

Course introduction, some computer science terminology, and using RStudio

Computer Programming Fundamentals and
Bioinformatics in R
Fall 2018, Lecture 1

This course should help you:

- Get your research done!
- Use R more effectively
- Write your own custom analyses from scratch
- Create robust, versatile code that you can use across multiple projects
- Understand computer science concepts such that you can teach yourself other languages
- Process bioinformatics data in R

Who I am

- Lindsay Clark
- Research Specialist, Dept. of Crop Sciences
- Ph.D. Genetics, UC Davis 2011
- Creator of the R packages polysat and polyRAD and the Python program TagDigger

Course outline

- Weeks 1-7: programming fundamentals and graphics
- Weeks 8-11: making reproducible and shareable code
- Weeks 12-15: bioinformatics

Course format

- Lectures Tues 9:00-10:50, broken up by demonstrations and mini-exercises.
- Lab Thurs 9:00-9:50, turn in for grade.
- Office hours Thur 10-11
- Non-graded homeworks for extra practice
- In-class midterm Oct 23 on material through Oct 11.

Final project

- Create a small R package that does something useful for your research
- Can work alone or in groups; scope of the package should be proportional to the number of people working on it
- Proposals due Nov. 13
- Feel free to consult with me early and often about your ideas

Project example

- LI-COR 6400: clamps onto the leaf of a plant and takes physiological readings (esp. relating to photosynthesis)
- Outputs data to spreadsheet:



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC	
1	OPEN 6.3.2																													
2	Tue Sep 22 2015 10:57:27																													
3	Unit=	PSC-2696																												
4	LCF=	LCF-1133																												
5	LCFCals=	-1.99	-0.31	-2988																										
6	LightSour	6400-40 FI	1	0.16																										
7	A/D AvgTi	4																												
8	Config=	/User/Configs/UserPrefs/clark.xml																												
9	Remark=	day 15 warm																												
10																														
11	Obs	HHMMSS	genotype	treatment	FTime	EBal?	Photo	Cond	Ci	FCnt	DCnt	Fo	Fm	Fo'	Fm'	Fs	Fv/Fm	Fv'/Fm'	PhiPS2	Adark	RedAbs	BlueAbs	%Blue	LeafAbs	PhiCO2	qP	qN	NPQ	ParIn	
12	in	in	in	in	in	in	out	out	out	in	in	in	in	in	in	in	out	out	out	in	in	in	in	out	out	out	out	out	in	
13	Remark=	"11:17:38 LCF Lamp: ParIn -> 1000 uml, with blue = 10 percent, actinic on"																												
14	Remark=	"11:17:38 CO2 Mixer: CO2R -> 400 uml"																												
15	Remark=	"11:17:38 Coolers: Off"																												
16	Remark=	"11:17:38 Flow: Fixed -> 400 umol/s"																												
17	Remark=	"11:38:13 CO2 Mixer -> OFF"																												
18	Remark=	"11:53:01 CO2 Mixer: CO2R -> 400 uml"																												
19	Remark=	"11:57:53 Flow: Fixed -> 400 umol/s"																												
20	Remark=	"11:58:06 Coolers: Tleaf -> 25.00 C"																												
21	Remark=	"12:48:46 Flow: Fixed -> 400 umol/s"																												
22	Remark=	"12:50:05 Fs=303 Msr=3 Mod=0.25 Filter=5 Gain=10"																												
23	Remark=	"12:50:05 MPF#1 8151 um, Fmax=349 Int=357 +/- 0 SIp= 7 +/- 0 Rmp=100 P1=200 P2=600 P3=300 Int=9 Mod=20 Filter=50"																												
24	Remark=	"12:50:06 Fm'=357"																												
25	Remark=	"11:50:14 Dark#1 Fmin=254 Dur=6 FarRed=8 Pre=1 Post=4 Mod=0.25 Filter=5"																												
26	Remark=	"12:50:15 Fo'=254"																												
27	1	12:50:33	pms-347 r:w		6776.5	0	6.812169	0.051513	179.1075	1	1	0	0	254.1499	357.5414	303.3118	#DIV/0!	0.289173	0.151674		-1	0.87	0.92	10.10221	0.875051	0.008945	0.524507	1.406813	-1	998.0
28	Remark=	"13:54:50 LCF Lamp: ParIn -> 1000 uml, with blue = 10 percent, actinic on"																												
29	Remark=	"13:54:51 CO2 Mixer: CO2R -> 400 uml"																												
30	Remark=	"13:54:51 Coolers: Tleaf -> 25.00 C"																												
31	Remark=	"13:54:51 Flow: Fixed -> 400 umol/s"																												
32	Remark=	"14:36:56 Flow: Fixed -> 400 umol/s"																												
33	Remark=	"14:38:26 Fc=458 Msr=3 Mod=0.25 Filter=5 Gain=10"																												

