% and %<>%) that can be useful in other places.
- glue provides an alternative to `paste()` that makes it easier to combine data and strings.

Useful (+ mostly Free) Resources

- Tidyverse Skills for Data Science Book: https://jhudatascience.org/tidyversecourse/ (more about the tidyverse, some modeling, and machine learning)
- Tidyverse Skills for Data Science Course: https://www.coursera.org/specializations/tidyverse-data-science-r (same content with quizzes, can get certificate with $)
- R for Data Science: http://r4ds.had.co.nz/ (great general information)
- R basics by Rafael A. Irizarry: https://rafalab.github.io/dsbook/r-basics.html (great general information)
- Open Case Studies: https://www.opencasestudies.org/ (resource for specific public health cases with statistical implementation and interpretation)

Useful (+ mostly Free) Resources

- Dataquest: https://www.dataquest.io/
  (general interactive resource)
- Various “Cheat Sheets”:
  https://www.rstudio.com/resources/cheatsheets/
- R reference card: http://cran.r-project.org/doc/contrib/Short-refcard.pdf
- R jargon: https://link.springer.com/content/pdf/bbm%3A978-1-4419-1318-0%2F1.pdf
- R vs Stata: https://link.springer.com/content/pdf/bbm%3A978-1-4419-1318-0%2F1.pdf
- R terminology: https://cran.r-project.org/doc/manuals/r-release/R-lang.pdf

Getting Started With R

Your Setup

If you can, we suggest working virtually with a **large monitor or two screens**. This setup allows you to follow along on Zoom while also doing the hands-on coding.

- Install the latest version from: https://posit.co/
- Install RStudio from : https://posit.co/download/rstudio-desktop/

RStudio is an **integrated development environment** (IDE) that makes it easier to work with R.

More on that soon!

RStudio - Major concepts

- **RStudio** - an Integrated Development Environment (IDE) for R - makes it easier to use R.
- **Source/Editor** - “Analysis” Script + Interactive Exploration - In a .R file (we call a script), code is saved on your disk
- **R Console** - Where code is executed (where things happen) - Code is not saved on your disk
- **Workspace/Environment** - Tells you what objects are in R. What exists in memory/what is loaded?/what did I read in?
- **R Markdown** - Files (.Rmd) help generate reports that include your code and output.

RStudio

- **Quarto** - An open-source scientific and technical publishing system. Files (.qmd) help generate reports that include your code and output. https://quarto.org/
- **R Project** - Helps you organize your work. Helps with working directories (discussed later). Allows you to easily know which project you're on.
- **Quarto Project** - Quarto projects are directories that provide: A way to render all or some of the files in a directory with a single command (e.g. quarto render myproject).
- RStudio Keyboard shortcuts: http://www.rstudio.com/ide/docs/using/keyboard_shortcuts

What is Reproducibility?

- **Reproducibility** - A different analyst re-performs the analysis with the same code and the same data and obtains the same result.
- **Repeatable** - keeping everything the same but repeating the analysis - do we get the same results
- **Reproducible** - using the same data and analysis but in the hands of another researcher - do we get the same results?
- **Replicable** - with new data do we obtain the same inferences?

Running Your First R Program

- Now that you have installed R and RStudio successfully, let's try to create your first R program. We will try to create a simple Hello World program.
- A Hello World program is a simple program that simply prints a `Hello World` message on the screen. It's generally used to introduce a new language to learners.

```
1 message <-"Hello World!"
2 print(message)
```

[1] "Hello World!"

Running Your First R Program

```
1 message <-"Hello World!"
2 print(message)
```

[1] “Hello World!” * Here, we have created a simple variable called `message.` We have initialized this variable with a simple message string called `"Hello World!"`. On execution, this program prints the message stored inside the variable.

- Every output in R is preceded by a number (say n) in square brackets. This number means that the displayed value is the nth element printed.

R as a Calculator

```
1  2 + 2
```

[1] 4

```
1  2 * 4
```

[1] 8

```
1  2^3
```

[1] 8

Note: when you type your command, R inherently thinks you want to print the result.

