Introduction to R; Algorithms

Myung Jung Kim

mjkim12@illinois.edu

6/30/2021

Section 1

Features of R

What is R?

- R is a programming language and software environment used primarily for statistical computing and graphics.
- It is free and open-source.
- R has a diverse and welcoming community
  - A massive set of packages for statistical modeling, machine learning, visualization, and importing and manipulating data.
- Powerful tools for communicating your results.
  - Using RMarkdown, you can turn your results into HTML files, PDFs, Word documents, PowerPoint presentations,dashboards and mored

Some Negatives

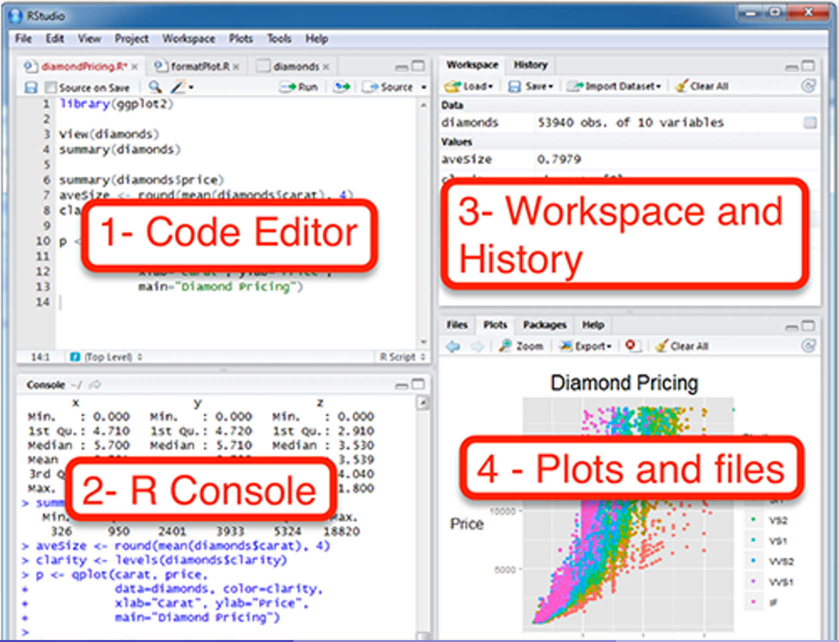
- Poorly written R code can be terribly slow.
- There are multiple ways of doing the same thing. (inconsistency across different packages).
- Handles large datasets (100 MB), but can have some trouble with massive datasets (GBs)
- Error trapping can be confusing and frustrating

Installation

You need to download two software:

1. **R** from the Comprehensive R Archive (CRAN) Network
2. **RStudio** from the Rstudio
   - RStudio (IDE for working with R). It's an excellent interface used to interact with R!

Four Panes



```
library(ggplot2)

View(diamonds)
summary(diamonds)

summary(diamonds$price)
aveSize <- round(mean(diamonds$carat), 4)
cla

p <


             main="Diamond Pricing")
```

1- Code Editor

```
       x                 y                 z
 Min.   : 0.000   Min.   : 0.000   Min.   : 0.000
 1st Qu.: 4.710   1st Qu.: 4.720   1st Qu.: 2.910
 Median : 5.700   Median : 5.710   Median : 3.530
 Mean                                          3.539
 3rd Q                                         4.040
 Max.                                          1.800
> summ
   Min.                                   Max.
    326      950     2401     3933     5324    18820
> aveSize <- round(mean(diamonds$carat), 4)
> clarity <- levels(diamonds$clarity)
> p <- qplot(carat, price,
+            data=diamonds, color=clarity,
+            xlab="Carat", ylab="Price",
+            main="Diamond Pricing")
>
```