← File header

← Column headers

← Machine- or user-generated remarks with time stamps

← Data

Functions in a one-person project:

- Import data. Stores remarks and measurements in separate tables.
- Make a publication-quality A/Ci curves (rate of photosynthesis vs. concentration of CO₂ in leaf)
- Estimate Rubisco enzyme activity from the measured parameters

A second person on this project might make functions that:

- Filter out poor quality measurements from the data (negative C_i , stomatal conductance too low)
- Show remarks to user and interactively ask them whether to throw out measurements before or after each remark
- Export data in a more readable format

A third person on the project might:

- Add some additional analysis or plotting functions
- Design a new class specifically for storing LI-COR data (we will talk more about what classes are)
- Make a nice tutorial for the whole package

Let's take a look at what's going
on in RStudio.

What is an interpreter?

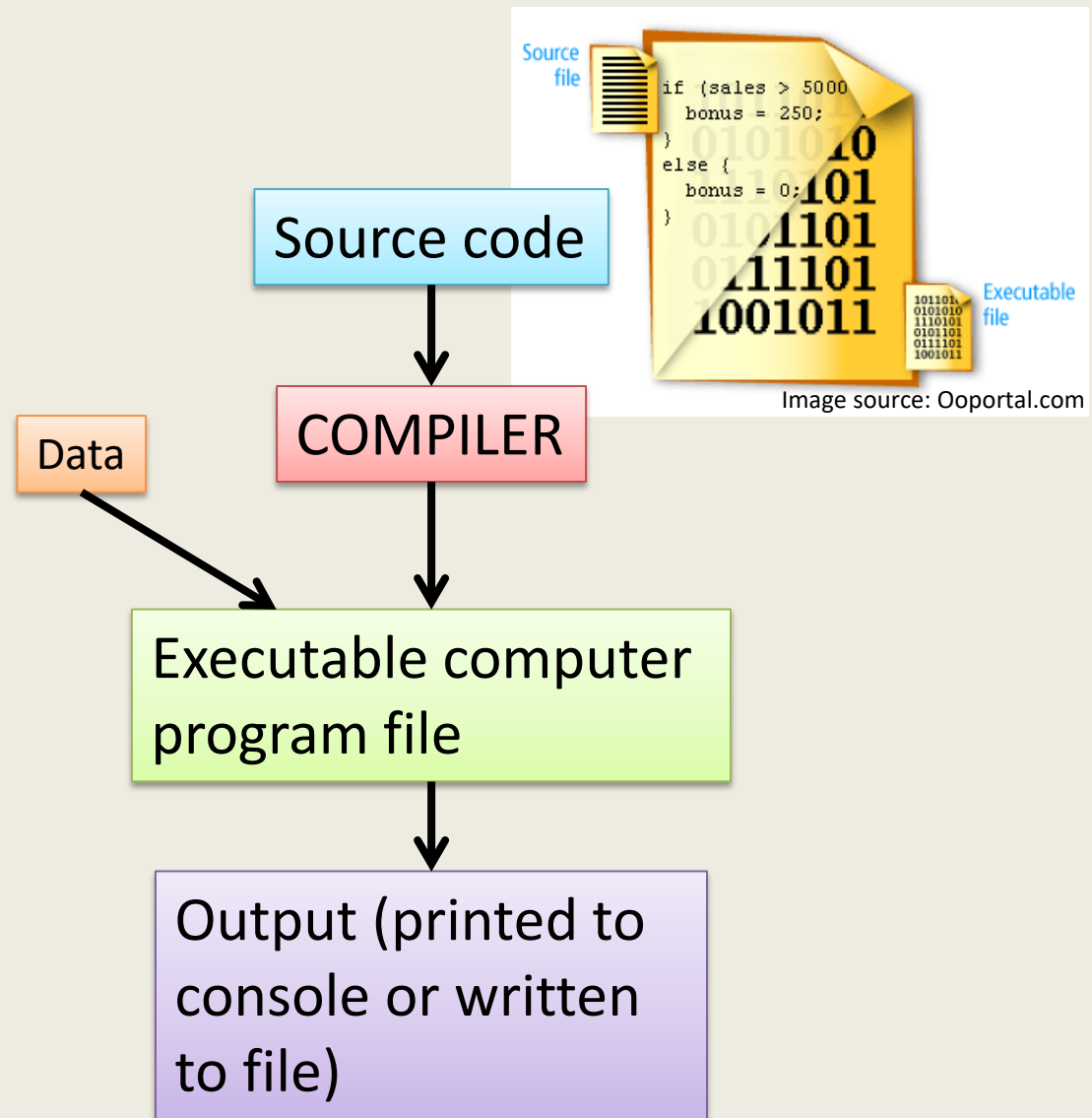
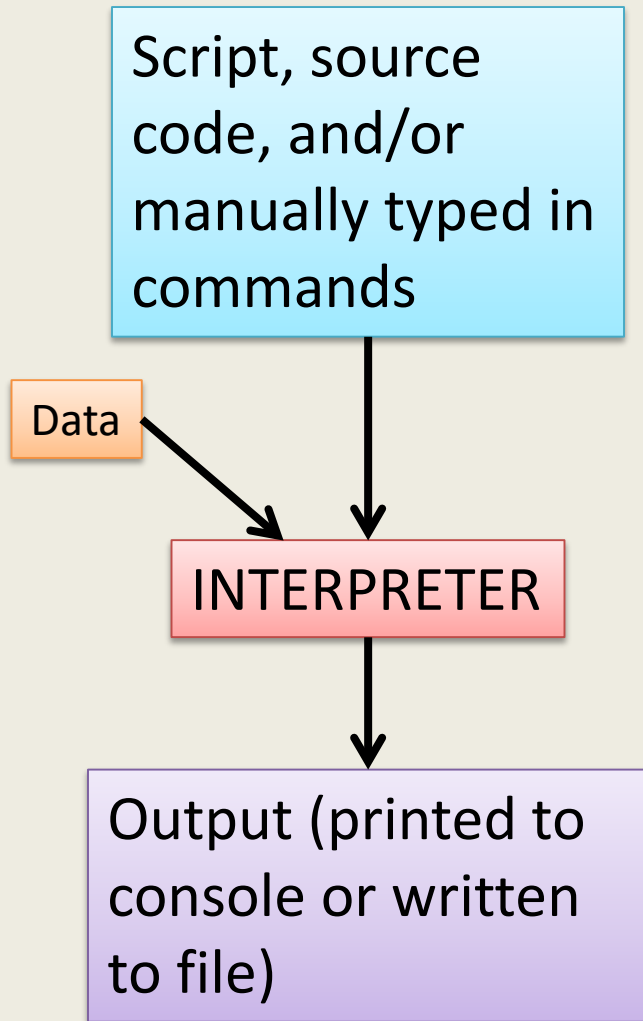
- The program that is running in the lower left corner of RStudio (“console”)
- An **interpreter** is a computer program that takes user commands, converts them into machine code (binary), and then executes those commands
- R is an **interpreted language**, or **scripting language**, because it is designed to be processed by an interpreter

