Seminar in Data Analysis with R
Week 1
PH 700A

Rick Calvo

Table of contents

0.1 Housekeeping Items

- The Syllabus
- The Schedule
- Attendance
- Assignments
- Canvas
- Use of Generative AI
- GitHub Education

0.2 Welcome to R!

- This course will serve as an introduction to data analysis using **R** and **RStudio**
- No prior coding experience is required
- You should already know about the various statistical analyses

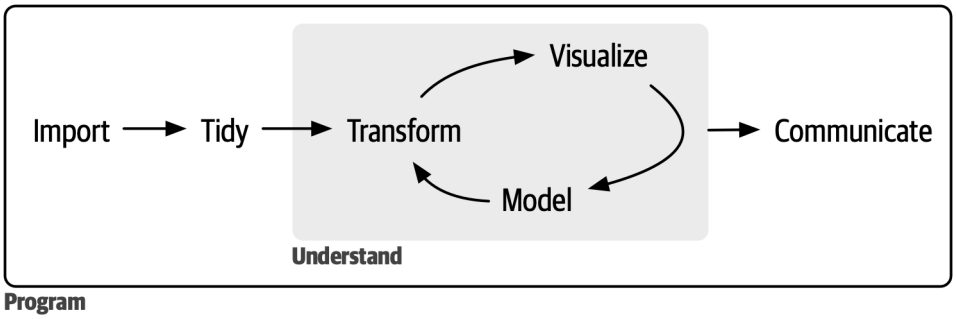
0.2.1 Two options to access the software:

- Supply your own system and install **base R** and **RStudio**
- Sign up for a **Posit Cloud** account under the free plan and access it online

0.3 System Requirements (for PCs)

- A **64-bit operating system** such as Windows 10/11, a recent Linux distribution, or macOS 10.13 or later
- A modern processor with two or more cores
- At least 4gb of memory
- Lots of storage (for datasets, packages, and other files you create)