2- R Console

Data	
diamonds	53940 obs. of 10 variables
Values	
aveSize	0.7979

3- Workspace and History

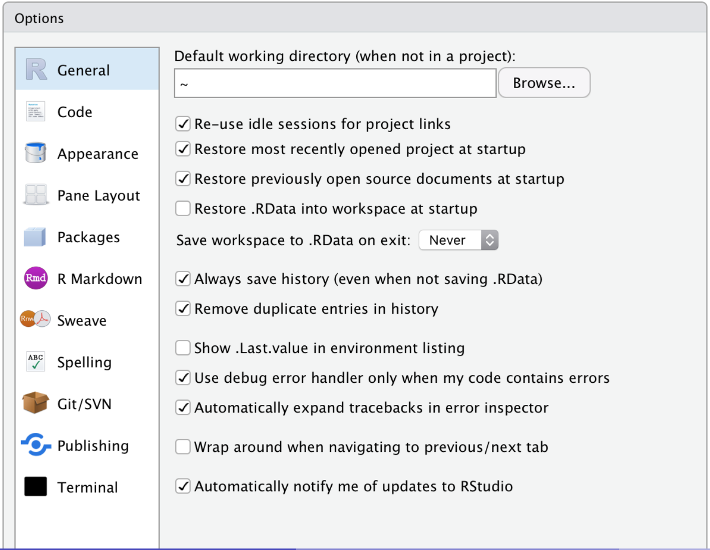
4 - Plots and files

Error and Debugging

- Symbol on the console (=State of R)
  - **Ready** (>) for new code
  - **Waiting** (+) for you to finish old code
- Errors
  - **Misspelled object or function:** R is case-sensitive, so if you don't use the correct capitalization you'll receive an error.
  - **Punctuation problem**: Having an extra space, missing a comma, or using a comma (,) instead of a period (.) can give you an error.
- Reference: https://bookdown.org/ndphillips/YaRrr/debugging.html

General Settings (RStudio)

Find the ‘**Tools**’ (‘Preferences’) menu item, navigate to ‘**Global Options**’ (‘Code Editing’)



Default working directory (when not in a project): ~

☑ Re-use idle sessions for project links

☑ Restore most recently opened project at startup

☑ Restore previously open source documents at startup

□ Restore .RData into workspace at startup

Save workspace to .RData on exit: Never

☑ Always save history (even when not saving .RData)

☑ Remove duplicate entries in history

□ Show .Last.value in environment listing

☑ Use debug error handler only when my code contains errors

☑ Automatically expand tracebacks in error inspector

□ Wrap around when navigating to previous/next tab

☑ Automatically notify me of updates to RStudio

Working Directory

Working directory is a folder where R reads and saves files.

- To Change the working directory: **`setwd()`**
- To know the current R working directory: **`getwd()`**

RMarkdown

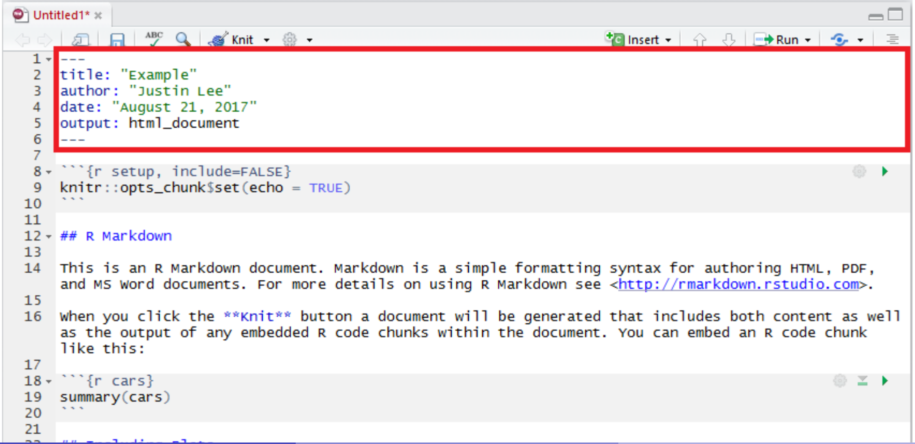
We usually use **RMarkdown** (`File`→ `New File`→ `RMarkdown`).

- **RMarkdown** is a file format for making dynamic documents with R (extension: `.Rmd`)
  - Click the '**knit**' button to produce HTML files, PDFs, Word documents, PowerPoint presentations, and etc.
  - Reference: R Markdown
- An RMarkdown document contains three parts:

1. YAML metadata
2. Text
3. Code Chunks

1) YAML metadata

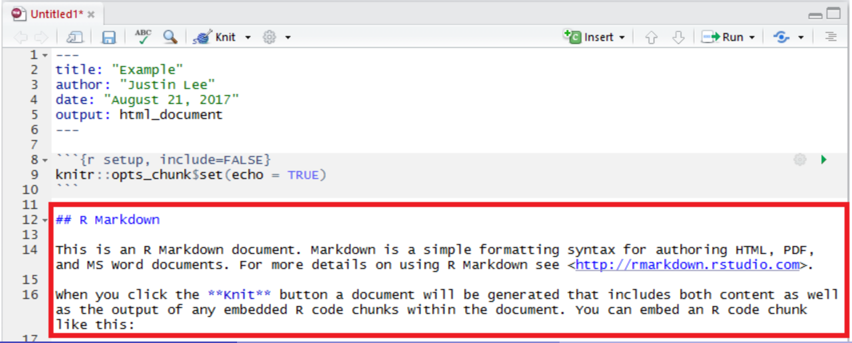
- YAML is where you specify document configurations, layout and properties such as name, date, output format, etc.
- You can also include additional formatting options such as a table of contents
- Reference:R Markdown: The Definitive Guide



~~~
---
title: "Example"
author: "Justin Lee"
date: "August 21, 2017"
output: html_document
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
```

