Lecture 1: Introduction to R

Course Logistics, Objects, and Packages

Yujin Jeong

STATS 195

Course Logistics

STATS 195 will run for 4 weeks: January 17, 19, 24, 26, 31, February 2, 7, 9.

- Lectures: Tue, Thu 12:00 PM - 1:20 PM at STLC 115
- Office hours: Thu 1:30 PM - 3:30 PM at Sequoia 105
- Class website: https://stanford-stats195.github.io/winter2023/

Grading (Satisfactory/No Credit):

- Homework assignments (30%)
- Final project proposal (10%)
- Final project report (60%)

Assignments

Homework:

- work individually
- 3 assignments due by Jan 25 (Wed), Feb 1 (Wed), Feb 8 (Wed) 11:59pm

Final project:

- work in groups up to 2 students
- project proposal due by Feb 1 (Wed) 11:59pm
- final project report due by Feb 15 (Wed) 11:59pm

Late day policy:

- no later than 2 days post due date; 20% penalty per day
- up to three late days are forgiven at the end of the quarter

Course Objectives

Through this class, students will be able to :

- Navigate the R ecosystem at a basic level (RStudio, CRAN, R Markdown)
- Import, transform, plot, and save the data
- Perform simple data modeling and statistical analyses in R

Overview of the course:

- Week 1: Introduction to R
- Week 2: Data visualization with **ggplot2**
- Week 3: Data transformation with **dplyr**
- Week 4: Statistical modeling and testing

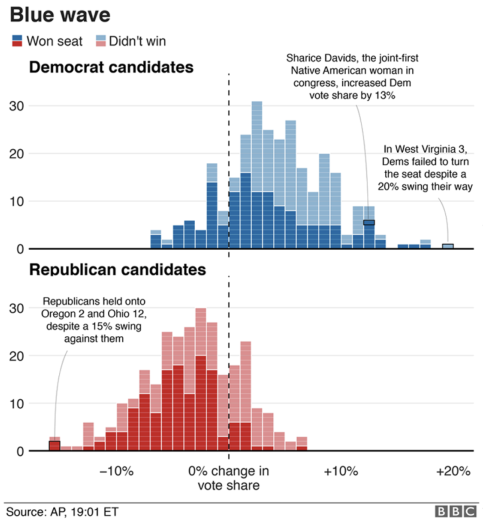
The R language

R is an open-source language for statistical computing and graphics.

Why learn R?

R was specifically designed for statistics and data analysis. R offers:

- a large collection of tools for data analysis (which other programming languages don’t have)
- a suite of libraries for matrix computations
- facilities for generating high-quality graphics and data display



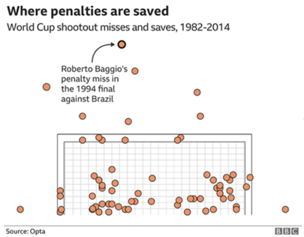
Blue wave

Sharice Davids, the joint-first Native American woman in congress, increased Dem vote share by 13%

In West Virginia 3, Dems failed to turn the seat despite a 20% swing their way

Republicans held onto Oregon 2 and Ohio 12, despite a 15% swing against them

Source: AP, 19:01 ET

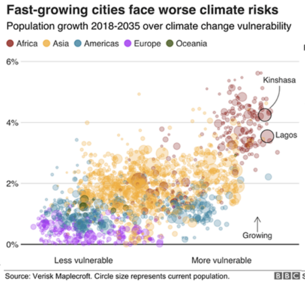


Where penalties are saved

World Cup shootout misses and saves, 1982-2014

Roberto Baggio's penalty miss in the 1994 final against Brazil

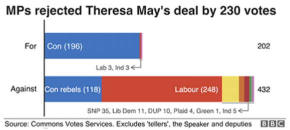
Source: Opta



Fast-growing cities face worse climate risks

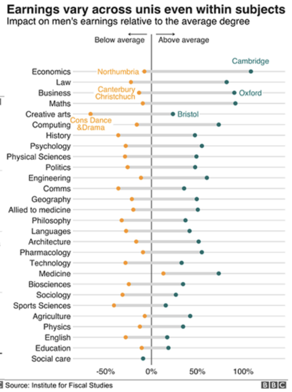
Population growth 2018-2035 over climate change vulnerability

Source: Verisk Maplecroft. Circle size represents current population.