0.4 Workflow

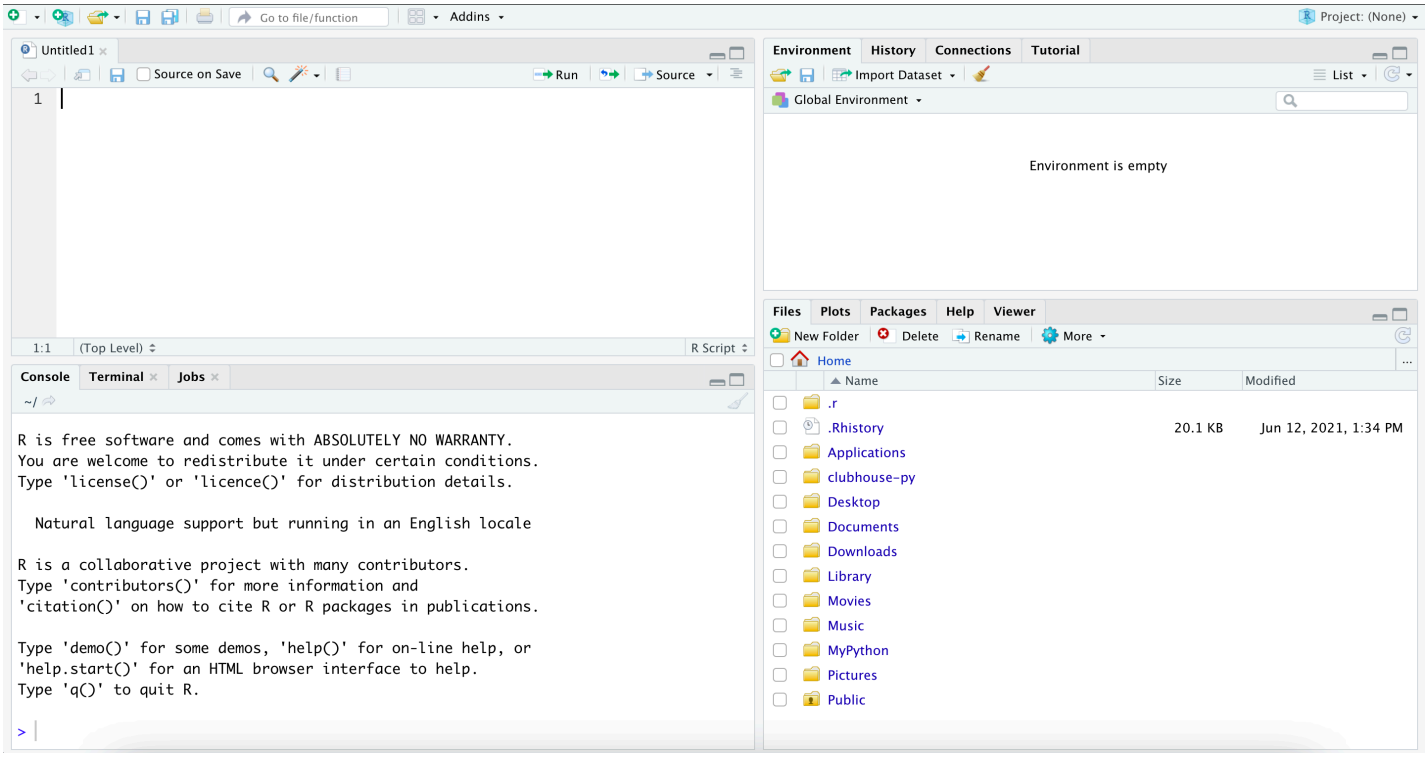


- Import and clean data in a reproducible way
- Analyze data and document your protocol
- Generate results
- Communicate findings

0.5 What's the point?

- Programming is a marketable skill
- Data analysis and public health are intertwined
- Current society is data-oriented
- R is open source and versatile
- Research requires a strong understanding of data management and analysis
- Peer-reviewed publications are a universal currency (in research and academia)

0.6 The RStudio Interface



0.7 Console

- The console is the command-line interpreter for R. You provide it some input and it will display output in a linear fashion
- It is essentially **base R**

0.8 Source

- Holds your commands, comments, notes, and references to data elements (your *code/syntax*)
- Also for viewing raw data in spreadsheet format

> **Note**
>
> - To run code out of a standard syntax file, you highlight the lines you want run and then press the "RUN" button or press Control+Enter
> - Initiating a run without highlighting lines will run the first non-empty line from where the cursor is located
> - Partially highlighting lines and pressing "RUN" will usually result in error

0.9 Environment

- Where the objects that are imported or created through your code are listed for easy reference
- Includes variables you create, lists, data frames, results, etc.
- Anything that you would likely reference in your code will be shown in this area

0.10 Packages

- The source of strength for R
- This window shows all the packages that are currently installed on the system and updates when you install more
- Each package has its own maintenance team and is susceptible to versioning issues

> **Note**
>
> Even though the package may be installed on your system, it is not usable until it's loaded into the R instance.

0.11 Packages (Continued)

We will use the following packages (and more) over the semester

- stats
- tidyverse
- haven
- gtsummary
- ggplot2

1 Getting Started

1.1 Simple Calculations

- At its most basic level, R is a calculator
- Primary arithmetic operators include:
  - + for addition
  - - for subtraction
  - / for division
  - * for multiplication
  - ^ for exponentiation
  - == for equivalence

1.2 Calculation Example

You can type an arithmetic problem directly into the console and it will calculate the answer for you.

$(1 + 2)^3$

```
(1 + 2)^{3}
```

```
[1] 27
```

$43 + 12/32$

```
43+12/32
```

```
[1] 43.375
```

$(43 + 12)/32$

```
(43+12)/32
```

```
[1] 1.71875
```

> **Note**
>
> Make sure you obey the order of operations (PEMDAS)!

