

Documenting and Data Analysis in R

R Basics

Maxwel Coura Oliveira, PhD

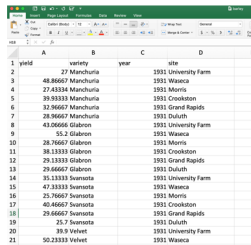
University of Wisconsin-Madison

Outline

- How to install R and Rstudio
- Intro to R, RStudio and Markdown
- Data types
- Importing datasets



+ - * / ^ <- [] \$ > <
== ! %in% & |

A screenshot of the RStudio interface showing a data frame with 21 rows and 5 columns. The columns are labeled A, B, C, D, and E. The data represents corn yield experiments across different varieties, years, and sites.

	A	B	C	D	E
1	yield	variety	year	site	
2		27 Manchuria	1931	University Farm	
3		48.86667 Manchuria	1931	Waseca	
4		27.43334 Manchuria	1931	Morris	
5		39.93333 Manchuria	1931	Crookston	
6		32.96667 Manchuria	1931	Grand Rapids	
7		28.96667 Manchuria	1931	Duluth	
8		43.06666 Glabron	1931	University Farm	
9		55.2 Glabron	1931	Waseca	
10		28.76667 Glabron	1931	Morris	
11		38.13333 Glabron	1931	Crookston	
12		29.13333 Glabron	1931	Grand Rapids	
13		29.46667 Glabron	1931	Duluth	
14		35.13333 Swansota	1931	University Farm	
15		47.33333 Swansota	1931	Waseca	
16		25.76667 Swansota	1931	Morris	
17		40.46667 Swansota	1931	Crookston	
18		29.46667 Swansota	1931	Grand Rapids	
19		25.7 Swansota	1931	Duluth	
20		39.9 Velvet	1931	University Farm	
21		50.23333 Velvet	1931	Waseca	

Whats is R? RStudio?

- **R** – a programming language + software that interprets it
- **RStudio** – popular software to write R scripts and interact with the R software

Why learn R?

- Free, open source, cross platform
 - 10,000+ “packages”
 - Works on many data types
- Statistical data analysis
- Produced high-quality graphics
- Reproducibility and repeatability
- Write documents and manuscripts



How to download R? RStudio?



[Home]

Download
CRAN

R Project

About R
Logo
Contributors
What's New?
Reporting Bugs
Conferences
Search
Get Involved: Mailing Lists
Developer Pages
R Blog

R Foundation

Foundation
Board
Members
Donors
Donate

Help With R
Getting help

The R Project for Statistical Computing

Getting Started

R is a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS. To [download R](#), please choose your preferred CRAN mirror.

If you have questions about R like how to download and install the software, or what the license terms are, please read our [answers to frequently asked questions](#) before you send an email.

News

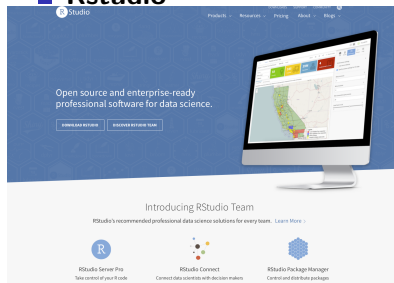
- **R version 3.6.1 (Action of the Toss)** has been released on 2019-07-05.
- **useR!** 2020 will take place in St. Louis, Missouri, USA.
- **R version 3.6.3 (Great Truthy)** has been released on 2019-03-11.
- The R Foundation Conference Committee has released a [call for proposals](#) to host useR! 2020 in North America.
- You can now support the R Foundation with a renewable subscription as a [supporting member](#).
- The R Foundation has been awarded the Personality/Organization of the year 2018 award by the professional association of German market and social researchers.