## R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see <http://rmarkdown.rstudio.com>.

When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

```{r cars}
summary(cars)
```
~~~

2) Text

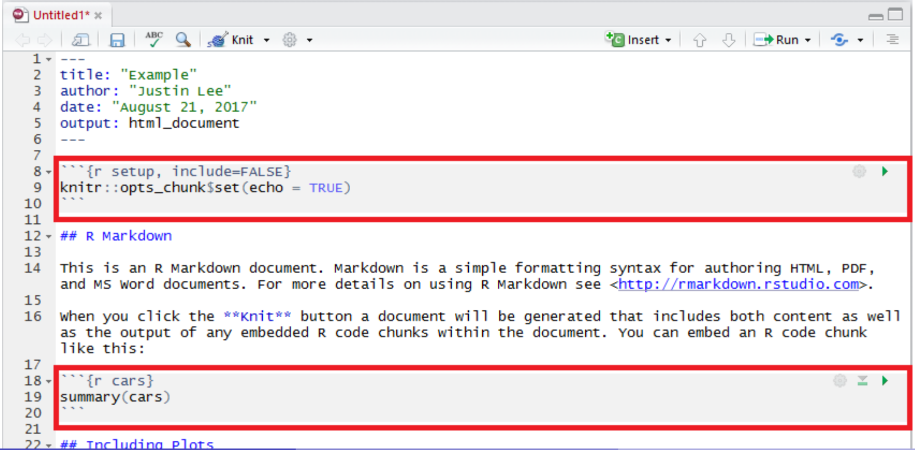
- The text in an R Markdown document is written with the Pandoc’s Markdown.
- Emphases: *italics* (surrounded text by *), **bold** (surround text by **), or `code style` (surround text by ‘)
- Headers, subheads: #, ##, ###,. . . + space
- Lists: ∗, +, or − with a space
- Reference: https://bookdown.org/yihui/rmarkdown/markdown-syntax.html



~~~
---
title: "Example"
author: "Justin Lee"
date: "August 21, 2017"
output: html_document
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
```

## R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF,
and MS Word documents. For more details on using R Markdown see <http://rmarkdown.rstudio.com>.

When you click the **Knit** button a document will be generated that includes both content as well
as the output of any embedded R code chunks within the document. You can embed an R code chunk
like this:
~~~

3) Code Chunk

- Where you write your code!
- Make the chunk by:
  - **shift + command + i** (OS X: **Ctrl + option + i**)
  - Or, by typing the chunk delimiters ```` ```{r} ```` and ```` ``` ````
- You run the code: **command + return**



~~~
---
title: "Example"
author: "Justin Lee"
date: "August 21, 2017"
output: html_document
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
```

## R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see <http://rmarkdown.rstudio.com>.

When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

```{r cars}
summary(cars)
```

## Including Plots
~~~

More about Chunk

- **Chunk Name:** Chunks can be given an optional name: `` ```{r by-name} ``.
- **Chunk Options:** Chunk output can be customized with options. The most important set of options controls if your code block is executed and what results are inserted in the finished report.
  - `eval = FALSE`, prevents code from being evaluated.
  - `include = FALSE` runs the code, but doesn't show the code or results in the final document.
  - `echo = FALSE` prevents code, but not the results from appearing in the finished file. Use this when writing reports aimed at people who don't want to see the underlying R code.
  - `message = FALSE` or `warning = FALSE` prevents messages or warnings from appearing in the finished file.
- Reference: https://r4ds.had.co.nz/r-markdown.html

EXERCISE (1)

1. Set up your working directory.
2. Open Rmarkdown file.
3. Create the file with your name as the author and the title as “Exercise 1”
4. Below the sample contents, add a new header with any title and some texts.
5. Make a code chunk with a chunk name “Exercise 1-1”, and put `100` in the code chunk with a comment. Make both the code and output appear in the final document.
6. Make another code chunk with a chunk name “Exercise 1-2”, and put `10-3` in the code chunk. Make only the output (no code) appear in the final document.
7. Knit the file to a PDF file and save it.

Section 2

Basic Algorithms

Basic arithmetic operations