1.3 Objects

- Objects are anything in R that store values
- Shown in the Environment window
- Objects can be created through *assignment*:
  - Assignment operators include <- and =
    - **<-** is used for assignment
    - **=** is used for arguments when you're creating or using a function
  - Assigned objects are manipulable and can be self-referential

1.4 Object Assignment Example

```
starting <- 10
```

```
ending <- 9
```

```
wording <- "Week 1"
```

```r
starting <- 10
ending <- 9
wording <- "Week 1"
```

1.5 Assignment vs. Equivalence

Create an object called `a` that is equal to `1`

```
a = 1
```

Evaluate the veracity of equivalence

```
a == 1
```

[1] TRUE

```
a == 2
```

[1] FALSE

1.6 Object Nomenclature

- Object names are case sensitive
- Can be any alphanumeric combination but must start with a letter
- Using underscores _ and periods . are okay
- Per the Google R style guide:
  - New variables should start with a lower case letter and follow **camelCase** or include **underscores** between words

```
thisIsAnExampleOfCamelCase
```

```
anotherVariableInCamelCase
```

1.7 Functions

- Basically tiny programs for performing a specific task
- Functions are called using commands/command names.
  - Always followed by round brackets ()
  - Arguments/parameters to be used by the function are placed within the round brackets

```
print(x)
```

1.8 Arguments/Parameters

- Functions require **input** in the form of "arguments"
- Specific arguments must be stated for a function to work
- Required arguments are called **compulsory arguments**.
- You can view the package help files to see what those compulsory arguments are

> Side-note on Arguments/Parameters
>
> Technically, arguments are the actual data that is passed into the function via the parameter. The parameter is the place-holder item for in the function that is filled by the argument when the function is run. R help files tend to refer to everything inside the round brackets as "arguments".

1.9 Calling Functions Example

Command	Description
`?print`	View the help file on the "print" function
`help(print)`	Another way to view help for the "print" function
`wording <- "Week 1"`	Assigns the text string in the quotes to the object, "wording"
`print(wording)`	Issues the command to perform "print" using the "wording" as the compulsory argument

```
?print
help(print)
wording <- "Week 1"
print(wording)
```

```
[1] "Week 1"
```

1.10 Example Function

```
PercentChange <- function(a, b) {

(b-a)/a

}

answer <- PercentChange(5,10)

answer
```

```r
PercentChange <- function(a, b) {
  (b-a)/a
}

answer <- PercentChange(5,10)
answer
```

```
[1] 1
```

The value we get is 1, meaning that we experienced a positive 100% change.

Now that the function is defined, we can use it anytime!

1.11 Example Function (Continued)

`answer2 <-PercentChange(10,9) answer2`

```r
answer2 <-PercentChange(10,9)
answer2
```

```
[1] -0.1
```

We get is -0.1, meaning the value decreased by 10%.

1.12 More on Compulsory Arguments

- Must be referenced as shown in the help file

> **Tip**
>
> If `PercentChange` requires two arguments `PercentChange(a,b)` then both `a` and `b` must be specified either directly (with an equals sign = for assignment) or in their order as listed in the help file/definition.

1.13 Compulsory Arguments Example

`answer2 <-PercentChange(10,9)`

10 is listed first, so 10 takes the "a" parameter and 9 takes the "b" parameter

`answer3 <-PercentChange(b=ending, a=starting)`

"b" is assigned to "ending" and "a" is assigned to "starting"

```r
answer2 <-PercentChange(10,9)
answer2
```

```
[1] -0.1
```

```
answer3 <-PercentChange(b=ending, a=starting)
answer3
```

```
[1] -0.1
```

1.14 Data Types

- Numeric
  - Integers
  - Doubles
- Character
  - Text strings of any length
- Logical
  - Binary true/false

`mode(var)` tells you what the variable type is

`typeof(var)` tells you the sub-type, if applicable

1.15 Data Structures

Groupings of values

- Vector
- Factor
- Matrix
- Data frame

The data structure is determined by what and how many kinds of data types it is composed of

1.16 Vectors

- A sequence of data elements of the same type (one dimensional list)

`dogAges <- c(5, 4, 5, 7, 9, 11, 15, 2)`

Vector "dogAges" contains values that are numeric integers

c stands for "combining". R combines the values in the round brackets to form the vector.

