Statistics for Social Research

Week 1: Introduction

Wenhao Jiang

Department of Sociology
New York University

September 9, 2022

Main Goals

Main Goals

- Introduce `R` and `RStudio`
- Answer questions from the textbook
- Recap important concepts in lecture

[1]Acknowledgement: The slides of introducing R is largely based on Applied Statistics with R from https://daviddalpiaz.github.io/appliedstats/

Introduction of R and RStudio

Getting Started

- Why R
  1. If you have used Excel before - R is more handy and efficient in both data manipulation and statistical analysis

Getting Started

- Why R
  1. If you have used Excel before - R is more handy and efficient in both data manipulation and statistical analysis
  2. If you have used Stata, SAS, or SPSS before - R is free, with professionals developing open-sourced new packages every day

Getting Started

- Why R
  1. If you have used Excel before - R is more handy and efficient in both data manipulation and statistical analysis
  2. If you have used Stata, SAS, or SPSS before - R is free, with professionals developing open-sourced new packages every day
  3. If you have used Python before - R and its packages are more user-friendly to social scientists who are not professional programmers

Getting Started

- Why R
  1. If you have used Excel before - R is more handy and efficient in both data manipulation and statistical analysis
  2. If you have used Stata, SAS, or SPSS before - R is free, with professionals developing open-sourced new packages every day
  3. If you have used Python before - R and its packages are more user-friendly to social scientists who are not professional programmers
  4. Support of LaTeX in RMarkdown to write academic papers, make academic slides with R **codes knitted**.

Getting Started

- Why R
  1. If you have used Excel before - R is more handy and efficient in both data manipulation and statistical analysis
  2. If you have used Stata, SAS, or SPSS before - R is free, with professionals developing open-sourced new packages every day
  3. If you have used Python before - R and its packages are more user-friendly to social scientists who are not professional programmers
  4. Support of LaTeX in RMarkdown to write academic papers, make academic slides with R **codes knitted**.
- R is useful in both academia and in companies that require statistical/data analysis

Getting Started

- Why `RStudio`
  - RStudio, like most IDEs, provides a graphical interface to `R`, making it more user-friendly, and providing dozens of useful features

Getting Started

- Install in the following order
  1. R: https://www.r-project.org/
  2. RStudio: https://www.rstudio.com/
- Now open RStudio

Quick tour of RStudio

- There are four panels
  1. Source: write your own codes
  2. Console: view outputs
  3. Environment/History: view stored datasets and variables
  4. Files/Plots/Packages/Help
- In the Source panel
  - Write your own codes
  - Save your code in `.R` file
  - Click `Run` command to run your entire code
- In the console panel
  - After clicking `Run` in the source panel, your code is evaluated
  - You can directly type your code here to implement

Basic Calculations

To get started, we'll use `R` like a simple calculator.

Addition, Subtraction, Multiplication and Division

Math	R	Result
$3 + 2$	`3 + 2`	5
$3 - 2$	`3 - 2`	1
$3 \cdot 2$	`3 * 2`	6
$3/2$	`3 / 2`	1.5

```r
1 + 3
```

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

Exponents

Math	R	Result
3^2	`3 ^ 2`	9
$2^{(-3)}$	`2 ^ (-3)`	0.125
$100^{1/2}$	`100 ^ (1 / 2)`	10
$\sqrt{100}$	`sqrt(100)`	10

Mathematical Constants

Math	R	Result
π	`pi`	3.1415927
e	`exp(1)`	2.7182818

Logarithms

- Note that we will use ln and log interchangeably to mean the natural logarithm.
- There is no `ln()` in `R`, instead it uses `log()` to mean the natural logarithm.

Math	`R`	Result
$\log(e)$	`log(exp(1))`	1
$\log_{10}(1000)$	`log10(1000)`	3
$\log_2(8)$	`log2(8)`	3
$\log_4(16)$	`log(16, base = 4)`	2

Trigonometry

Math	R	Result
$\sin(\pi/2)$	`sin(pi / 2)`	1
$\cos(0)$	`cos(0)`	1

Descriptive Statistic

Math	R
$\frac{1}{n}\sum_{i=1}^{n} x_i$	`mean(x)`
$\frac{1}{n-1}\sum_{i=1}^{n} (x_i - \bar{x})^2$	`var(x)`