R as a Calculator

- The R console is a full calculator
- Try to play around with it:
  - +, -, /, * are add, subtract, divide and multiply
  - ^ or ** is power
  - parentheses – (and) – work with order of operations
  - %% finds the remainder

R as a Calculator

```
1  2 + (2 * 3)^2
```

[1] 38

```
1  (1 + 3) / 2 + 45
```

[1] 47

```
1  6 / 2 * (1 + 2)
```

[1] 9

R as a Calculator

Try evaluating the following:

- `2 + 2 * 3 / 4 -3`
- `2 * 3 / 4 * 2`
- `2^4 - 1`

Variables (Identifiers) in R

- Variables are used to store data, whose value can be changed according to our need.
- A variable is a name given to a memory location, which is used to store values in a computer program.
- Variables in R programming can be used to store numbers (real and complex), words, matrices, and even tables.
- R is a dynamically programmed language which means that unlike other programming languages, we do not have to declare the data type of a variable before we can use it in our program.
- Unique name given to variable (function and objects as well) is identifier.

Rules for writing Identifiers in R

- Identifiers can be a combination of letters, digits, period (.) and underscore (_).
- It must start with a letter or a period. If it starts with a period, it cannot be followed by a digit.
- It should not start with a number (e.g: 2x)
- It should not start with a dot followed by a number (e.g: .2x)
- It should not start with an underscore (e.g: _x)
- Reserved words in R cannot be used as identifiers(e.g: TRUE, FALSE)

Basically, there are 5 naming conventions

- alllowercase: e.g. `myname`
- period.separated: e.g. `new.name`
- underscore_separated: e.g. `my_name`
- lowerCamelCase: e.g. `myName`
- UpperCamelCase: e.g. `MyName`

Assigning Values to Objects

- You can create objects from within the R environment and from files on your computer
- R uses `<-` to assign values to an object name (you might also see `=` used, but this is not best practice)
- Object names are case-sensitive, i.e. X and x are different

```
1 x <- 2
2 x
```

[1] 2

```
1 x * 4
```

[1] 8

```
1 x + 2
```

[1] 4

Creating Variables

Using equal(=) operator

```
1  x = 10
```

Using leftward(<-) operator

```
1  y <- 15
```

Reserved Keaywords in R

- Don’t use any reserved keyword as variable name. List all of reserved words in R by using (?Reserved).

```
1 ?Reserved
```

Entering Input

At the R prompt/console we type expressions.

```
num <- 10
```

The <- symbol is the **assignment** operator. The grammar of the language determines whether an expression is complete or not.

Evaluation

When a complete expression is entered at the R console, it is evaluated and the result of evaluated expression is returned. The result may be auto-printed.

```
1 x <- 10
2 x
```

[1] 10

```
1 x <- 10
2 print(x)
```

[1] 10

```
1 x <- 10
2 cat(x)
```

10

R Comments

Comments are portions of a computer program that are used to describe a piece of code. For example,

```r
# declare variable
age = 24

# print variable
print(age)
```

```
[1] 24
```

Types of Comments in R

In general, all programming languages have the following types of comments:

- single-line comments
- multi-line comments

However, in R programming, there is no functionality for multi-line comments. Thus, you can only write single-line comments in R.

R Single-Line Comments

```r
# this code prints Hello World
print("Hello World")
```

[1] “Hello World”

```r
# check type of variables
age <- 30
class(age)
```

[1] “numeric”

R Multi-Line Comments

- As already mentioned, R does not have any syntax to create multi-line comments.
- However, you can use consecutive single-line comments to create a multi-line comment in R. For example,

```
# this is a print statement
# it prints Hello World

print("Hello World")
```

[1] “Hello World”

Purpose of Comments

As discussed above, R comments are used to just document pieces of code. This can help others to understand the working of our code.

Here are a few purposes of commenting on an R code:

- It increases readability of the program for users other than the developers.
- Comments in R provide metadata of the code or the overall project.
- Comments are generally used by programmers to ignore some pieces of code during testing.
- They are used to write a simple pseudo-code of the program.

How to Create Better Comments?

You should always keep in mind the following points while writing comments.

- Use comments only to describe what a particular block of code does, not how it does.
- Don’t overuse comments. Try to make your code self-explanatory.
- Try to create comments that are as precise as possible.
- Don’t use redundant comments.