What do we call a text file that we send to the interpreter?

- If the text file contains a series of commands to be executed one after another, it is called a **script**.
- If a text file contains code that defines new functions etc. but doesn't produce any output, it is called **source code**.
- (You can define functions within a script as well.)

What is the alternative to an interpreter?

- A **compiler** is a computer program that reads source code, converts it to machine code, then saves that machine code as a binary executable file.
- Languages like C and Java are **compiled languages**
- The R interpreter is built from compiled C code, and many R functions can **call** compiled C code (runs faster)
- Some languages like Python have both interpreter and compilers available.



More on compiling

- We will do compiling later with the Rcpp package.
- Some bioinformatics software is distributed as source code that you have to compile yourself, especially if you are running Linux. (There are so many flavors of Linux that it is impossible to make a binary that will work on all of them.)

```
>tar -xvf treemix-1.0.tar.gz  
>cd treemix-1.0  
>./configure  
>make  
>make install
```



Installation steps for the software TreeMix, which includes the “make” command for compiling C code.

Setting up projects in RStudio

- Go to File → New Project
- For today we will go to New Directory → New Project
- Projects make it easy to group together related scripts and data
- They also help keep your workspace separate for separate tasks
- Good idea to make subfolders like “data”, “figures”, “edited_data”

My_project

My_project.Rproj (project file for RStudio)

My_analysis.R (script)

My_envt.RData (objects
saved from environment)

My_functions.R (source code)

My_data.csv (dataset)

My_graph.pdf (figure)

My_output.csv (results)

.git
.gitattributes
.gitignore
(for tracking changes)



You could give someone this whole folder
and they could reproduce your analysis.

