Welcome to Workshop: Introduction to R, Rstudio, and Data

I. Please sign in on the sign in sheet (this is so I can follow up to get feedback).

II. If you haven't already, download R and Rstudio, install to your laptop.

III. Download materials you'll need from my website (http://faculty.washington.edu/jhrl/Teaching.html or google Janneke HilleRisLambers at University of Washington – go to Teaching tab, scroll down (zip file). Or ask me for a USB stick.

Introduction to R, Rstudio, and coding

I. What / Why R?

II. Rstudio & R
   A. The Source, Console, Help and Environment panes
   B. Functions and Data Objects

III. Getting started
   A. Data & Project Management
   B. Good Coding practice

IV. Data wrangling
   A. ChickenScript.R; A demo of how to read in and examine data, merge and subset, define variables.
   B. Nutnet data: explore in pairs

V. Further topics & Resources

But first, brief introductions…

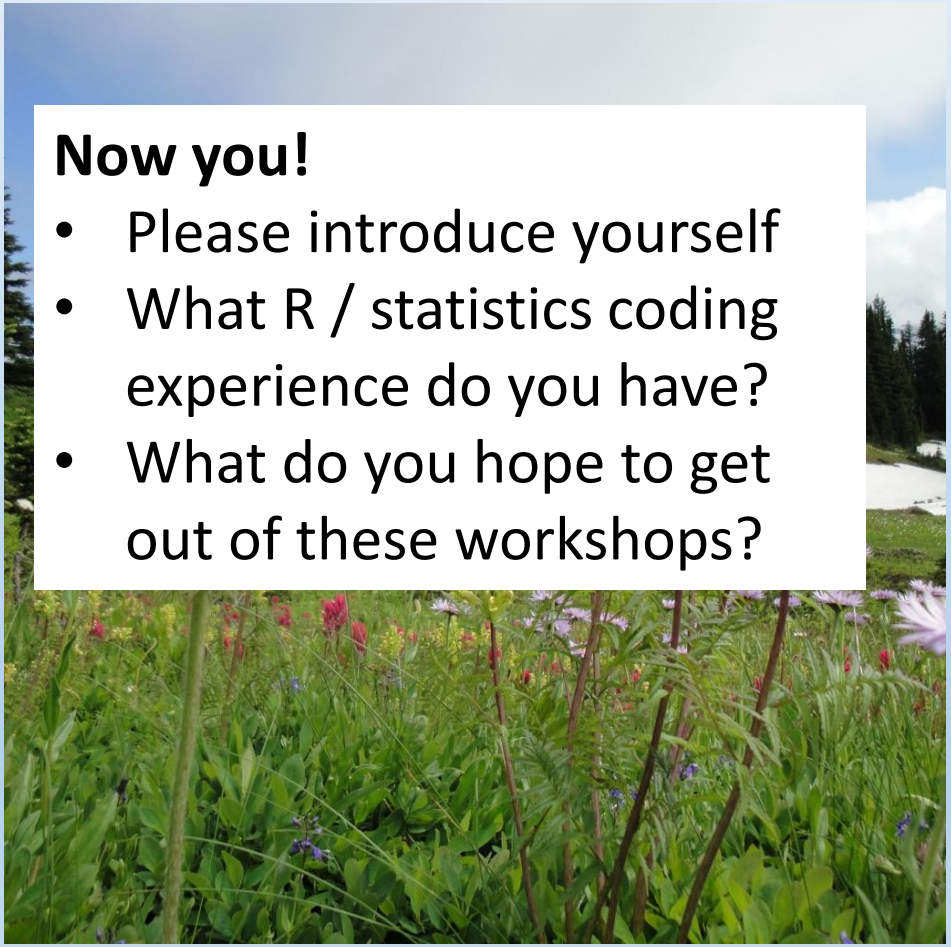
Walker Endowed Professor of Natural History
University of Washington, Seattle (USA)

My Goals:

Introduction (no background required)
Not just coding / statistics (e.g. project management, experimental design)
Collaborative: help each other
Feedback (what worked, what didn't)

Research interests:

Plant Community Ecology, Global Change
Statistics / Coding: since graduate school



Now you!

