Getting started with R

Matt Lee (mlee8@g.harvard.edu), PHS Launch 2022

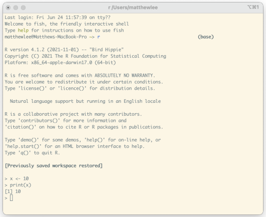
What is R?

- R is an open-source **interpreted** programming language: when you install R, you install an *interpreter* that translates your R code into computer code (sometimes called "machine" code), which is what actually gets run
- This is in contrast to **compiled** languages (e.g. C or C++), where the programmer writes code that is directly converted into machine code
- Several advantages of interpreted languages: much more user friendly, easily read, consistent across operating systems
- Some disadvantages: often slower, and less control over system hardware*

Action step: Install R (https://cloud.r-project.org/)

R Studio: Integrated Development Environments (IDEs)

- We can interact with R directly via a command line (e.g. Terminal)



```
Last login: Fri Jun 24 11:57:39 on tty??
Welcome to fish, the friendly interactive shell
Type help for instructions on how to use fish
matthewlee@Matthews-MacBook-Pro ~> r                                   (base)

R version 4.1.2 (2021-11-01) -- "Bird Hippie"
Copyright (C) 2021 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin17.0 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Previously saved workspace restored]

> x <- 10
> print(x)
[1] 10
> 
```

- But this is not very pretty or reproducible! A population alternative is to use an IDE, such as R Studio, which is a program that adds a whole lot of convenience to writing and running R code
- IDEs are not the language themselves, they provide a way to interact with the language installed on your computer in a friendly way

Action step: Install R Studio (https://www.rstudio.com/products/rstudio/download/)

Working in R Studio

Key R Studio panes:

- Console: Runs R code, either interactively or via an R script
- Terminal: Convenient terminal application (primarily useful for version control programs like git/GitHub)
- Environment: Objects you've saved to your **working R environment**
- Files: File navigator, useful if you need to figure out where data/R scripts are located
- Plots: Plots generated will populate in this pane – you can also export plots you create using the "Export" button

R Scripting

Use of an **R script** helps keep your code neat and reproducible

- In R Studio: File → New File → R Script
- This is simply a text file with the extension ".R" that will hold all of the R commands we want to run
- Similar to other languages, "#" is reserved for comments

There are 5 main **data types** in R:

- Character (e.g. `"hello world"`)
- Numeric (e.g. `3.14159265`)
- Integer (e.g. `5L`)
- Logical (`TRUE`, `FALSE`)
- Complex (`1i`)

Other special values include missing (`NA`), not-a-number (`NaN`), null (`NULL`), and infinity (`Inf`, `-Inf`)

R Scripting

R also has various **data structures**, but the main ones are:

- Vectors: a collection of elements of one data type
- Lists: a collection of objects of arbitrary types (e.g. the first element could be a vector, the second element could be a matrix, the third element could be a data frame)
- Matrices: a vector with dimensions defined
- Data frames: structure that most resembles a data set, each variable is a single data type
- Factors: a numeric vector that has a label attribute (like a Stata label)

R is also built on **functions**, some of which are provided in the base installation and others that can be installed or created from scratch:

```r
a_function_with_3_args(arg1 = ..., arg2 = ..., arg3 = ...)
```

Functions take **arguments** or inputs, and return **values** (in R parlance) or outputs.

R Basics

Creating a vector, then assigning it to the variable `x` or `y`:

```
# "<-" is the assignment operator
x <- c(1, 2, 3) # c() is a function!
print(x)

## [1] 1 2 3

# short hand for sequences of numbers
y <- 1:3
print(y)

## [1] 1 2 3

# vectors can contain strings
a_string <- c("hello", "world")
print(a_string)

## [1] "hello" "world"
```

R Basics

What type of vector is `what_type`:

```
what_type <- c(1, "hello")
print(what_type)


## [1] "1"     "hello"
```

R Basics

Generating matrices and data frames:

```
my_mat <- matrix(data = c(1:9), nrow = 3, ncol = 3)
print(my_mat)

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

# turning this into a data frame
my_df <- data.frame(my_mat)
print(my_df)

##   X1 X2 X3
## 1  1  4  7
## 2  2  5  8
## 3  3  6  9
```

R Basics

Indexing into 2-dimensional matrices and data frames:

```
my_mat <- matrix(data = c(1:9), nrow = 3, ncol = 3)
print(my_mat[2,3]) # bracket method [row, column]
```

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

```
my_df <- data.frame(my_mat)
print(my_df$X1)  # pulling out a column of a data.frame
```

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

```
print(my_df[,1]) # brackets work here too
```

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

```
print(my_df[2,3])
```

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

R Basics

R does math (e.g. `+, -, *, /, >, >=, ==`):

```
# rnorm() simulates draws from normal distributions
x <- rnorm(n = 10, mean = 0, sd = 3)
print(x)

##  [1] -5.7411212 -0.8227805  3.6030561  2.5185614  5.1350010 -4.7169971
##  [7]  2.8750995  0.3494500  3.2149292  4.7576505

head(x) # what does head do?

## [1] -5.7411212 -0.8227805  3.6030561  2.5185614  5.1350010 -4.7169971

mean(x)

## [1] 1.117285

x[1] + x[2] # indexing for vectors (i.e. a 1-dimension matrix)

## [1] -6.563902
```