News via Twitter



Video tutorial

Rstudio

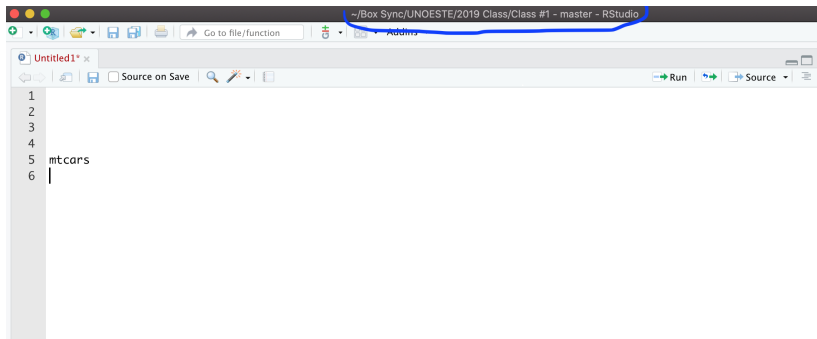


Setup a working directory

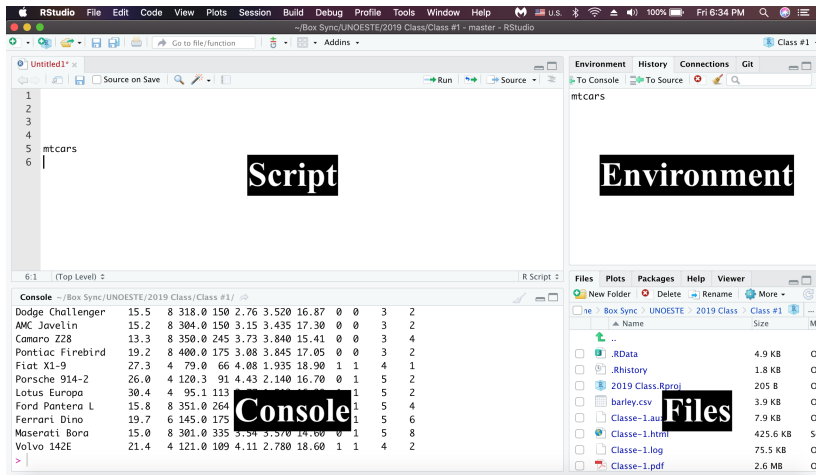
- Open RStudio
- *File > New project > New directory > Empty project*
- Enter a name for this new folder: r-basics
- Choose a convenient location:
- ~/ is the Documents folder on the computer
- Click “*Create project*”

Create a new R script

- File > New File > R script (.R or .Rmd)
- Save it in your project directory
- Look on the top left of the R Studio window to see where it's saved

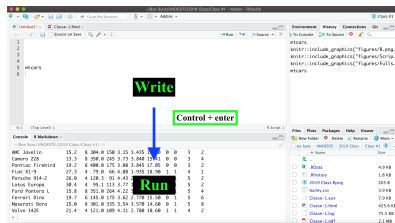


RStudio interface



Script vs Console

- Both accept commands
- Console: runs the commands
- Script: commands you want to save for later;
 - Must be run in console
 - Ctrl+enter to run



Let's start coding!

- Operators: Arithmetic, assignment, extraction, logical
- Functions: names, arguments, output
- Data types: classes, vectors, data frames

Let's start coding!

Operators	What it does	Symbol
Arithmetic	Math on numbers	+ - * / ^
Assignment	Creates objects (left) with	<-
Extraction	Take out or replace part of an object	[] \$
Logical	Compares values, returns TRUE/FALSE	> < == ! %in% &

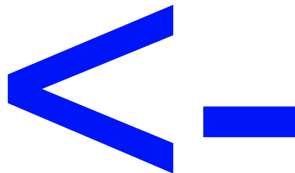
Let's start coding!