MPs rejected Theresa May's deal by 230 votes

Source: Commons Votes Services. Excludes 'tellers', the Speaker and deputies



Earnings vary across unis even within subjects

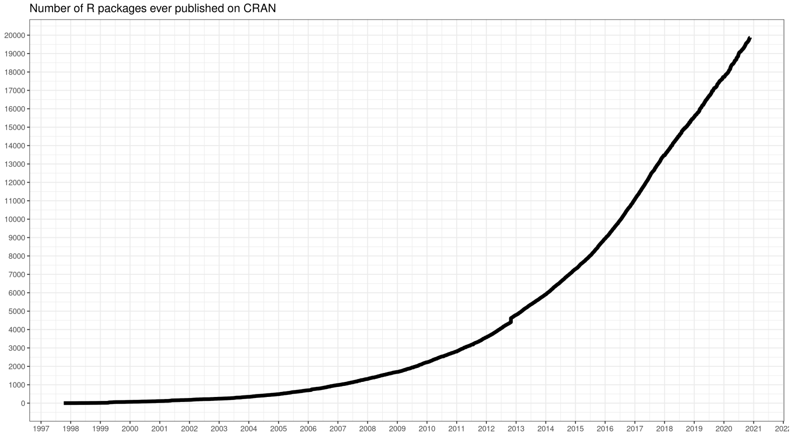
Impact on men's earnings relative to the average degree

Source: Institute for Fiscal Studies

Packages

A package is a collection of R functions, data, and documentations.

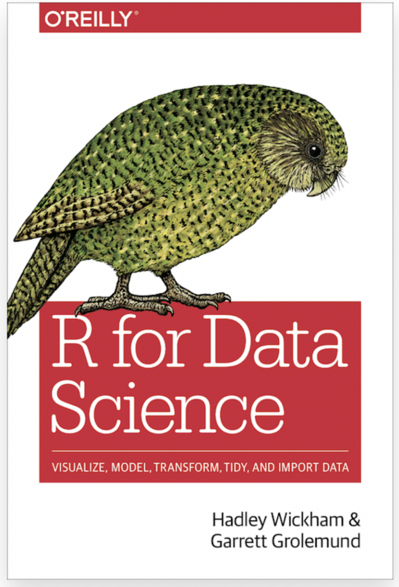
- R comes in-built with a core set of packages. e.g. base, datasets, graphics, stats
- Most user-created packages are available for free on The Comprehensive R Archive Network (CRAN).
- As of December 2022, there are 18980 packages on CRAN.



- Many user-created R packages contain implementations of cutting edge statistics methods.

Textbook

There is no textbook for this class. However, **R for Data Science** by Garrett Grolemund and Hadley Wickham (Freely available at: https://r4ds.had.co.nz) is recommended as an aid to understanding the course material. Some of the course material is based on this book.