`dogOrder <- 1:8` is the same as

`dogOrder <- c(1,2,3,4,5,6,7,8)`

Another vector "dogOrder" contains a sequence of values 1-8.

```
dogAges <- c(5, 4, 5, 7, 9, 11, 15, 2)
dogOrder <- c(1,2,3,4,5,6,7,8)
```

1.17 Vectors (Continued)

Vectors can also be composed of strings

```
dogNames <-c("Rufio", "Chubbs", "Buster", "Stormy", "Saki", "Spike", "Flower", "Frida")
```

```
length(dogNames)
```

```
dogNames <-c("Rufio", "Chubbs", "Buster", "Stormy", "Saki", "Spike", "Flower", "Frida")
length(dogNames)
```

```
[1] 8
```

We have a vector containing several dog names.

The `length` function tells us how many items are in the vector.

1.18 Applying Names to Vectors

- We can apply the names from **dogNames** to other vectors assuming that the order of values is consistent

```
names(dogAges) <- dogNames
```

```
head(dogAges)
```

```
names(dogAges) <- dogNames
head(dogAges)
```

```
 Rufio Chubbs Buster Stormy   Saki  Spike
     5      4      5      7      9     11
```

```
tail(dogAges)
```

```
Buster Stormy   Saki  Spike Flower  Frida
     5      7      9     11     15      2
```

head() allows to view the beginning parts in the vector

- For long vectors, the display of values will be truncated
- Use **tail()** to view the end parts of the vector

1.19 Addressing Vector Elements

- If we want to view, extract, or perform additional functions on **specific** parts of the vector, we can address the element(s) we need

```
head(dogAges)
```

```
 Rufio Chubbs Buster Stormy   Saki  Spike
     5      4      5      7      9     11
```

`dogAges[1]` returns the first element in the vector

```
dogAges[1]
```

```
Rufio
    5
```

You can alternatively query by name.

```
dogAges["Saki"]
```

```
dogAges["Saki"]
```

```
Saki
   9
```

1.20 Addressing Vector Elements

You can also query by a specified range or multiple elements

```
dogAges[2:4]
```

```
dogAges[c("Chubbs","Buster","Stormy")]
```

```
dogAges[2:4]
```

```
Chubbs Buster Stormy
     4      5      7
```

```
dogAges[c("Chubbs","Buster","Stormy")]
```

```
Chubbs Buster Stormy
     4      5      7
```

- Both methods return 3 values; The first by the order and the second by specifically stating the names.

1.21 Modifying Vectors

Now that we can address elements specifically, we can modify or remove them as needed.

```
dogAges[5] <- 8
```

```
dogAges[5] <- 8
dogAges
```

```
 Rufio Chubbs Buster Stormy   Saki  Spike Flower  Frida
     5      4      5      7      8     11     15      2
```

- This changed element #5 (Saki) from 9 to 8.

We can remove an element by inputing a "-" before the order number and reassign the vector to the object name.

```
dogAges[-5]
```

```
dogAges[-5]
```

```
 Rufio Chubbs Buster Stormy  Spike Flower  Frida
     5      4      5      7     11     15      2
```

```
dogAges <- dogAges[-5]
dogAges
```

```
 Rufio Chubbs Buster Stormy  Spike Flower  Frida
     5      4      5      7     11     15      2
```

```
spike <- "Spike"
# dogAges[-spike]
# How to Cite strings while modifying vectors
```

- Saki was removed from the vector.

1.22 Combining Vectors

Appending vectors combines two or more to lengthen the "list" of elements.

```
oceansideDogAges <- c(5, 5, 4, 3, 10, 6, 12, 15, 11)
```

```
sanDiegoDogAges <- c(3, 5, 3, 2, 5, 9, 8, 10, 9, 1, 1, 3)
```

```
elCajonDogAges <- c(1, 1, 3, 7, 9, 5, 3, 5, 9, 9, 1, 2)
```

```
sdCountyDogAges <-(c(oceansideDogAges, sanDiegoDogAges, elCajonDogAges)) length(sdCountyDogAges)
```

```
oceansideDogAges <- c(5, 5, 4, 3, 10, 6, 12, 15, 11)
sanDiegoDogAges <- c(3, 5, 3, 2, 5, 9, 8, 10, 9, 1, 1, 3)
elCajonDogAges <- c(1, 1, 3, 7, 9, 5, 3, 5, 9, 9, 1, 2)
sdCountyDogAges <-(c(oceansideDogAges, sanDiegoDogAges, elCajonDogAges))
length(sdCountyDogAges)
```

```
[1] 33
```