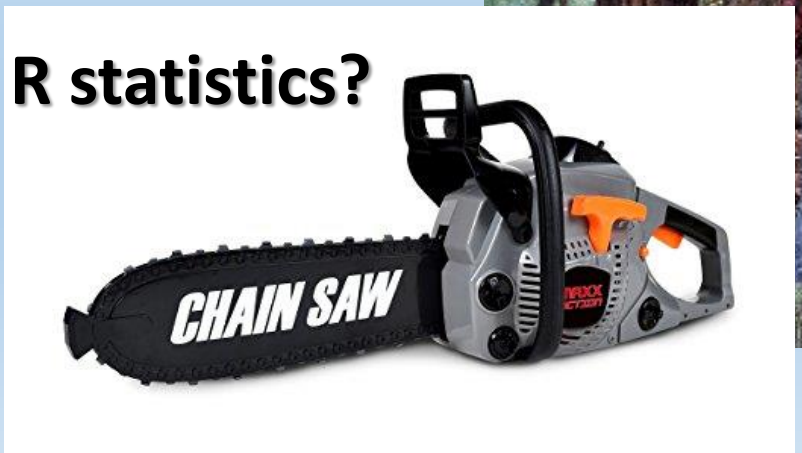
- Please introduce yourself
- What R / statistics coding experience do you have?
- What do you hope to get out of these workshops?

I. What is R?

- Computer language & environment for statistical computing & graphics. Script based (text computer code), not GUI based (menu / point & click).
- Tools for **Data Handling** and manipulation
- Large collection of statistical tools (packages) for **Data Analysis**; contributed by many experts
- Graphical interface for **Visualizing Data** & results from statistical analyses
- Relatively simple and effective, widely used, free, open source…

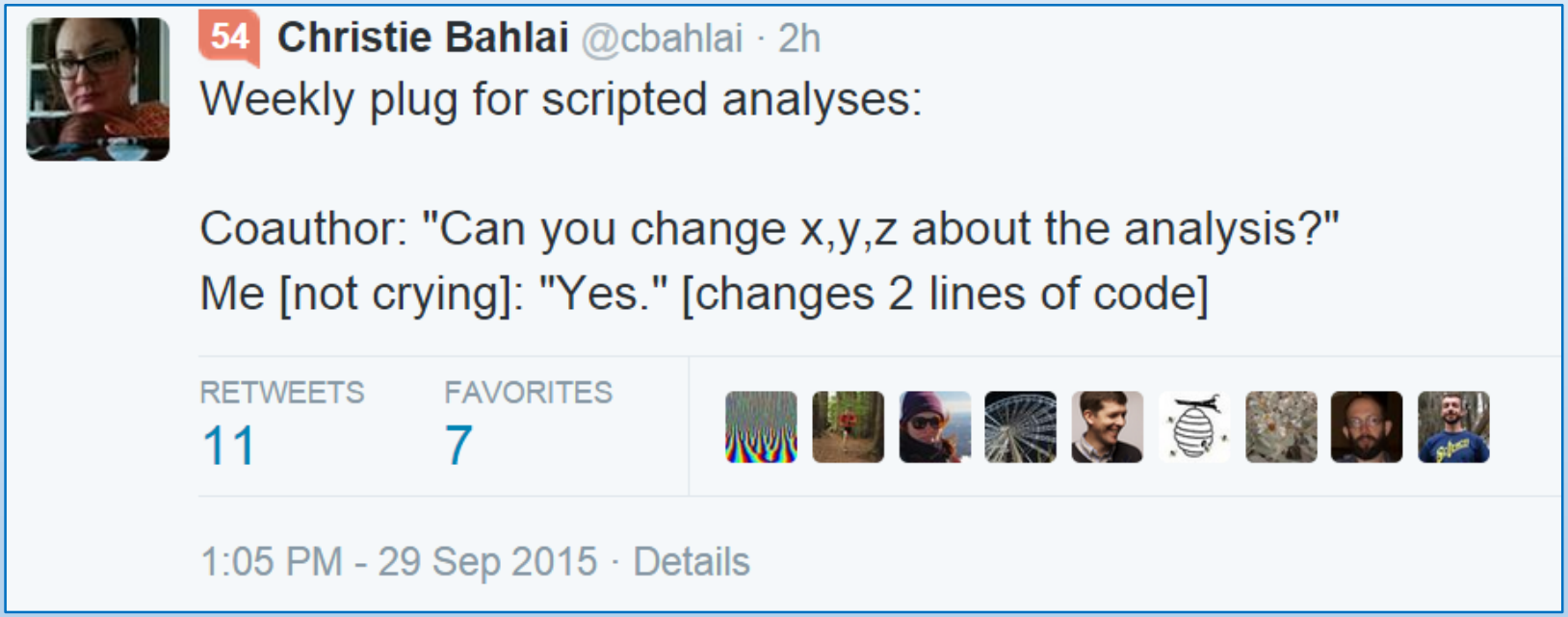
I. Why R?