- Does math
 - Add: $2+2$
 - Subtract: $3-1$
 - Multiply: $4*4$
 - Divide: $5/2$
- Sends results to the console
 - CTRL+Enter



Assignment operator

- Saves values into objects
 - `object <- value`
 - `weightkg <- 55`
- Overwrites previous values
- Combine with arithmetic operators:
 - `weightlb <- 2.2*weightkg`



Exercise 1: Operators

What are the values of each variable after each statement?

```
mass <- 89 # mass?
```

```
age <- 35 # age?
```

```
mass <- mass * 2.0 # mass?
```

```
age <- age - 20 # age?
```

```
mass_index <- mass/age #mass_index?
```

Functions and arguments

- A sequence of instructions that perform a task
 - Have names
 - Accepts arguments (input)
 - Return a value (output)

Input	Output
<code>sqrt(9)</code>	3
<code>round(3.14159)</code>	3
<code>round(x=3.14159), digits=2</code>	3

Getting help

- Documentation

```
?round # Opens a page for round
```

```
args(round) # display arguments
```

```
## function (x, digits = 0)  
## NULL
```

- Google “R +”function name”

- Other websites

- Stack overflow (Q&A)
- R bloggers (tutorials)

Data types

- R guesses what type of data is stored in an object

- Basic types:

Numeric

Character

Logical

- Can be easy to tell

Examples:

- `x <- 32` (Numeric)
- `x <- "car"` (Character)
- `x <- TRUE` (Logical)

Data structures

Data structure	Description	Function
vector	Multiple values of the same type	c(), vector
factor	Multiple integers with text labels	factor
data frame	Multiple vectors of the same length grouped to make columns	read.csv(), data frame

Vector

- Most common data type
- Series of one data type
- Concatenate function: `c()`

Input: values separately by commas

Output: a vector object

```
# Example: a list of yields  
yield_ha <- c(3000, 2890, 3100, 2990)  
# Example: a list of cars  
cars <- c("audi", "toyota", "ford")
```

Inspecting vectors

- Vectors have characteristics:
 - **Length**: number of values
 - **Class**: type of values

```
length(yield_ha)  # Try with length(cars)
```

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

```
class(cars)       # Try with class(yield_ha)
```

```
## [1] "character"
```

```
str(yield_ha)     # Try with str(cars)
```

```
##  num [1:4] 3000 2890 3100 2990
```

Adding values to a vector

- Use an existing vector as an argument to `c()`
- Put it in the order you want them to appear in the output vector

```
# Add to the end of the vector  
yield_ha <- c(yield_ha, 3315)
```

```
# Add to the beginning of the vector  
yield_ha <- c(3050, yield_ha)
```

Class coercion

- What happens if you mix types?
- R converts to a type that works for all elements
- Use `class()` to see what R picked

Type	As character	As numeric	As logical
logical	"TRUE"	1	TRUE
numeric	"35"	35	NA
character	"Paulista"	NA	NA

Exercise 2: Vectors

- What types are these vectors?

```
num_char <- c(1, 2, 3, "a")  
num_logical <- c(1, 2, 3, TRUE)  
char_logical <- c("a", "b", "c", TRUE)  
tricky <- c(1, 2, 3, "4")
```

Hint: use `class()`

Factors

- Represent categorical data
 - Stored as integers with text labels
 - Data frames convert character columns to factors
- *factor()* - create a factor
- Create a character vector

```
sex <- c("male", "female", "female", "male")
```

- Change vector to a factor

```
sex <- factor(sex)
```


Levels

- Unique text labels of a factor object
- `levels()` - displays labels
- `nlevels()` - displays number of levels

Function	Output
<code>levels(sex)</code>	"female", "male"
<code>nlevels(sex)</code>	2
<code>factor(sex, levels = c("male", "female"))</code>	"male", "female"
<code>levels(sex)</code>	

Exercise 3: Types

```
ranks <- c("2", "5", "7", "3", "3")  
f_ranks <- factor(ranks)  
n_ranks <- as.numeric(f_ranks)
```

- What result do you expect to get?
- What do you get when you run the code?