- There are 33 elements in the composite vector called, "sdCountyDogAges"

1.23 Factors

- A one-dimensional list of discrete values
- Akin to dummy variables for statistical models

```
furLength <- factor(c("long", "short", "medium", "long"))
```

```
str(furLength)
```

```
furLength <- factor(c("long", "short", "medium", "long"))
str(furLength)
```

```
 Factor w/ 3 levels "long","medium",..: 1 3 2 1
```

- The `str` function shows you the structure of the factor

1.24 Factors Continued

- By default, factors are unordered
- Create a new factor to denote the order based on an old factor

```
furLength_ord <- factor(furLength, levels=c("short", "medium", "long"), ordered=TRUE)
```

```
min(furLength_ord)
```

```
max(furLength_ord)
```

```
furLength_ord <- factor(furLength, levels=c("short", "medium", "long"), ordered=TRUE)
min(furLength_ord)
```

```
[1] short
Levels: short < medium < long
```

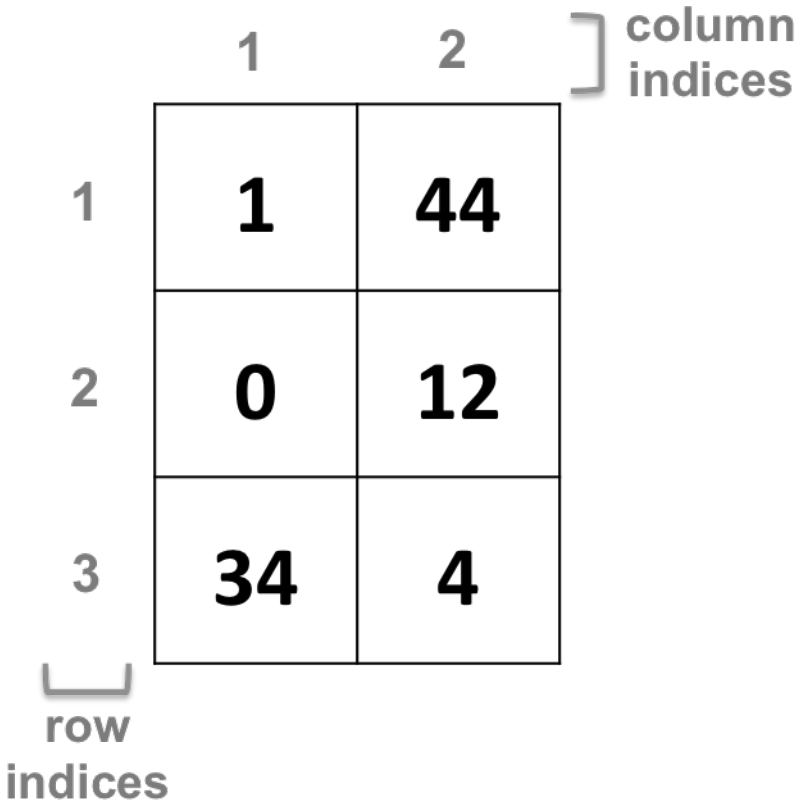
```
max(furLength_ord)
```

```
[1] long
Levels: short < medium < long
```

- min() and max() tell you the order and the minimum and maximum values of the factor, respectively

1.25 Matricies

- Two-dimensional data objects
- Values are referenced by row and column (like coordinates)



`MatrixName[a,b]` is the syntax, where **a** is the row and **b** is the column

- There will be more on matrices when we address model diagnostics and data reduction

1.26 Data Frames

- Data Frames ("df") are a mixture of various data types
- Looks like a traditional spreadsheet
- Can be assembled using other objects, or imported from other sources
- Much more on this at a later

1.27 Resources

- https://google.github.io/styleguide/Rguide.html
- https://cran.r-project.org/web/packages/available_packages_by_name.html
- http://www.cookbook-r.com/
- https://r4ds.hadley.nz/