Data Types in R

R Data Types

- A variable can store different types of values such as numbers, characters etc.
- These different types of data that we can use in our code are called data types. For example,

```
1  x <- 123L
```

- Here, `123L` is an integer data. So the data type of the variable `x` is `integer`.

R Data Types

We can verify this by printing the class of `x` using `class()` function.

```
x <- 123L
# print value of x
x
```

[1] 123

```
# print type of x
class(x)
```

[1] “integer”

Different Types of Data Types

Data Type	Example	Description
Logical	True, False	It is a special data type for data with only two possible values which can be construed as true/false.
Numeric	12,32,112,5432	Decimal value is called numeric in R, and it is the default computational data type.
Integer	3L, 66L, 2346L	Here, L tells R to store the value as an integer,

Different Types of Data Types

Data Type	Example	Description
Complex	Z=1+2i, t=7+3i	A complex value in R is defined as the pure imaginary value i.
Character	'a', '"good"', "TRUE", '35.4'	In R programming, a character is used to represent string values. We convert objects into character values with the help of `as.character()` function.
Raw		A raw data type is used to holds raw bytes.

Logical Data Type

The `logical` data type in R is also known as `boolean` data type. It can only have two values: `TRUE` and `FALSE`. For example,

```
bool1 <- TRUE
# print bool1
bool1
```

[1] TRUE

```
# print type of bool1
class(bool1)
```

[1] “logical”

```
# print bool2
bool2 <- FALSE
bool2
```

[1] FALSE

```
# print type of bool2
class(bool2)
```

[1] “logical”

Logical Data Type

You can also define `logical` variables with a single letter - `T` for `TRUE` or `F` for `FALSE`. For example,

```
1 is_weekend <- F
2 class(is_weekend)  # "logical"
```

[1] “logical”

```
1 is_weekday <- T
2 class(is_weekday)  # "logical"
```

[1] “logical”

Numeric Data Type

In R, the `numeric` data type represents all real numbers with or without decimal values. For example,

```
# floating point values
weight <- 63.5
weight
```

[1] 63.5

```
# check variable types
class(weight)
```

[1] “numeric”

```
# real numbers
height <- 182
height
```

[1] 182

```
# check variable types
class(height)
```

[1] “numeric”

Integer Data Type

The integer data type specifies real values without decimal points. We use the suffix L to specify integer data. For example,

```
integer_variable <- 186L
integer_variable
```

[1] 186

```
# check variable types
class(integer_variable)
```

[1] “integer”

Complex Data Type

The `complex` data type is used to specify purely imaginary values in R. We use the suffix `i` to specify the imaginary part. For example,

```
# 2i represents imaginary part
complex_value <- 3 + 2i

# print class of complex_value
class(complex_value)
```

[1] “complex”

Character Data Type

- The `character` data type is used to specify character or string values in a variable.
- In programming, a string is a set of characters. For example, `'A'` is a single character and `"Apple"` is a string.
- You can use single quotes `''` or double quotes `""` to represent strings. In general, we use:
  - `''` for character variables
  - `""` for string variables

Character Data Type

For example,

```
# create a string variable
fruit <- "Apple"
class(fruit)
```

[1] “character”

```
# create a character variable
my_char <- 'A'
class(my_char)
```

[1] “character”

Raw Data Type

A `raw` data type specifies values as raw bytes. You can use the following methods to convert character data types to a raw data type and vice-versa:

- `charToRaw()` - converts character data to raw data
- `rawToChar()` - converts raw data to character data

Raw Data Type

For example,

```
# convert character to raw
raw_variable <- charToRaw("Welcome to Programiz")

print(raw_variable)
```

[1] 57 65 6c 63 6f 6d 65 20 74 6f 20 50 72 6f 67 72 61 6d 69 7a

```
print(class(raw_variable))
```

[1] “raw”

```
# convert raw to character
char_variable <- rawToChar(raw_variable)

print(char_variable)
```

[1] “Welcome to Programiz”

```
print(class(char_variable))
```

[1] “character”