- The right tool for (many of our) jobs



I. Why R?

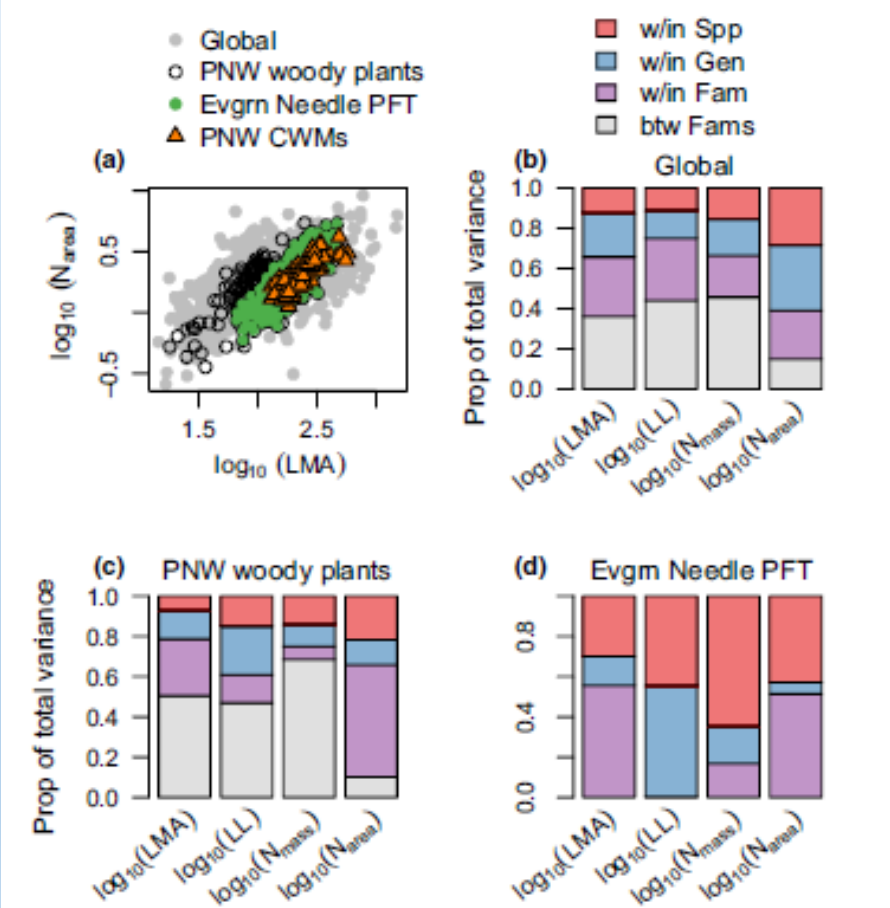
- The right tool for (many of our) jobs
- Reproducible, shareable code & tools for collaboration



Rule of thumb: every analysis you do on a dataset will have to be redone 10–15 times before publication. Plan accordingly.

I. Why R?

- The right tool for (many of our) jobs
- Reproducible, shareable code & tools for collaboration
- Publication quality plots (also easily reproducible)



Anderegg et al. in press

I. Common uses of R

1. Explore data via summaries, plots or classical statistical analyses: ANOVA, LM, GLM, …
2. Advanced analyses: Bayesian inference, random forests, spatial, mixed effects, … (via packages)
3. Publication quality figures
4. Larger projects (e.g. publication): functions, scripts, documentation, reproducibility
5. Build your own R package and share via github.com

I. Learning R…

- R is a programming language, the learning curve can be steep so be patient
- Like human languages, what you get out is what you put in
- Increased productivity when fluent
- Many sources of help: online, books, labmates, etc.

NOTE:

- You learn to program by making mistakes
- Expect to make errors, learn from them, avoid them, but don’t let errors frustrate you

II. Rstudio: what is it?

- Portal through which to use R (IDE).
- Simultaneously write code (in a script), execute code line by line, manage data, get help, view plots

LET'S GET STARTED!