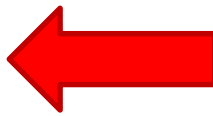
Intro to R syntax

Comments are your friends

- Explanations of what the code is doing
- Explanations of why the code was written a certain way
- Placeholders describing code that you are going to write
- Reminders of things you need to fix
- Code that you don't want to run but also don't want to delete

My favorite historical code comment

```
175  # Page 801
176      CAF      TWO          # WCHPHASE = 2 ----> VERTICAL: P65,P66,P67
177      TS       WCHPHOLD
178      TS       WCHPHASE
179      TC       BANKCALL     # TEMPORARY, I HOPE HOPE HOPE
180      CADR      STOPRATE    # TEMPORARY, I HOPE HOPE HOPE
181      TC       DOWNFLAG     # PERMIT X-AXIS OVERRIDE
182      ADRES     XOVINFLG
183      TC       DOWNFLAG
184      ADRES     REDFLAG
185      TCF       VERTGUID
```



... from Apollo-11 lunar landing guidance calculations

The assignment operator: <–

- Less than sign immediately followed by hyphen (looks like an arrow)
- Equals sign also works (but is bad style)
- Assigns a name (AKA **pointer** or **reference**) to a value
- Any time we change a value in R, this operator is used

Row.names	Var1	Var2	Var3
Obs1	0.4	A	20
Obs2	1.6	A	21
Obs3	0.7	B	30
Obs4	0.9	B	106

Data frames

- This is the type of object we get when we use the `read.table` family of functions (including `read.csv`)
- 2-d table with variables in columns and observations in rows
- Designed for linear regression analysis (although now people do many other things with R)

Indexing a data frame

The diagram illustrates three ways to access data in a data frame:

- `df$Var1` points to the entire **Var1** column.
- `df[[3]]` points to the entire **Var3** column.
- `df[3, 2]` points to the specific value **0.7** at the intersection of **Obs3** and **Var1**.

Row.names	Var1	Var2	Var3
Obs1	0.4	A	20
Obs2	1.6	A	21
Obs3	0.7	B	30
Obs4	0.9	B	106

- You can both extract and assign data using all three methods
- Each column of a data frame is an **atomic vector**

Atomic vectors

- Most basic data type in R
- Every column of a data frame is a vector
- One-dimensional, ordered list of values
- All values must be same type (numbers, character strings, etc.)
- Use the `c` function to make a vector from scratch, or combine vectors.
- `NA` = missing data

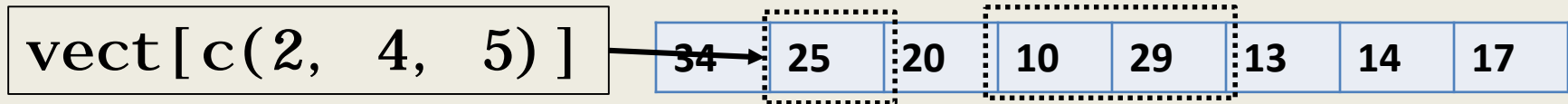
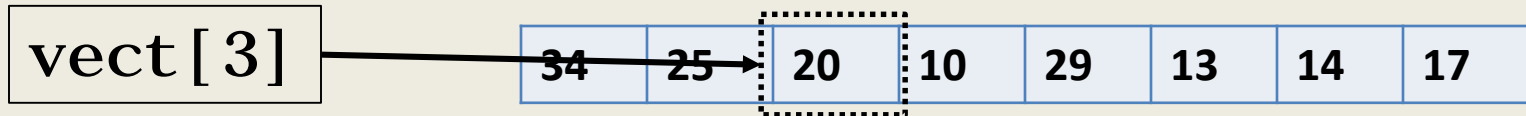
Arithmetic **operators** in R

- All of them work for both scalars (one value) and vectors (multiple values)
 - + addition
 - - subtraction
 - * multiplication
 - / division
 - ^ exponent
 - %% modulo (remainder after division)
 - %/% integer division
- These follow order of operations. Parentheses can be used.

Mini exercise

- Multiply plant height * number of stems * (stem diameter)² to get something that should be roughly proportional to biomass.
- Add the vector as a new column to the data frame.
- Make a scatter plot of this new vector vs. biomass.