Operators in R

Operators in R

- In R, operators are symbols or characters that perform specific operations on variables, values, or expressions.
- R provides various types of operators, including arithmetic operators, assignment operators, comparison operators, logical operators, and more.
- Operators in R can mainly be classified into the following categories.
  - Arithmetic Operators
  - Relational Operators
  - Logical Operators

Arithmetic Operators

Operator	Operation	Example
+	Addition	5 + 2 = 7
-	Subtraction	4 - 2 = 2
*	Multiplication	2 * 3 = 6
/	Division	4 / 2 = 2
%%	Modulo	5 %% 2 = 1
^	Power	4 ^ 2 = 16

Example: Arithmetic Operators

```
1 x <- 10
2 y <- 2
3
4 # Addition
5 x+y
```

[1] 12

```
1 # Subtraction
2 2-5
```

[1] -3

```
1 # Multiplication
2 2 * 5
```

[1] 10

```
1 # Division
2 2 / 5
```

[1] 0.4

Example: Arithmetic Operators

```
1 x <- 10
2 y <- 2
3
4 # Exponent
5 2 ^ 5
```

[1] 32

```
1 # Modulus(Remainder from division)
2 2 %% 5
```

[1] 2

Relational Operators

Operator	Operation	Example
>	Greater that	`5 > 6` returns `FALSE`
<	Less than	`5 < 6` returns `TRUE`
==	Equals to	`10 == 10` returns `TRUE`
!=	Not equal to	`10 != 10` returns `FALSE`
>=	Greater than or equal to	`5 >= 6` returns `FALSE`
<=	Less than or equal to	`6 <= 6` returns `TRUE`

Example: Relational Operators

The `output` of a comparison is a `boolean value`. For example, to check if two numbers are equal, you can use the `==` operator.

```
x <- 10
y <- 23

# compare x and y
x == y # FALSE
```

[1] FALSE

Similarly, to check if x is less than y, you can use the `<` operator.

```
x <- 10
y <- 23

# compare x and y
x < y # TRUE
```

[1] TRUE

Logical Operators

Logical operators are used to compare the output of two comparisons. There are `three` types of logical operators in R. They are:

- AND operator (&)
- OR operator (|)
- NOT operator (!)

AND Operator (&)

- The AND operator `&` takes as input two logical values and returns the output as another logical value.
- The output of the operator is `TRUE` only when `both` the input logical values are either `TRUE` or evaluated to `TRUE`.
- Let a and b represent two operands. `0` represents FALSE and 1 represents TRUE. Then,

a	b	a & b
1	1	1
1	0	0
0	1	0
0	0	0

Example: AND Operator (&)

```
# print & of TRUE and FALSE combinations
TRUE & TRUE
```

[1] TRUE

```
TRUE & FALSE
```

[1] FALSE

```
FALSE & TRUE
```

[1] FALSE

```
FALSE & FALSE
```

[1] FALSE

```
# print & of TRUE and FALSE combinations
x <- 10
y <- 23
z <- 12

# compare
x<y & y>z
```

OR Operator (|)

The OR operator | returns TRUE if all or any one of the logical inputs is TRUE or evaluates to TRUE. If all of them are FALSE, then it returns FALSE. Consider the table below.

a	b	a | b
1	1	1
1	0	1
0	1	1
0	0	0

Example: OR Operator (|)

```
# print | of TRUE and FALSE combinations
TRUE | TRUE
```

[1] TRUE

```
TRUE | FALSE
```

[1] TRUE

```
FALSE | TRUE
```

[1] TRUE

```
FALSE | FALSE
```

[1] FALSE

```
# print | of TRUE and FALSE combinations
w <- 54
x <- 12
y <- 25
z <- 1

w>x | x>y | z>w
```

NOT (!) Operator

The NOT operator ! is used to negate the logical values it is used on. If the input value is TRUE, it will turn to FALSE and vice-versa.

a	!a
1	0
0	1

Example: NOT (!) Operator

```
1 # print ! of TRUE and FALSE
2 !TRUE
```

[1] FALSE

```
1 !FALSE
```

[1] TRUE

Here, the output is the negation of the input.

- We can use the ! operator with comparisons.
- For example, !(x > 12) is the same as x <= 12. This means that x is not greater than 12. Which means that x can be less than or equal to 12.

Questions?