These are instructions

Do / look / find this

> Type this (but not the >)

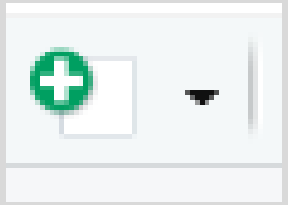
This is something useful / important

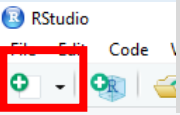
Instruction:

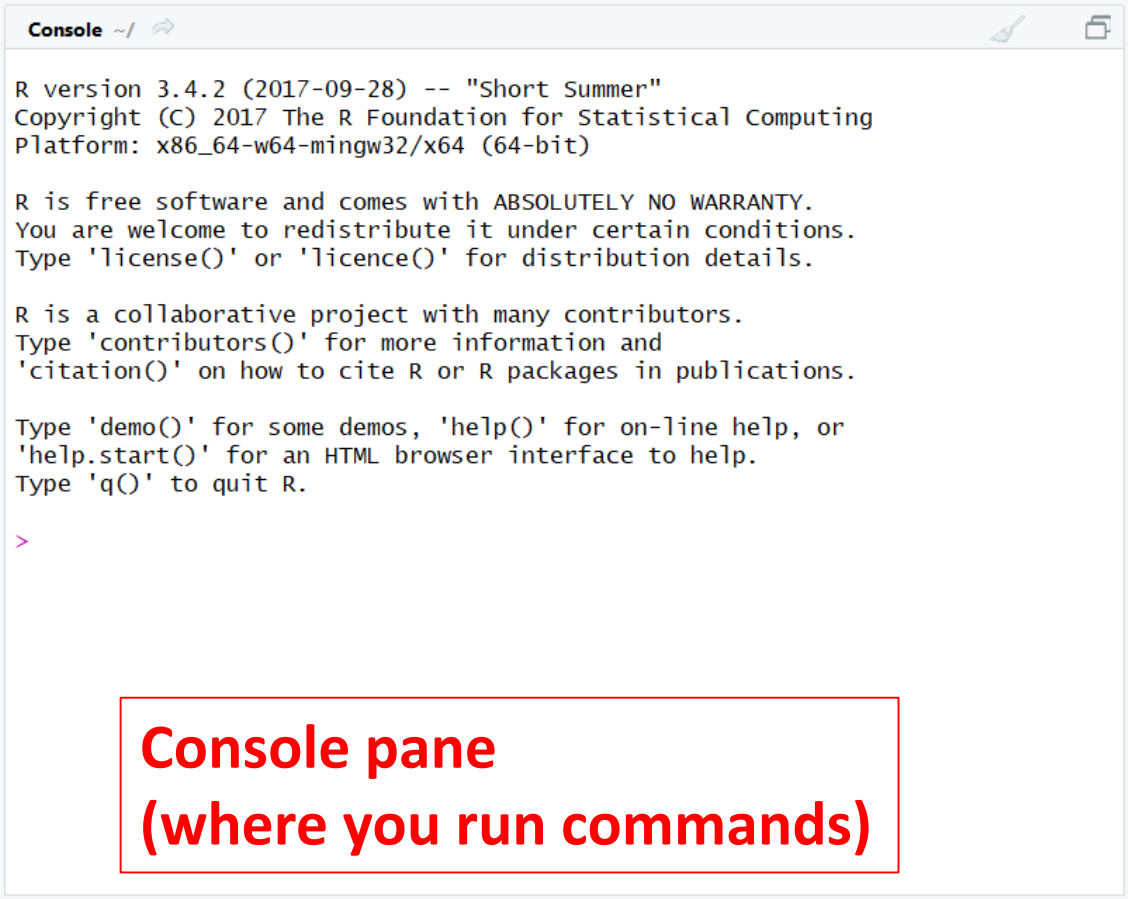
Open Rstudio

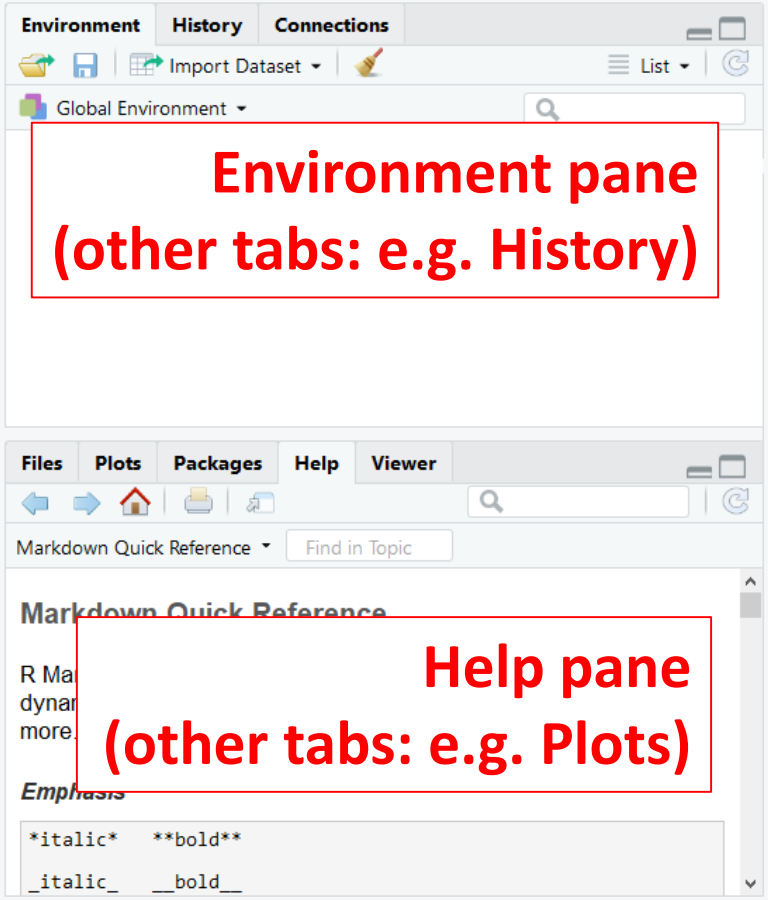
II. Rstudio

Instruction:

- Click on the Green Plus (i.e )
- Then, click on R script

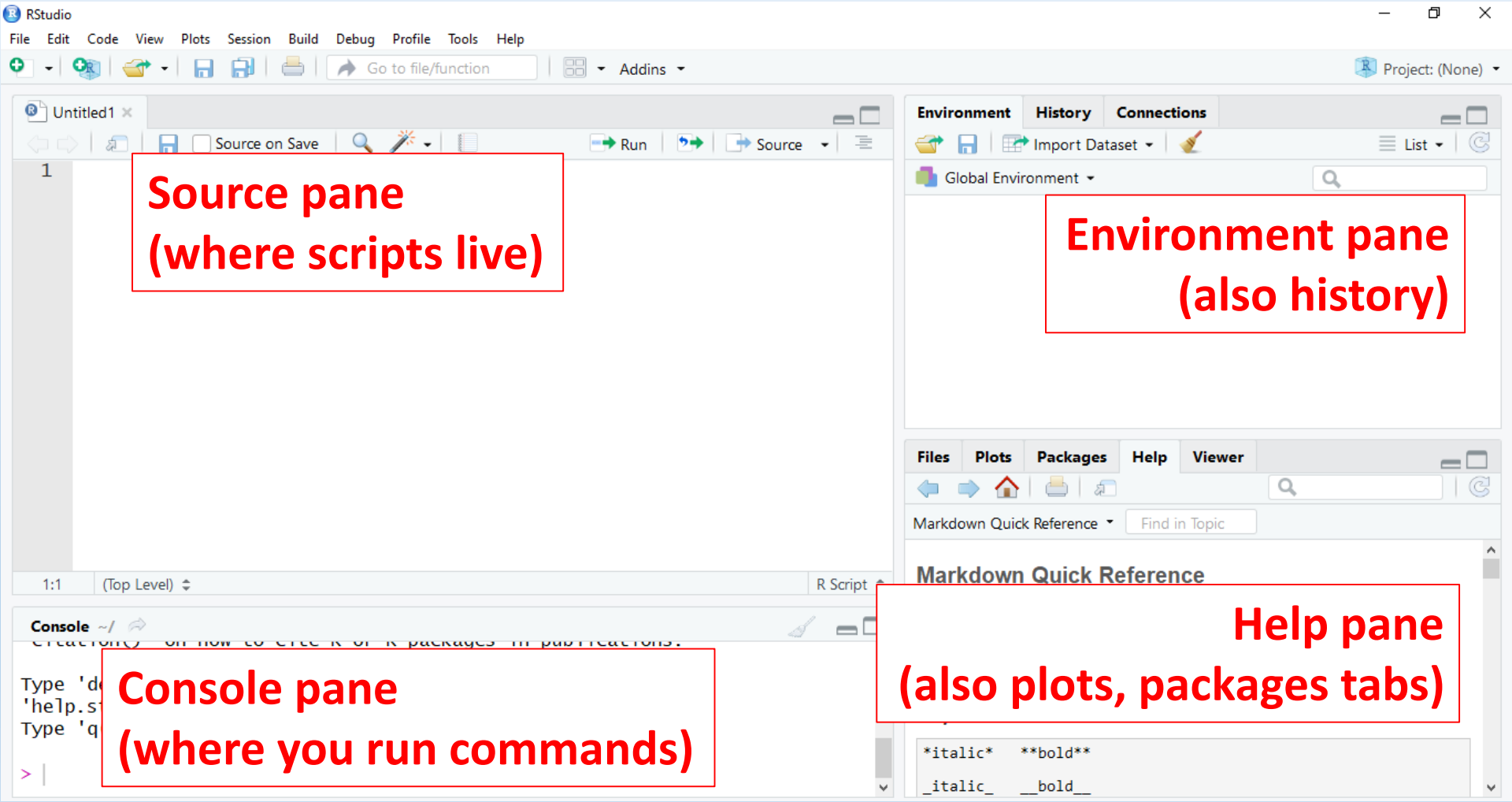