Indexing a vector



Indexing a vector

- Use square brackets
- Can be used for retrieving or assigning a value
- We can index vectors by other vectors
- Negative numbers mean all values *except* those indices

Mini exercise

- Extract and print the first five values of your new vector
- Extract and print the first five rows of the data frame

Booleans

- Also known as “logical” values
- These are TRUE and FALSE (must be in all-caps)
- Operators:
 - `&` : AND (true if both are true)
 - `|` : OR (true if one or both are true)
 - `!` : NOT (true if false, false if true)
 - Parentheses can be used for compound expressions
- You can have a vector of Booleans just like you can have a vector of numbers. Operators are vectorized.

TRUE	TRUE	FALSE	&	TRUE	FALSE	TRUE	=	TRUE	FALSE	FALSE
------	------	-------	---	------	-------	------	---	------	-------	-------

Some operators that return Boolean values

- `==` : equals
- `!=` : does not equal
- `>` : greater than
- `<` : less than
- `>=` : greater than or equal to
- `<=` : less than or equal to
- `%in%` : value is found within a vector

Booleans can be used for indexing

- TRUE returns the corresponding value, FALSE does not
- Be careful, since NA returns NA (this is a good reason to use `which`)

20	15	32
----	----	----

 $\left[\begin{array}{|c|c|c|} \hline \text{TRUE} & \text{TRUE} & \text{FALSE} \\ \hline \end{array} \right] = \begin{array}{|c|c|} \hline 20 & 15 \\ \hline \end{array}$

Booleans can be used in math

- TRUE is treated like 1, FALSE like 0
- The `sum` and `mean` functions can be handy to use on a Boolean vector

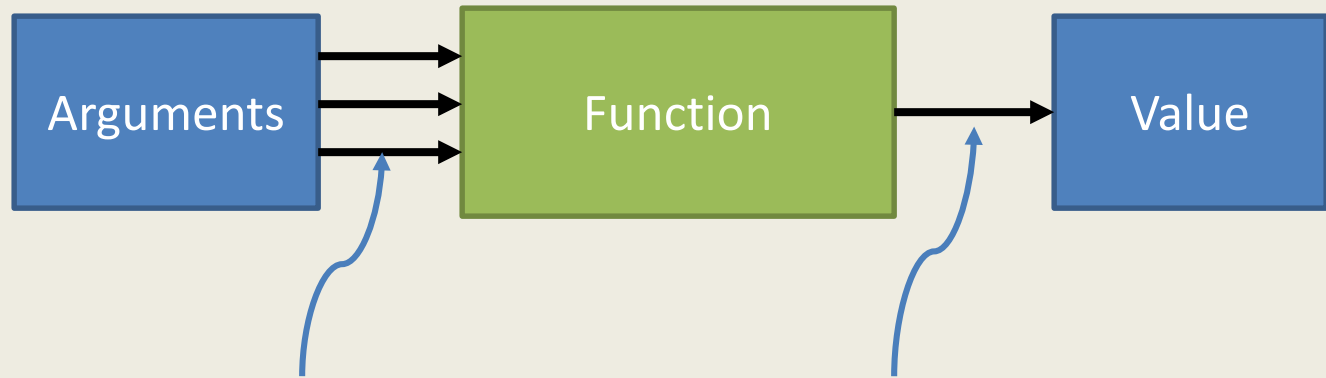
$$\text{sum}(\text{TRUE} \text{ TRUE} \text{ FALSE}) = 2$$

$$\text{mean}(\text{TRUE} \text{ TRUE} \text{ FALSE}) = 0.67$$

Mini-exercise

- Retrieve just the rows of the data frame for plants that have more than 220 stems AND stem diameter of at least 7.
- Use sum to count how many plants had biomass yield over 1000.

Using functions



We **pass** arguments to a function

The function **returns** a value

A function should always return the same value if given the same arguments (unless it is a random number generator)

Using functions

- Type ? and then the function name to get the help page for a function
- The function identifies arguments based on name if indicated, or order if not

Random Samples and Permutations

Description

`sample` takes a sample of the specified size from the elements of `x` using either with or without replacement.