Getting Help

- In using R as a calculator, we have seen a number of functions: `sqrt()`, `exp()`, `log()` and `sin()`.
- To get documentation about a function in R, simply put a question mark in front of the function name or call function `help()` and `RStudio` will display the documentation, for example:

```
?log
?sin
help(log)
help(sin)
```

Installing Packages

- One of the main strengths of R as an open-source project is its package system
- To install a package, use the `install.packages()` function.
  - Think of this as buying a recipe book from the store, bringing it home, and putting it on your shelf.

```
install.packages("ggplot2")
```

- Once a package is installed, it must be loaded into your current R session before being used.
  - Think of this as taking the book off of the shelf and opening it up to read.

```
library(ggplot2)
```

Installing Packages

Packages like `ggplot2` are extremely powerful

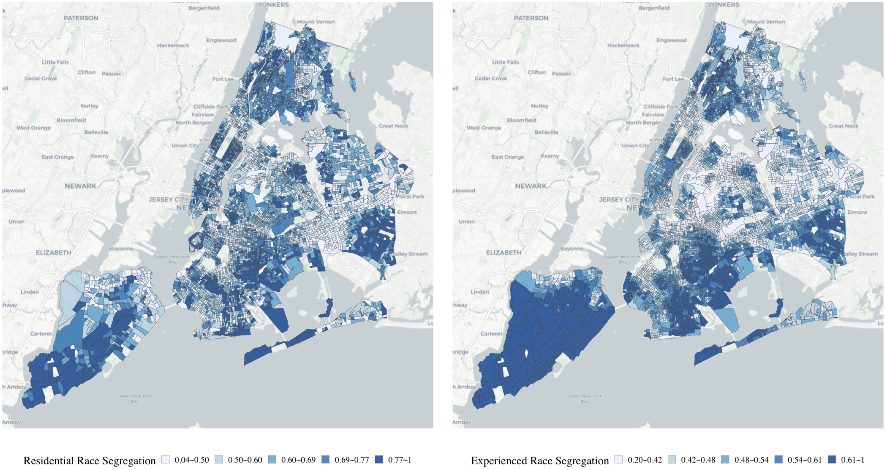

Figure 1: Residential v. Experienced Segregation in New York City, 2018-2020

- Once you close R, all the packages are closed and put back on the imaginary shelf.
- The next time you open R, you do not have to install the package again, but you do have to load any packages you intend to use by invoking `library()`.

Helps

- The RStudio team has developed a number of "cheatsheets" for working with both `R` and RStudio.
- This particular cheatsheet for Base `R` will summarize many of the concepts in this document.

Data

Data Types

R has a number of basic data *types*.

- Numeric
  - Also known as Double. The default type when dealing with numbers.
  - Examples: `1`, `1.0`, `42.5`
  - Store *numerical* variables
- Logical
  - Two possible values: `TRUE` and `FALSE`
  - You can also use `T` and `F`
  - `NA` is also considered logical
- Character
  - Examples: `"a"`, `"Statistics"`, `"1 plus 2."`
  - Usually store *categorical* variables

Data Structures

- R also has a number of basic data *structures*.
- A data structure is either
  - homogeneous (all elements are of the same data type)
  - heterogeneous (elements can be of more than one data type).

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

Vector

Vectors

Basics of vectors

- Many operations in R make heavy use of **vectors**.
  - Vectors in R are indexed starting at 1.
- The most common way to create a vector in R is using the `c()` function, which is short for "combine".

```
c(1, 3, 5, 7, 8, 9)
```

```
## [1] 1 3 5 7 8 9
```

Basics of vectors

- If we would like to store this vector in a **variable** we can do so with the **assignment** operator =
  - The variable `x` now holds the vector we just created, and we can access the vector by typing `x`

```
x = c(1, 3, 5, 7, 8, 9)
x
```

```
## [1] 1 3 5 7 8 9
```

```
# The following does the same thing
x <- c(1, 3, 5, 7, 8, 9)
x
```

```
## [1] 1 3 5 7 8 9
```

- The operator = and <- work as an assignment operator.
- In R code the line starting with # is `comment`, which is ignored when you run the code.