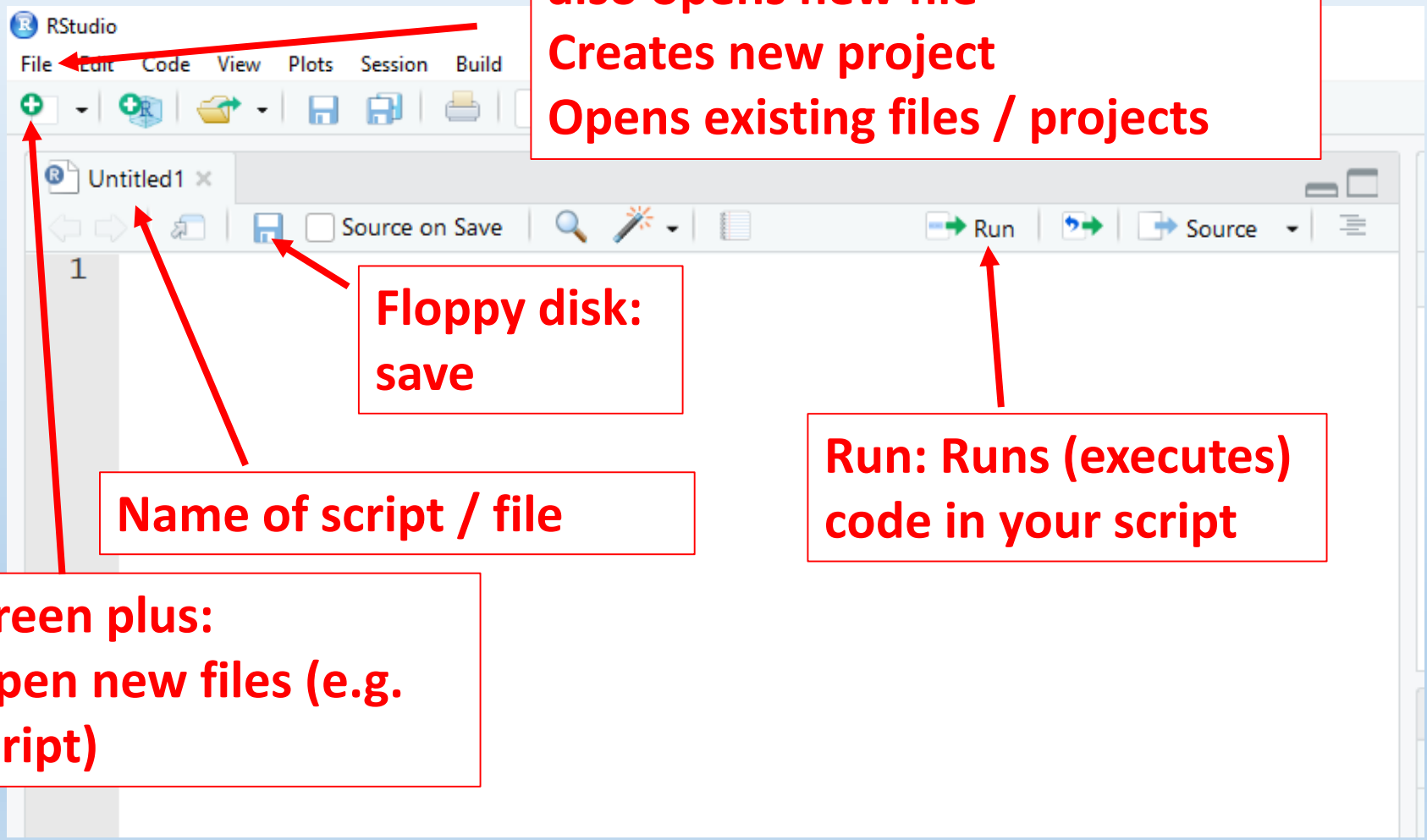
II. Rstudio: the four panes

- This is what you'll usually see