```
2+2  #addition: +
```

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

```
10-3 #subtraction: -
```

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

```
11*5 #multiplication: *
```

```
## [1] 55
```

```
10/2 #division: /
```

```
## [1] 5
```

```
2^4 #exponentiation ^
```

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

Functions in R

- A function is a self-contained collection of commands that together constitute a single, more complex mathematical task.
- Basic arithmetic functions

```
log2(10) # logarithms base 2 of x
```

```
## [1] 3.321928
```

```
exp(10) # Exponential of x
```

```
## [1] 22026.47
```

```
abs(10.43) # absolute value of x
```

```
## [1] 10.43
```

- Other R's built-in functions:
  - `mean()`
  - `sum()`
  - `length()`
  - `lm()`, to run a basic least squares linear regression (OLS).
- Use functions from external packages or write your own. .

R Packages

- Along with a number of built-in functions, as an open-source project, R has many packages that provide additional functions and data.
- To install: `install.packages("name")`
- To load: `library(name)`
- Install once, load everytime you open a new session!

Example

- Install the package called MASS

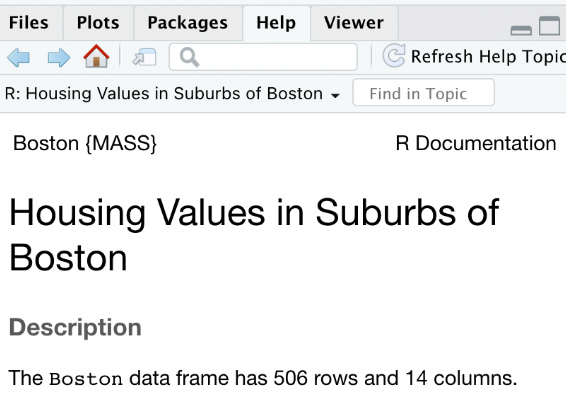
```r
install.packages("MASS")
library(MASS)
ls("package:MASS")
```

```
##   [1] "abbey"             "accdeaths"         "addterm"
##   [4] "Aids2"             "Animals"           "anorexia"
##   [7] "area"              "as.fractions"      "bacteria"
##  [10] "bandwidth.nrd"     "bcv"               "beav1"
##  [13] "beav2"             "biopsy"            "birthwt"
##  [16] "Boston"            "boxcox"            "cabbages"
##  [19] "caith"             "Cars93"            "cats"
##  [22] "cement"            "chem"              "con2tr"
##  [25] "contr.sdif"        "coop"              "corresp"
##  [28] "cov.mcd"           "cov.mve"           "cov.rob"
##  [31] "cov.trob"          "cpus"              "crabs"
##  [34] "Cushings"          "DDT"               "deaths"
##  [37] "denumerate"        "dose.p"            "drivers"
##  [40] "dropterm"          "eagles"            "enlist"
##  [43] "epil"              "eqscplot"          "farms"
##  [46] "fbeta"             "fgl"               "fitdistr"
##  [49] "forbes"            "fractions"         "frequency.po
##  [52] "GAGurine"          "galaxies"          "gamma.disper
##  [55] "gamma.shape"       "gehan"             "genotype"
```

Getting Help

- After running the `ls()` command, we see the MASS package contains something called **"Boston"** (16th element)
- Wonder what it is? Use **`help()`** function or **`?name`**

```
help(Boston)
?Boston
```



Boston {MASS} R Documentation

Housing Values in Suburbs of Boston

Description

The `Boston` data frame has 506 rows and 14 columns.

You can see that “Boston” is a data frame.

- You can see how the data look like by putting `View(Boston)` in the command line.

EXERCISE (2)

1. Install and load a package called “car.” 2. Using `ls()` function, explore what the package contains.
2. Using ? or `help()` function, find and explain what the “Boot” is in the package.

Creating objects (Assigning)

- Using “assignment operator” ((“<-”), we assign values on the right to objects on the left.
- No “->”
- “=” works too, but not recommended.

Example