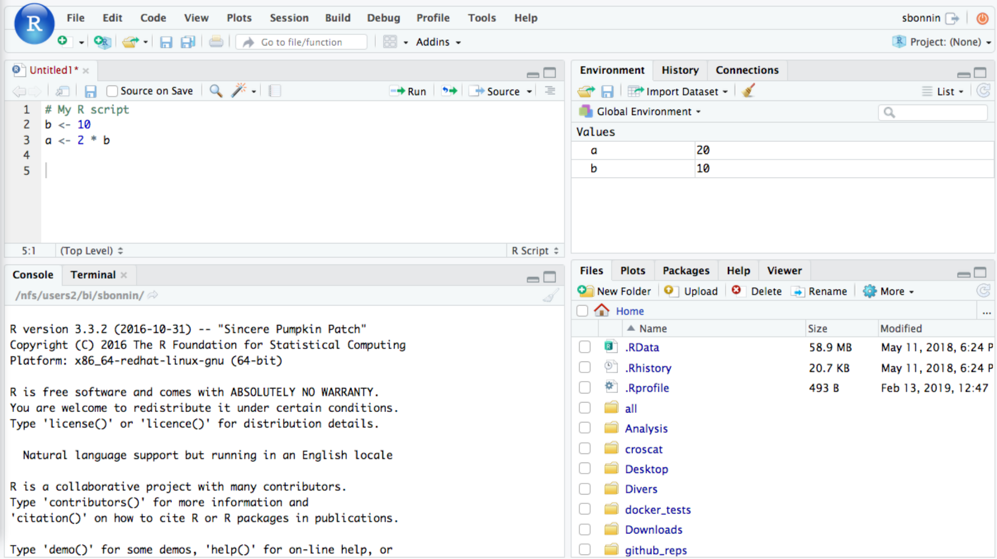
Installing R and RStudio

Installing R and RStudio

- Download R from the CRAN website.
- Download RStudio (an integrated development environment (IDE) for R) from the official website.

RStudio layout

- top-left: scripts
- bottom-left: R console
- top-right: objects, history and environment
- bottom-right: tree of folders, graph window, packages, help window, viewer



```
# My R script
b <- 10
a <- 2 * b
```

```
/nfs/users2/bi/sbonnin/

R version 3.3.2 (2016-10-31) -- "Sincere Pumpkin Patch"
Copyright (C) 2016 The R Foundation for Statistical Computing
Platform: x86_64-redhat-linux-gnu (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
```

Values

a	20
b	10

Name	Size	Modified
.RData	58.9 MB	May 11, 2018, 6:24 P
.Rhistory	20.7 KB	May 11, 2018, 6:24 P
.Rprofile	493 B	Feb 13, 2019, 12:47
all		
Analysis		
croscat		
Desktop		
Divers		
docker_tests		
Downloads		
github_reps		

Basics of Coding in R

R as a calculator

You can use R as a high-powered calculator. For example,

```r
4 * exp(10) + sqrt(2)
```

```
## [1] 88107.28
```

Some other classical arithmetic:

```r
5^2 # power
5 ** 2 # power (used a lot in other programming languages)
5 %% 2 # 5 modulo 2
5 %/% 2  # Integer division
5 / 2 # Division
```

```
## [1] 25
## [1] 25
## [1] 1
## [1] 2
## [1] 2.5
```

Data Types in R

The most common data types in R are 'double', 'character', 'integer', and 'logical'.

```
typeof(7.3)
typeof("text")
typeof(0)
typeof(FALSE)
```

```
## [1] "double"
## [1] "character"
## [1] "double"
## [1] "logical"
```

Data Types in R

We can change the type of a variable to type `` `x` `` using the function `` `as.x` ``. For example,

```
as.character(6507232300)
typeof(6507232300)
typeof(as.character(6507232300))
```

```
## [1] "6507232300"
## [1] "double"
## [1] "character"
```

We can also change variables to numbers or boolean variables.

```
as.numeric("123")
as.logical(123)
as.logical(0)
```

```
## [1] 123
## [1] TRUE
## [1] FALSE
```

Objects and Assignments

Often, we want to store the result of a computation so that we can use it later. R allows us to do this by variable assignment. Variable assignment can be done using the following operators: `` `=` ``, `` `<-` ``:

```
x = 2
```

```
x <- 2
```

We can use `` `x` `` in computations:

```
x^2 + 3*x
```

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

We can also reassign `` `x` `` to a different value:

```
x <- x^2
print(x)
```

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

Naming variables

Variable names must start with a letter and can only contain letters, numbers, `` `_` `` and `` `.` ``.