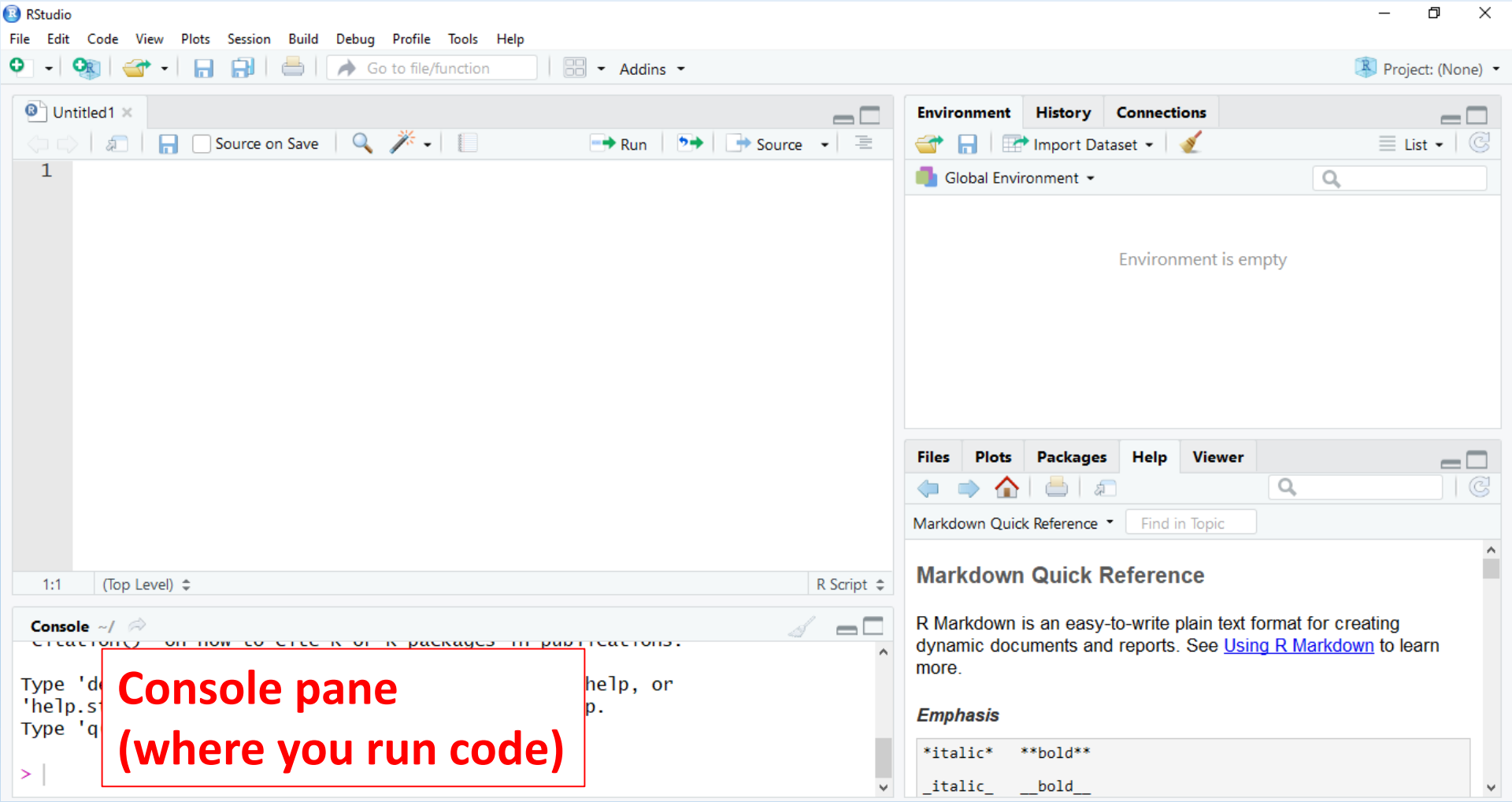


II. Rstudio: the **source** pane

- Scripts (your code)