Subsetting vectors

- Subset by position
- Syntax: square brackets []
- Combine with c()

```
animals<-c("cat", "dog", "pig")
```

```
animals[2] #Display second value
```

```
## [1] "dog"
```

```
animals[c(3,2)] #Display multiple values
```

```
## [1] "pig" "dog"
```

```
animals[c(1,2,3,2,1)] #Display repeated values
```

```
## [1] "cat" "dog" "pig" "dog" "cat"
```

Logical expressions

- Make comparisons
- Evaluates each element in a vector against a value
- Output: **TRUE** or **FALSE**
 - For each vector value

Logical expressions

Logical operator	Meaning
>	Greater than
<	Less than
==	Equal to
!=	Not equal to
&	and
	or
!	not
%in%	Contained in

Example: logical expressions

- Create a weight variable:

```
biomass_g <- c(22, 33, 37, 51, 59)
```

- Evaluate each weight:

```
biomass_h <- biomass_g > 50  
biomass_h
```

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

Conditional subsetting

- Keep **TRUE**, drop **FALSE**
- Input: a logical expression
- **Output**: vector with elements that match the logical expression
- Subset using **TRUE**/**FALSE** vector

```
biomass_g[biomass_h]
```

```
## [1] 51 59
```

- Same as

```
biomass_g[biomass_g>50]
```

```
## [1] 51 59
```

Combining logical expressions

- Combine multiple conditionals
- `|` = or (either)
- `&` = and (both)
- Biomass under 30 or over 50:

```
biomass_g[biomass_g<30 | biomass_g>50]
```

```
## [1] 22 51 59
```

- Biomass over 30 and under 50:

```
biomass_g[biomass_g>30 & biomass_g<50]
```

```
## [1] 33 37
```


Conditional subsetting: characters (==)

- == operator
- Compares each value in a vector with a character string
- Combine with | for multiple

Make a character vector

```
crops <- c("corn", "soybean", "wheat", "alfalfa")
```

Crops that are corn

```
crops[crops=="corn"]
```

```
## [1] "corn"
```

Crops that are corn or cats

```
crops[crops=="corn" | crops=="wheat"]
```

```
## [1] "corn" "wheat"
```

Conditional subsetting: characters (%in%)

- **%in%** operator
- Selects elements of the first vector that are in the second vector
- **Input:** vectors
- **Output:** a **TRUE**/**FALSE** list

Which values in animals are in the right hand vector?

```
crops %in% c("corn", "soybean", "hemp", "wheat", "beans")
```

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

Use **TRUE**/**FALSE** vector to subset

```
crops[crops %in% c("corn", "soybean", "hemp", "wheat", "beans")]
```

```
## [1] "corn"      "soybean" "wheat"
```

Missing data

- NA - harder to overlook missing data
- Argument: `na.rm = TRUE`

```
na.rm = TRUE #Ignores missing data
```

```
heights <- c(2, 4, 4, NA, 6, 7) #create a dataset
```

- Mean of a missing value?

```
mean(heights)
```

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

- Remove the missing data

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

```
## [1] 4.6
```

Remove missing data

- `is.na()` - Returns **TRUE** if the value is NA
- `complete.cases()` - returns **FALSE** if missing
- `na.omit()` - returns object with missing values removed

Remove NAs 3 ways:

```
heights[!is.na(heights)]
```

```
## [1] 2 4 4 6 7
```

```
heights[complete.cases(heights)]
```

```
## [1] 2 4 4 6 7
```

```
na.omit(heights)
```

```
## [1] 2 4 4 6 7
```

```
## attr("na.action")
```

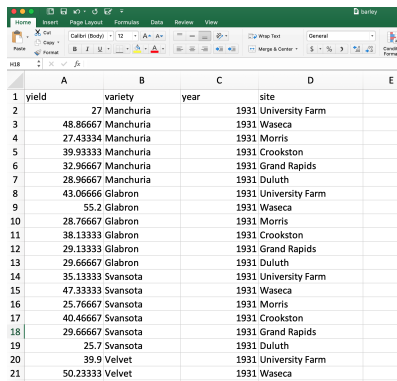
```
## [1] 4
```

```
## attr("class")
```

```
## [1] "omit"
```

Starting with data!