Some words are reserved in R and cannot be used as object names:

- `` `Inf` `` stands for positive infinity. R will return this when the value is too big.
- `` `NULL` `` denotes a null object which is often used as undeclared function argument.
- `` `NA` `` represents a missing value.
- `` `NaN` `` means "Not a Number". R will return this when a computation is undefined, e.g. 0/0.

Vectors

The **vector** is a 1-dimensional array whose entries are the same type. For example, the following code produces a vector containing the numbers 1, 2, and 3:

```r
vec <- c(1, 2, 3)
vec
```

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

Vectors

Typing out all the elements can be tedious. Sometimes there are shortcuts we can use. The following code assigns a vector of the numbers 1 to 20 to `` `vec` ``:

```
vec <- 1:20
vec
```

```
##  [1]   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20
```

What if we only want even numbers from 1 to 50 (inclusive)? We can manipulate vectors using arithmetic operations. Note that **arithmetic operations happen element-wise**.

```
even <- 1:10 * 2
even
```

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

```
odd <- even - 1
odd
```

```
## [1]  1  3  5  7  9 11 13 15 17 19
```

Vectors

We can use the `c()` function to combine ("concatenate") several small vectors into one large vector.

```
z <- 1:5
z <- c(z, 3, z)
z
```

```
## [1] 1 2 3 4 5 3 1 2 3 4 5
```

We can use the `seq` function to generate numerical sequences.

```
seq(from = 5, to = 10, by = 0.5)
```

```
## [1]  5.0  5.5  6.0  6.5  7.0  7.5  8.0  8.5  9.0  9.5 10.0
```

```
seq(from = 5, to = 10, length.out = 10)
```

```
## [1]  5.000000  5.555556  6.111111  6.666667  7.222222  7.777778  8.333333  8.888889  9.444444 10.000000
```

Vectors Indexing

R allows us to access individual elements in a vector. Unlike many other programming languages, indexing begins at 1, not 0. To return the first even number:

```r
even[1]
```

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

To extract the 5th to 9th even number (inclusive):

```r
even[5:9]
```

```
## [1] 10 12 14 16 18
```

To extract just the 5th and 9th even numbers:

```r
even[c(5,9)]
```

```
## [1]  10 18
```

Vectors Indexing

What if I want all even numbers except the first two? We can use negative indexing:

```r
even[-c(1,2)]
```

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

We can use the `` `length` `` function to figure out how many elements there are in a vector. What happens if I try to extract an element from an index greater than its length?

```r
length(odd)
odd[11]
```

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

Matrices

Matrices are the 2-dimensional analogs of vectors. As with vectors, elements of matrices have to be of the same type. Use the `` `matrix()` `` command to change a vector into a matrix:

```
A <- matrix(LETTERS, nrow = 2)
A
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13]
## [1,] "A"  "C"  "E"  "G"  "I"  "K"  "M"  "O"  "Q"  "S"   "U"   "W"   "Y"  
## [2,] "B"  "D"  "F"  "H"  "J"  "L"  "N"  "P"  "R"  "T"   "V"   "X"   "Z"  
```

Notice that R takes the elements in the vector you give it and fills in the matrix column by column. If we want the elements to be filled in by row instead, we have to put in a `` `byrow = TRUE` `` argument:

```
B <- matrix(letters, nrow = 2, byrow = TRUE)
B
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13]
## [1,] "a"  "b"  "c"  "d"  "e"  "f"  "g"  "h"  "i"  "j"   "k"   "l"   "m"  
## [2,] "n"  "o"  "p"  "q"  "r"  "s"  "t"  "u"  "v"  "w"   "x"   "y"   "z"  
```

Matrices

To get the dimensions of the matrix, we can use the `` `dim` ``, `` `nrow` `` and `` `ncol` `` functions:

```
dim(B)
```

```
## [1]  2 13
```