R Basics

Many of these operators can also be used to index in more nuanced ways:

```
x <- c(1, 2, 3, 4, 5, 5)

# indexing using operators
x < 3


## [1]  TRUE  TRUE FALSE FALSE FALSE FALSE


x[x < 3]


## [1] 1 2


x[x == 7]


## numeric(0)


x[x == 5]


## [1] 5 5
```

R Basics

Beware default behavior for missing data:

```
x <- c(1, 3, 5, NA)
print(x)
```

```
## [1]  1  3  5 NA
```

```
mean(x)
```

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

```
mean(x, na.rm = TRUE)
```

```
## [1] 3
```

`na.rm = TRUE` is an example of an **optional argument** we can pass to the `mean()` function, telling R to ignore the missing values

R Basics: Check in

Complete the following for understanding. In an R script:

1. Create a vector, $\mathtt{z} = \{0, 0, 4, 6, 7, 10\}$
2. Print the first 3 elements of `z`
3. Calculate the standard deviation of `z` (*hint: look at the sd() function. Entering "?sd" in the R console will bring up its documentation. What arguments does this function take?*)
4. Replace the 4th element of `z` with `NaN` (*hint: combine what you know about indexing with the assignment operator*)
5. Calculate the standard deviation of this new vector
6. Save the R script with this code to your desktop folder

R Basics for Data

A general pipeline for working with data (in R or any other language) is to:

1. Read in the raw data
2. Clean data (saved to a new object)
3. Analyze data
4. Produce results

R Basics for Data

Data can be read into R in various ways depending on the type of file – the base installation includes functions to read in CSV files:

```
my_data <- read.csv("path/to/csv_file.csv")
```

For other types of data files, other packages may need to be installed:

```
install.packages("haven")
library(haven)
my_stata_data <- read_dta("path/to/stata_file.dta")

install.packages("openxlsx")
library(openxlsx)
my_xlsx_data <- read.xlsx("path/to_excel_file.xlsx")
```

R Basics for Data

For example, let's say we have a file called "dat.csv" that's located in my desktop folder, with variables for age and whether an individual is an Aquarius:

```
my_data <- read.csv("/Users/matthewlee/Desktop/dat.csv")
```

Basic summary of the data:

```
summary(my_data)

##       age           aquarius
##  Min.   :18.00   Min.   :0.0000
##  1st Qu.:27.00   1st Qu.:0.0000
##  Median :29.00   Median :0.0000
##  Mean   :29.49   Mean   :0.4998
##  3rd Qu.:32.00   3rd Qu.:1.0000
##  Max.   :41.00   Max.   :1.0000

nrow(my_data)

## [1] 5000
```

R Basics for Data

Let's say we want to generate a new variable, `notAquarius`, the reciprocal of `aquarius`. Good practice is to **not alter the raw data**, so we'll create a copy. In base R we could do:

```
clean_data <- my_data

# dollar sign to index the new and old vars
clean_data$notAquarius <- abs(1 - clean_data$aquarius)
head(clean_data)
```

```
##   age aquarius notAquarius
## 1  28        0           1
## 2  30        1           0
## 3  27        1           0
## 4  34        0           1
## 5  30        0           1
## 6  27        0           1
```

R Basics for Data

The tidyverse collection of packages, including dplyr, provides some convenient methods for working with data beyond base R. Packages must be installed using the install.packages() function, and loaded using the library() function.

```
install.packages("dplyr")
library(dplyr)
```

A key feature of dplyr is the **pipe operator**, denoted by %>%, which allows us to feed the results of one function directly into another without creating a new intermediate object.

```
a_data_frame %>%
    filter(x1 > 5) %>% # filtering to obs where x1 > 5
    summarize(mean_of_y = mean(y)) # mean of y in this subset
```

There is detailed documentation for the many functions in the tidyverse packages, which can be found here at: https://www.tidyverse.org/

R Basics for Data

We can use dplyr functions to generate our notAquarius as well using the mutate() function:

```
clean_data <-
    my_data %>%
    mutate(notAquarius = abs(1 - aquarius))
head(clean_data)
```

```
##   age aquarius notAquarius
## 1  28        0           1
## 2  30        1           0
## 3  27        1           0
## 4  34        0           1
## 5  30        0           1
## 6  27        0           1
```

R Basics for Data

Finally, let's do a basic analysis to compute the proportion of individuals who are an Aquarius by age. Again, `dplyr` functions make this so simple!

```
aqua_age <-
    clean_data %>%
    group_by(age) %>% # group by each unique age
    summarize(prop_aquarius = mean(aquarius))

nrow(aqua_age); head(aqua_age)
```