- How to load data tables into R
- Data set: barley yield in Minnesota, USA
- Stored in a **.csv** file
- **Rows**: observations of individual treatments
- **Columns**: Variables that describe the study
- `factor()` - create a factor



	A	B	C	D	E
1	yield	variety	year	site	
2		27 Manchuria		1931 University Farm	
3	48.86667	Manchuria		1931 Waseca	
4	27.43334	Manchuria		1931 Morris	
5	39.93333	Manchuria		1931 Crookston	
6	32.96667	Manchuria		1931 Grand Rapids	
7	28.96667	Manchuria		1931 Duluth	
8	43.06666	Glabron		1931 University Farm	
9	55.2	Glabron		1931 Waseca	
10	28.76667	Glabron		1931 Morris	
11	38.13333	Glabron		1931 Crookston	
12	29.13333	Glabron		1931 Grand Rapids	
13	29.66667	Glabron		1931 Duluth	
14	35.13333	Svansota		1931 University Farm	
15	47.33333	Svansota		1931 Waseca	
16	25.76667	Svansota		1931 Morris	
17	40.46667	Svansota		1931 Crookston	
18	29.66667	Svansota		1931 Grand Rapids	
19	25.7	Svansota		1931 Duluth	
20	39.9	Velvet		1931 University Farm	
21	50.23333	Velvet		1931 Waseca	

Download the data file

- Download data here - Link
- Right click (**barley.csv**) > Download Linked File
- Save it to your project folder

Tables to data frames

- R can read data tables
- Read tables using `read.csv()` or `read.csv2()`
- **Input:** a file name
- **Output:** table stored in a data frame

```
barley <- read.csv("barley.csv")  
barley
```

	yield	variety	year	site
## 1	27.00000	Manchuria	1931	University Farm
## 2	48.86667	Manchuria	1931	Waseca
## 3	27.43334	Manchuria	1931	Morris
## 4	39.93333	Manchuria	1931	Crookston
## 5	32.96667	Manchuria	1931	Grand Rapids
## 6	28.96667	Manchuria	1931	Duluth
## 7	43.06666	Glabron	1931	University Farm
## 8	55.20000	Glabron	1931	Waseca
## 9	28.76667	Glabron	1931	Morris
## 10	38.13333	Glabron	1931	Crookston
## 11	29.13333	Glabron	1931	Grand Rapids
## 12	29.66667	Glabron	1931	Duluth
## 13	35.13333	Svansota	1931	University Farm

Storing data in data frame

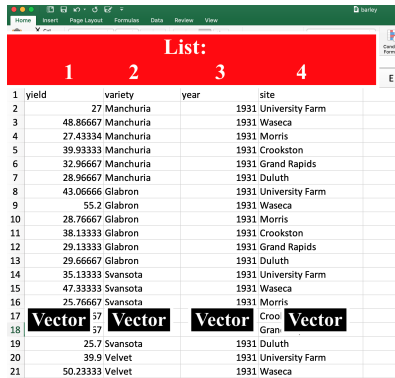
- 1 - Rows = observations
- 2 - Columns = variables
- 3 - All values in a column must be the same data type
- 4 - Must have same # rows in each column

Structure of a data frame

- A list of vectors
- Each column
- Is a vector
- Has a name
- Has a data type
- Is a subject to coercion
- List: any data type - every column can have a different data type

Structure of a data frame

- Stored in a **.csv** file
- **Rows**: observations of individual treatments
- **Columns**: Variables that describe the study
- **factor()** - create a factor



The screenshot shows an Excel spreadsheet with a data frame. The first 16 rows contain data, and the last 5 rows are highlighted with 'Vector' labels. The columns are labeled 1 through 4, corresponding to the variables yield, variety, year, and site.