A sequence of numbers as a vector

- The quickest and easiest way to do this is with the `:` operator, which creates a sequence of integers between two specified integers.

```
y <- 1:10
y
```

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

Useful functions for creating vectors

- Use the `seq()` function for a more general sequence.

```
seq(from = 1.5, to = 2.5, by = 0.1)
```

```
##  [1] 1.5 1.6 1.7 1.8 1.9 2.0 2.1 2.2 2.3 2.4 2.5
```

- Here, the input labels `from`, `to`, and `by` are optional.

```
seq(1.5, 2.5, 0.1)
```

```
##  [1] 1.5 1.6 1.7 1.8 1.9 2.0 2.1 2.2 2.3 2.4 2.5
```

- The `rep()` function repeat a single value a number of times.

```
rep("A", times = 10)
```

```
##  [1] "A" "A" "A" "A" "A" "A" "A" "A" "A" "A"
```

- The `rep()` function can be used to repeat a vector some number of times.

```
rep(x, times = 3)
```

```
##  [1] 1 3 5 7 8 9 1 3 5 7 8 9 1 3 5 7 8 9
```

- We have now seen four different ways to create vectors:
  1. `c()`
  2. `:`
  3. `seq()`
  4. `rep()`
- They are often used together

```
c(x, rep(seq(1, 5, 2), 2), c(1, 2), 2:4)
```

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

Length

- The length of a vector can be obtained with the `length()` function.

```
length(x)
```

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

```
length(y)
```

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

Subsetting

- Use square brackets, `[]`, to obtain a subset of a vector.
- We see that `x[1]` returns the first element.

```
x
```

```
## [1] 1 3 5 7 8 9
```

```
x[1]
```

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

```
x[3]
```

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

- We can also exclude certain indexes, in this case the second element.

```
x[-2]
```

```
## [1] 1 5 7 8 9
```

- We can subset based on a vector of indices.

```
x[1:3]
```

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

```
x[c(1,3,4)]
```

```
## [1] 1 5 7
```

- We could also use a vector of logical values

```
z <- c(TRUE, TRUE, FALSE, TRUE, TRUE, FALSE)
x[z]
```

```
## [1] 1 3 7 8
```

- Note that the length of `z` must of the same as `x`
- Combining `subset` with `length` could be very useful in summarizing the proportion of e.g. missing values

```
x[z] <- NA
x
```

```
## [1] NA NA  5 NA NA  9
```

```
length(x[is.na(x)])
```

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

Vectorization

- One of the biggest strengths of R is its use of vectorized operations.
  - Frequently the lack of understanding of this concept leads of a belief that R is *slow*.
- When a function like `log()` is called on a vector `x`, a vector is returned which has applied the function to each element of the vector `x`.

```
x = 1:10
x + 1
```

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

```
2 * x
```

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

```r
2 ^ x
```

```
##  [1]    2    4    8   16   32   64  128  256  512 1024
```

```r
round(sqrt(x), 2)
```

```
##  [1] 1.00 1.41 1.73 2.00 2.24 2.45 2.65 2.83 3.00 3.16
```

```r
round(log(x), 2)
```

```
##  [1] 0.00 0.69 1.10 1.39 1.61 1.79 1.95 2.08 2.20 2.30
```

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`	xequal 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

- Logical operators are vectorized.

```
x = c(1, 3, 5, 7, 8, 9)
x > 3
```

```
## [1] FALSE FALSE  TRUE  TRUE  TRUE  TRUE
```

```
x < 3
```

```
## [1]  TRUE FALSE FALSE FALSE FALSE FALSE
```

```
x == 3
```

```
## [1] FALSE  TRUE FALSE FALSE FALSE FALSE
```

```
x != 3
```

```
## [1]  TRUE FALSE  TRUE  TRUE  TRUE  TRUE
```

```r
x == 3 & x != 3
```

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

```r
x == 3 | x != 3
```

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

```r
is.na(x)
```

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

- This is extremely useful for subsetting.

```
x[x > 3]
```

```
## [1] 5 7 8 9
```

```
x[x != 3]
```

```
## [1] 1 5 7 8 9
```

Short exercise

1. Create the vector $z = (1, 2, 1, 2, 1, 2)$, which has the same length as x.
2. Pick up the elements of x which corresponds to 1 in the vector z, i.e., in position 1, 3, and 5.

```
z <- c(1,2,1,2,1,2)
x[z == 1]
```

```
## [1] 1 5 8
```

Q & A

- Are there any questions from lectures, textbooks, or `R`?