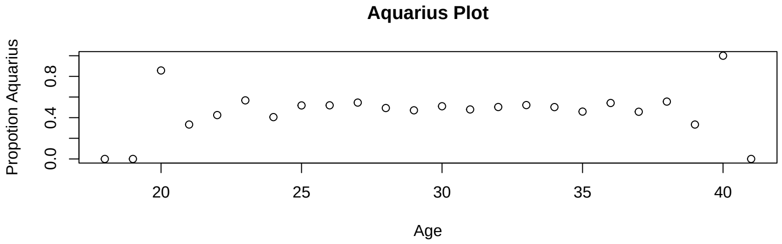
```
## [1] 24
## # A tibble: 6 x 2
##     age prop_aquarius
##   <dbl>         <dbl>
## 1    18         0
## 2    19         0
## 3    20         0.857
## 4    21         0.333
## 5    22         0.425
## 6    23         0.568
```

There are quite a few rows to this output, so maybe a table isn't the best way to visualize

R Basics for Data

Luckily, R has some great plotting functions:

```r
plot(x = aqua_age$age, y = aqua_age$prop_aquarius,
     xlab = "Age", ylab = "Propotion Aquarius", main = "Aquarius Plot")
```



R Basics for Data

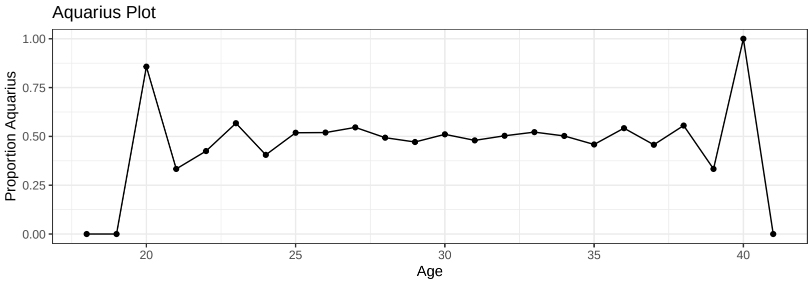
Another popular package for plotting is `ggplot2`:

```r
library(ggplot2)
ggplot(data = aqua_age,
       mapping = aes(x = age, y = prop_aquarius)) +
    geom_point() +
    geom_line() +
    theme_bw() +
    labs(title = "Aquarius Plot", x = "Age", y = "Proportion Aquarius")
```

Elements of ggplots are strung together using the + operator

- Plots start with the `ggplot()` function, which define the data to be plotted as well as the *aesthetics* (e.g. x/y variables, colors, fills, groups)
- `geom_point()` creates a scatter plot
- `geom_line()` then layers on a line plot
- `theme_bw()` adjusts the basic look of the plot
- `labs()` allows us to assign labels

R Basics for Data



ggplot2 is another package that has a bunch of functions to customize your plots. Documentation can be found here

R Basics for Data: Check in

Complete the following for understanding.

1. Download the demographics data from the 2017-18 cycle of NHANES, located here.
2. In an R script:
   2.1 Read in this data set (an .xpt file – you will need a function from the **haven** package, look up the documentation or Google to figure out which one)
   2.2 Pick two variables (one continuous, one categorical) of interest using the documentation posted online, and create a new version of the dataset that only contains these two variables (*hint: the `select()` function from the dplyr package might be useful*)
   2.3 For the categorical variable, the actual values are probably numeric. Using the NHANES documentation, create a new **factor** variable that adds informative labels to the original. (*hint: the `factor()` function might be useful*)
   2.4 Calculate the mean of the continuous variable by levels of the categorical level, and create a plot to show your results.

Analysis Management Tips

Here is a basic organization scheme that I use (prob overkill for classwork):

```
Project folder
 |
 |__ Folder for R code
 |    |
 |    |__ R
 |         |
 |         |__ readclean.R
 |         |__ models.R
 |         |__ create_tables.R
 |         |__ create_figures.R
 |__ out
 |__ run_analysis.R
```

- The R/ subfolder has general functions I write to clean the data, run the models, and generate results
- The out/ folder is where I save all of my output
- The run_analysis.R file is a wrapper script that calls all the functions needed to run through an analysis from start to finish. I usually run this file using the R command `rmarkdown::render(‘‘run_analysis.R’’)`, which will execute all the commands and generate a nice output file

Other R Tips

- If you are unsure what a function does or what arguments it takes, calling "`?mean`" or "`??mean`" will bring up or search help/documentation files
- Stack Overflow and Google are your best friends. If you have a question or see an error message, it's likely that someone else has also had the same issue and posted it to an online forum. Copy/pasting your error messages into Google is a good place to start when troubleshooting
- R is **case sensitive**, so a variable called `newvar` is different than a variable called `NewVar`
- Avoid using R function names for objects, e.g. don't do the following:

```r
mean <- 9
```

- Sometimes restarting R (or your computer) resolves some issues
- R cheat sheets are nice references: https://rstudio.com/resources/cheatsheets/

Troubleshooting/Q&A