	1	2	3	4
1	yield	variety	year	site
2		27 Manchuria		1931 University Farm
3		48.86667 Manchuria		1931 Waseca
4		27.43334 Manchuria		1931 Morris
5		39.93333 Manchuria		1931 Crookston
6		32.96667 Manchuria		1931 Grand Rapids
7		28.96667 Manchuria		1931 Duluth
8		43.06666 Glabron		1931 University Farm
9		55.2 Glabron		1931 Waseca
10		28.76667 Glabron		1931 Morris
11		38.13333 Glabron		1931 Crookston
12		29.13333 Glabron		1931 Grand Rapids
13		29.66667 Glabron		1931 Duluth
14		35.13333 Svansota		1931 University Farm
15		47.33333 Svansota		1931 Waseca
16		25.76667 Svansota		1931 Morris
17	Vector	57	Vector	Vector
18		57		Crookston
19		25.7 Svansota		1931 Duluth
20		39.9 Velvet		1931 University Farm
21		50.23333 Velvet		1931 Waseca

Inspecting data frames

Function	Output
<code>class</code>	Class of the object
<code>str</code>	structure: # rows, cols, data types
<code>dim</code>	look at dimensions of your data frame
<code>head</code>	look at first 6 rows (all columns)
<code>ls</code>	list objects returning vector names
<code>nrow/ncol</code>	number of rows/columns
<code>names</code>	column names
<code>summary</code>	summary stats for each column

Subsetting data frames

- Use the extraction operator `[]`
- Row column format: `data[row, column]`
- Select entire row/col: `data[, column]`
- Ranges: `data[a:b, column]`

Subsetting data frames

■ First row, second col:

```
barley[1,2]
```

```
## [1] Manchuria  
## 10 Levels: Glabron Manchuria No. 457 No. 462 No. 475 Peatland ... Wisconsin No. 38
```

■ First row, all cols:

```
barley[1,]
```

```
##   yield  variety year      site  
## 1    27 Manchuria 1931 University Farm
```

■ Rows 1-3, 3 column:

```
barley[1:3, 3]
```

```
## [1] 1931 1931 1931
```

Subsetting data frames

■ First column, all rows:

```
barley[,3]
```

```
## [1] 1931 1931 1931 1931 1931 1931 1931 1931 1931 1931 1931 1931 1931 1931 1931
## [15] 1931 1931 1931 1931 1931 1931 1931 1931 1931 1931 1931 1931 1931 1931 1931
## [29] 1931 1931 1931 1931 1931 1931 1931 1931 1931 1931 1931 1931 1931 1931 1931
## [43] 1931 1931 1931 1931 1931 1931 1931 1931 1931 1931 1931 1931 1931 1931 1931
## [57] 1931 1931 1931 1931 1932 1932 1932 1932 1932 1932 1932 1932 1932 1932 1932
## [71] 1932 1932 1932 1932 1932 1932 1932 1932 1932 1932 1932 1932 1932 1932 1932
## [85] 1932 1932 1932 1932 1932 1932 1932 1932 1932 1932 1932 1932 1932 1932 1932
## [99] 1932 1932 1932 1932 1932 1932 1932 1932 1932 1932 1932 1932 1932 1932 1932
## [113] 1932 1932 1932 1932 1932 1932 1932 1932
```

Subsetting data frames

- `barley["variety"]`
- `barley[, "variety"]`
- `barley[["variety"]]`
- `barley$variety`

Result is a `data.frame`

Result is a `vector`

Result is a `vector`

Result is a `vector`

Exercise 4: Subsetting

- 1) Create a data frame (`barley70`) containing only the observations from rows 1 to 70 of the `surveys` dataset.
- 2) Use `nrow()` to subset the last row in `barley70`.
- 3) Use `nrow()` to extract the row that is in the middle `barley70`. Store in a variable called `barleymid`.

Saving Data as .csv

- `write.csv()` or `write.csv2()`
- Input: data frame, destination file
- Output: a file to the specified location
- `write.csv(x = barley70, file = "barley70.csv")`

Need help

- **Email:** max.oliveira@wisc.edu
- Base R Cheat sheet: [Link](#)