```
weight_kg <- 55
weight_kg
```

```
## [1] 55
```

Data Types

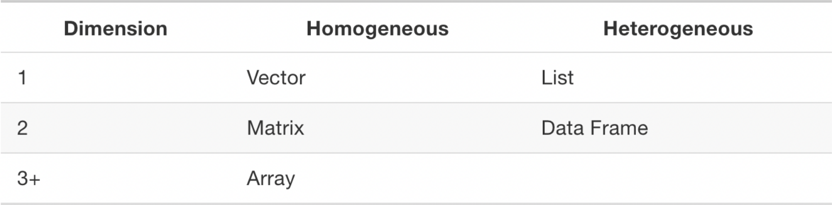
While doing programming, you use various variables to store various information (like how you assigned a value to objects). A **data type** is a type of information.

- R has 6 basic data types:
  - character: “a”, “mydata”
  - numeric: `5`, `13.5`
  - integer: `1:10`, `2L`
  - logical: `TRUE`, `FALSE`
  - complex: `1+4i`
- R provides many functions to examine features of vectors and other objects:
  - `class()` : what kind of object is it?
  - `typeof()` : what is the object’s data type?
  - `length()` : how long is it?

Basic Data Structures

A **data structure** is a collection of different forms and different types of data that has a set of specific operations that can be performed.

- A data structure is either **homogeneous** (all elements are of the same data type) or **heterogeneous** (elements can be of more than one data type).
- R's basic data structures



Dimension	Homogeneous	Heterogeneous
1	Vector	List
2	Matrix	Data Frame
3+	Array	

(1) Vector (& list)

Vector is a basic data structure in R.

- If you create vectors by specifying their content, R guesses the appropriate mode of storage for the vector.
  - When different types are mixed inside a vector, that's list. R finds a mode that can most easily accommodate all the elements it contains.

```r
x <- c(1,2,3) # a vector of mode numeric

y <- c(TRUE, FALSE, TRUE) # a vector of mode logical

family <- c("MJ", "Joon", "Yul") # a vector of mode character

z <- c(30, "gold") # a list of character
```

How to create a vector

Vectors are generally created using the `c()` function.

```
xx <- c(1,3,6,7,10)
xx
```

```
## [1]  1  3  6  7 10
```

You can also use ":" operator, `seq()`, and `rep()` functions

```
qq <- c(1:4)#create consecutive numbers
qq
```

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

```
yy <- seq(1, 3, by=0.5)  # generate sequence
yy
```

```
## [1] 1.0 1.5 2.0 2.5 3.0
```

```
zz <- rep(1,8) #replicate the value
zz
```

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

EXERCISE (3)

1. Assign prices to “strawberry”, “banana”, “apple”, and “orange”.
2. Combine them into a vector called “fruits.”
3. Using R’s built-in functions, calculate the average price and the total price of the fruits. (Show the code and outputs)

(2) Matrices

Matrices have rows and columns containing a single data type (2 dimensions). In a matrix, the order of rows and columns is important.

```
x <- 1:9
x
```

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

```
X <-  matrix(x, nrow =3, ncol =3)
X
```

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

(3) Data Frames

- Unlike a matrix, a data frame is not required to have the same data type for each element.
- Data frame is the most common way we store and interact with data!
- A data frame is a **list** of vectors. So, each vector must contain the same data type, but the different vectors can store different data types.

```r
example_data = data.frame(x = c(2, 4, 1, 7),
                          y = c(rep("Hello", 3), "Goodbye"),
                          z = rep(c(TRUE, FALSE), 2))


example_data
```

```
##   x       y     z
## 1 2   Hello  TRUE
## 2 4   Hello FALSE
## 3 1   Hello  TRUE
## 4 7 Goodbye FALSE
```

You can use a number of functions to obtain information about matrix and data frame

- dim() : dimension (the number of columns and rows)
- nrow() : number of rows
- ncol() : number of columns, it is also the number of observations

You will learn more about them in the "Playing with data I and II" sessions! (Subsetting, Vectorization, Logical operators, Adding elements, removing elements. . .)

EXERCISE (4)

1. Create three vectors with three different data types.
2. Create a data frame with 4 rows and 2 columns.
3. Check the dimension of the data frame with dim() function.

[Extra Note] Style and code conventions

- Develop the clear and consistent code style early on! This will benefit your scientific writing in general and make R Code easier to read, share, and verify.
- Some conventions:
  - When reading texts about R commands:
    - ★ Package names are shown in a bold font over a grey box, *e.g.* **`tidyr`**.
    - ★ Functions are shown in normal font followed by parentheses and also over a grey box , *e.g.* `plot()`, or `summary()`.
    - ★ Other R objects, such as data, function arguments or variable names are again in normal font over a grey box, but without parentheses, *e.g.* `x` and `apples`.
    - ★ Sometimes we might directly specify the package that contains the function by using two colons, *e.g.* `dplyr::filter()`.
  - **R style Guide:** Advanced R Style Guide by Hadley Wickham
    - ★ Left-hand assignment (foo <- , not -> foo)
    - ★ Variable and function names should be lowercase. Use an underscore (_) to separate words within a name.
    - ★ Generally, variable names should be nouns and function names should be verbs.
    - ★ Place spaces around all infix operators (=,+,-,<-, etc.).
    - ★ An opening curly brace { should never go on its own line and should

Reference

1. Intro R Workshop (Schlegel & Smit 2018)
2. Reference about Beamer presentation <https://bookdown.org/yihui/rmarkdown/beamer-presentation.html>
3. Data types and structure: <https://swcarpentry.github.io/r-novice-inflammation/13-supp-data-structures/>

(Optional) Subsetting

To subset a vector, we use square brackets, `[]`.