To access the element in the `` `i` ``th row and `` `j` `` column for the matrix `` `B` ``, use the index `` `i,j` ``:

```
B[1, 2]  # for the element in the 1st row and 2nd column
```

```
## [1] "b"
```

```
A[2, ]
A[2, 1:3]
A[, 2]
```

```
## [1] "B" "D" "F" "H" "J" "L" "N" "P" "R" "T" "V" "X" "Z"
## [1] "B" "D" "F"
## [1] "C" "D"
```

Lists

To have elements on different types in one data structure, we can use a list, which we create with `` `list()` ``. We can think of a list as a collection of key-value pairs. Keys should be strings.

```
person <- list(name = "John Doe", age = 26)
person
```

```
## $name
## [1] "John Doe"
##
## $age
## [1] 26
```

The `` `str` `` function can be used to inspect what is inside `` `person` ``:

```
str(person)
```

```
## List of 2
##  $ name: chr "John Doe"
##  $ age : num 26
```

Lists

To access the `name` element in `person`, we have 2 options:

```
person[["name"]]
```

```
person$name
```

The elements of a list can be anything, even another data structure! Let’s add the names of John’s children to the `person` object:

```
person$children = c("Ross", "Robert")
str(person)
```

```
## List of 3
##  $ name    : chr "John Doe"
##  $ age     : num 26
##  $ children: chr [1:2] "Ross" "Robert"
```

Dataframes

Dataframes are a special type of list in R. The values of the dataframe must all be vectors of the same length. You can think of each vector as a feature and each row as instance or subject in your dataset.

```
df <- data.frame(height = c(60,60,62,64,70,72,75),
                 ID = seq_len(7),
                 eyes = c("Green","Blue","Brown","Brown","Blue","Green","Brown"))
```

We can access information like it was a list:

```
is.list(df)
df$height
df$eyes[3]
```

```
## [1] TRUE
## [1] 60 60 62 64 70 72 75
## [1] "Brown"
```

Dataframes

This object will be heavily used in R for anything data science related.

```
# print first 6 rows of the data frame
head(df)
```

	height	ID	eyes
	<dbl>	<int>	<chr>
1	60	1	Green
2	60	2	Blue
3	62	3	Brown
4	64	4	Brown
5	70	5	Blue
6	72	6	Green

6 rows

```
# get the structure of the data frame
str(df)
```

```
## 'data.frame':  7 obs. of  3 variables:
##  $ height: num  60 60 62 64 70 72 75
##  $ ID    : int  1 2 3 4 5 6 7
##  $ eyes  : chr  "Green" "Blue" "Brown" "Brown" ...
```

Packages

Other people have been writing super useful functions, and uploading useful data, and designing other useful data types, so we don’t have to. They publish that work as R packages, and we can install and load them to use ourselves. Some packages are so useful that they come pre-installed and pre-loaded.

List of useful packages:

- `` `dplyr` ``: Transform data
- `` `ggplot2` ``: Make nice plots
- `` `readr` ``: Import data into R
- `` `tidyr` ``: Clean data
- `` `stringr` ``: Tools for working with character strings and regular expressions
- `` `lubridate` ``: Make working with dates and times easier
- `` `caret` ``: Tools for training regression and classification models
- `` `glmnet` ``: Advanced regression methods
- `` `maps` ``, `` `ggmap` ``: Tools for plotting spatial data

Packages

To install a package we can use the packages tab in Rstudio or the function 'install.packages'. To load a package, we mostly use 'library'.

```
install.packages('ISLR')
```

To access specific functions or other objects in a package, we can use "::", this will avoid confusion if a two or more packages use the same name for different functions. You can even do this to use functions or objects in a package without loading it (as long as it is installed).

```
head(ISLR::College)
```

If you do load it, you can call those objects directly:

```
library(ISLR)
head(College)
```