Introduction to R

Michael Neely, MD
Professor of Pediatrics and Clinical Scholar
Director, Laboratory of Applied Pharmacokinetics and Bioinformatics
University of Southern California, Children's Hospital Los Angeles

Children's Hospital LOS ANGELES
We Treat Kids Better

USC University of Southern California

LAPKB
Laboratory of Applied Pharmacokinetics & Bioinformatics

What is R?

- A programming language
- Data analysis tool
- An environment

How do I get it?

Basic objects in R

- Variables
  - Contain values
    - Numbers, characters, logical (TRUE/FALSE)
  - Simple to complex structures
    - Vector, Matrix, Data Frame, Array, List
- Functions
  - Accept objects and argument values
  - Perform operations on the objects, modified by argument values
  - Return results of the operations

Basic objects in R

- Environments
  - Self-contained “rooms” in the house (project)
  - Objects are generally exclusive to an environment
    - Can be passed between environments
  - Example: functions

Variables

- Can be named anything
- Best to avoid naming the same as a function
- Conventions
  - snake_case
  - camelCase
  - Values use the assignment operator "<-"
- Example
  - this_numerical_variable <- 1:5
  - myName <- c("Michael", "Neely")

Variable types

- Vector
  - One dimensional
  - One data type, e.g. all numbers or all characters
  - Access elements with [x]

```
> this_numerical_variable
[1] 1 2 3 4 5
> this_numerical_variable[2]
[1] 2
```

Variable types

- Matrix
  - Two dimensional
  - One data type, e.g. all numbers or all characters
  - Access elements with [r, c]

```
> m <- matrix(1:4, nrow = 2)
> m
     [,1] [,2]
[1,]    1    3
[2,]    2    4
> m[1,2]
[1] 3
> m[1,]
[1] 1 3
```

Variable types

- Array
  - Multi-dimensional
  - One data type, e.g. all numbers or all characters
  - Access elements with [a, b, c, …]

Variable types

- Data frame
  - Two dimensional
  - Each column is one data type, but can differ by column
  - Access elements with [r, c] or columns with $

```
> df <- data.frame(id = c("A","B"), wt = c(50, 60))
> df
  id wt
1  A 50
2  B 60
> df$wt
[1] 50 60
> df[2,]
  id wt
2  B 60
```

Variable types

- List
  - One dimensional
  - Each item can be any other data type or even a list
  - Access elements with [[x]] or $

```
> list1 <- list(m, df = df)
> list1
[[1]]
     [,1] [,2]
[1,]    1    3
[2,]    2    4

$df
  id wt
1  A 50
2  B 60
```

Functions

- Can be named anything
- The name of the function returns the code within the function

```
> print
function (x, ...)
UseMethod("print")
<bytecode: 0x12ff98330>
<environment: namespace:base>
```

- The name followed by parentheses activates the function

```
> print("Hello World")
[1] Hello World
```

Functions

- Assign a variable to capture result of a function

```
> a <- mean(1:4)
```

- Type the variable name to see its contents

```
> a
[1] 2.5
```

Function arguments

- Arguments are named
- Arguments usually have default values
- Argument shorthand tips
  - Omitting argument names

round(x, digits = 0)

```
> round(4.23, 1)
[1] 4.2
```

  - Skipping default arguments

```
> round(4.23)
[1] 4
```

Function arguments

- You must name arguments if…
  - You skip a default argument, you must name the subsequent

mean(x, trim = 0, na.rm = FALSE, …)

```
> mean(c(1:4, NA), na.rm = T)
[1] 2.5
```

  - You do not follow the order of the arguments

```
> mean(na.rm = T, x = c(1:4, NA))
[1] 2.5
```

Help on functions

- Type "?" followed by the function name for exact search

```
> ?range
```

- Type "??" followed by the function name for broader search

```
> ??range
```

R frameworks

- Most common: S3
  - Variables
    - Can have any form
    - Have a class

```
> class(pi)
[1] "numeric"
```

  - Functions
    - Generic handle many classes and call the appropriate…

plot(x, y, …)

    - Method, which is specific to a class of object

plot(res1$op) -> plot.PMop(res1$op)

R frameworks

- Most like object oriented programming: R6
  - Variables
    - Have pre-defined form
    - Still have a class
    - Have attached methods (functions) in the variable definition
  - Functions
    - Are attached to variables only

R6

```
> PM_data
<PM_data> object generator
  Public:
    data: NULL
    standard_data: NULL
    initialize: function (data, quiet = F)
    write: function (file_name)
    nca: function (...)
    plot: function (...)
    print: function (standard = F, viewer = T, ...)
    summary: function (formula, FUN, include, exclude)
    clone: function (deep = FALSE)
  Private:
    dataObj: NULL
    validate: function (dataObj = NULL, quiet)
  Parent env: <environment: namespace:Pmetrics>
  Locked objects: TRUE
  Locked class: FALSE
  Portable: TRUE
```

Pmetrics frameworks

- Prior to v. 2.0, Pmetrics was only S3
- As of v. 2.0, Pmetrics uses R6 and S3
  - All Pmetrics objects are R6
  - Relevant functions are attached
  - Functions that have applications outside a Pmetrics object (e.g. makeAUC) retain S3 versions

Why both frameworks?

- S3
  - Common
  - Familiar
  - Easy to create variables and functions on the fly
  - Useful when functions needed for objects not pre-defined
- R6
  - Ensures consistency for standard objects
  - No file moving
  - Reduces errors by ensuring objects are valid prior to running methods
  - Allows more complex yet organized data structures
  - Consistent for object-oriented programmers (e.g. Python, Julia)