```
xx
```

```
## [1]  1  3  6  7 10
```

```
xx[3] #returns the third element
```

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

```
xx[2:4] #we exclude certain indexes
```

```
## [1] 3 6 7
```

```
xx[c(1,5)] #again, excluding some
```

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

(Optional) Vectorization

One of the biggest strengths of R is its use of vectorized operations.

```
qq = 1:10
qq +1
```

```
##  [1]  2  3  4  5  6  7  8  9 10 11
```

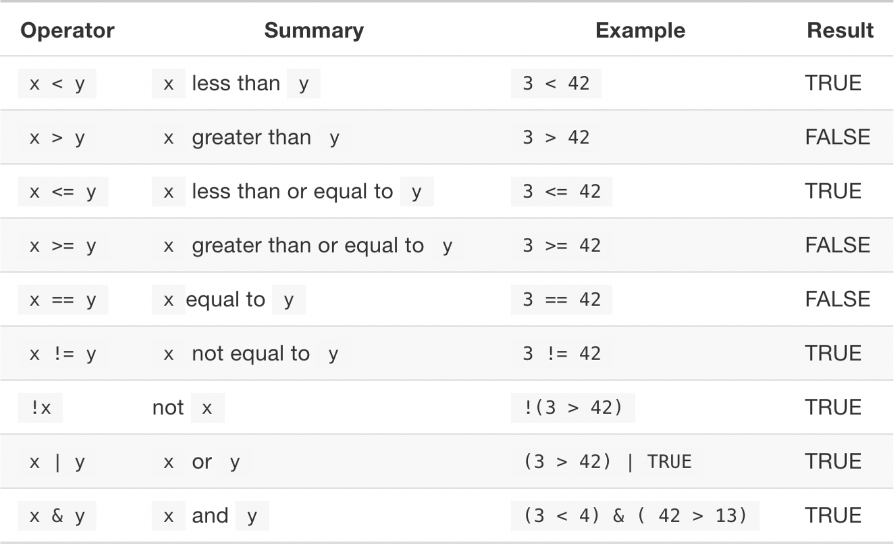
```
2*qq
```

```
##  [1]  2  4  6  8 10 12 14 16 18 20
```

```
log(qq)
```

```
##  [1] 0.0000000 0.6931472 1.0986123 1.3862944 1.6094379 1.79
##  [8] 2.0794415 2.1972246 2.3025851
```

(Optional) Logical Operators



Operator	Summary	Example	Result
`x < y`	`x` less than `y`	`3 < 42`	TRUE
`x > y`	`x` greater than `y`	`3 > 42`	FALSE
`x <= y`	`x` less than or equal to `y`	`3 <= 42`	TRUE
`x >= y`	`x` greater than or equal to `y`	`3 >= 42`	FALSE
`x == y`	`x` equal to `y`	`3 == 42`	FALSE
`x != y`	`x` not equal to `y`	`3 != 42`	TRUE
`!x`	not `x`	`!(3 > 42)`	TRUE
`x | y`	`x` or `y`	`(3 > 42) | TRUE`	TRUE
`x & y`	`x` and `y`	`(3 < 4) & ( 42 > 13)`	TRUE

In R, logical operators are vectorized.

(Optional) Adding Elements

The function `c()` (for combine) can be used to add elements to a vector.

```
family <- c(family, "baby2")
```

- More information: https://daviddalpiaz.github.io/appliedstats/data-and-programming.html