Usage

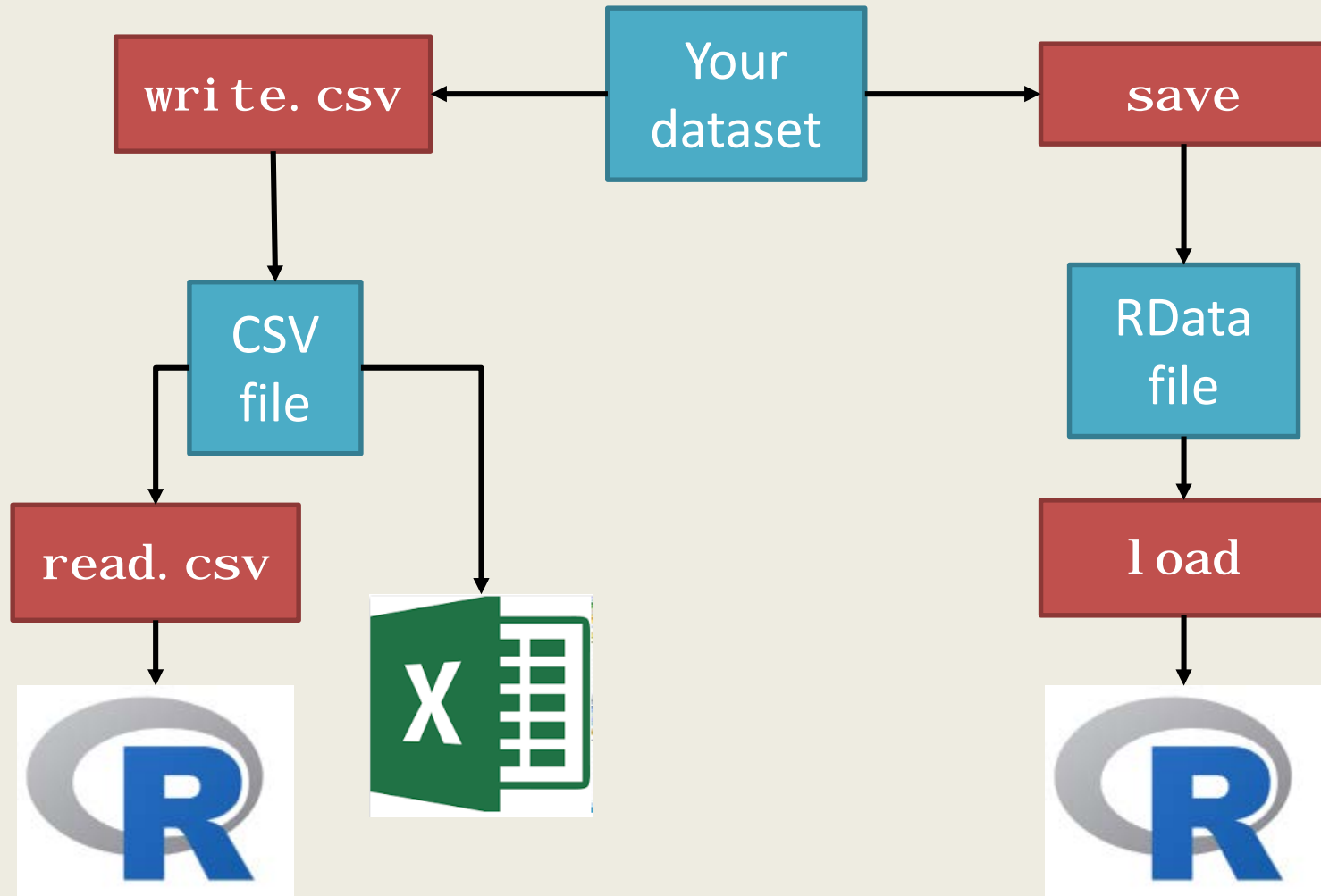
```
sample(x, size, replace = FALSE, prob = NULL)
```

```
sample.int(n, size = n, replace = FALSE, prob = NULL,  
          useHash = (!replace && is.null(prob) && size <= n/2 && n > 1e7))
```

Arguments with default values



Exporting data



Exporting data

- I recommend these options rather than “Save workspace image?”
- Only reload the data that you need
- Don’t accidentally overwrite important datasets when playing around
- Forces you to re-run script next time, make sure it is reproducible

Mini-exercise

- Save your subsetting data frame to a file
- Close RStudio, then reopen it and load your subsetting data frame back in

Last thought for today: priorities when writing code

- There are usually many ways that you can accomplish the same task
- Priority #1: **Make the code easy for a human to understand**
- Priority #2: Make the code run efficiently (only worth it if it is running slowly otherwise)
- Priority #3: Make the code concise (feeds into priorities #1 and #2)

Thursday's lab: manipulating data frames

- Extracting particular rows and columns
- Sorting
- Using grouping factors
- The use of dplyr (from the Tidyverse) to make code for these tasks easy to read