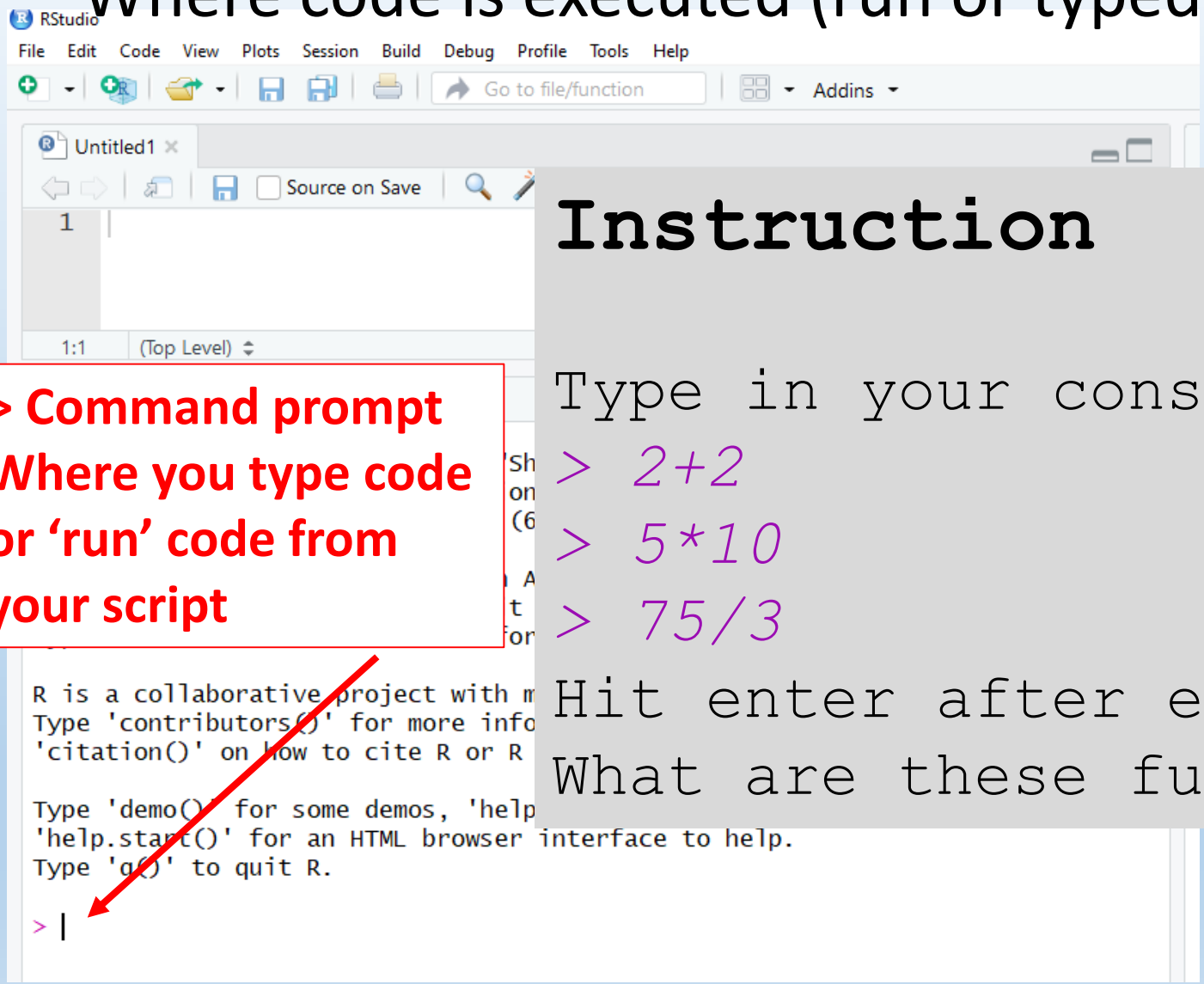


II. Rstudio: the **console** pane



II. Rstudio: the **console** pane

- Where code is executed (run or typed)



> Command prompt
Where you type code or 'run' code from your script

Instruction

Type in your console:

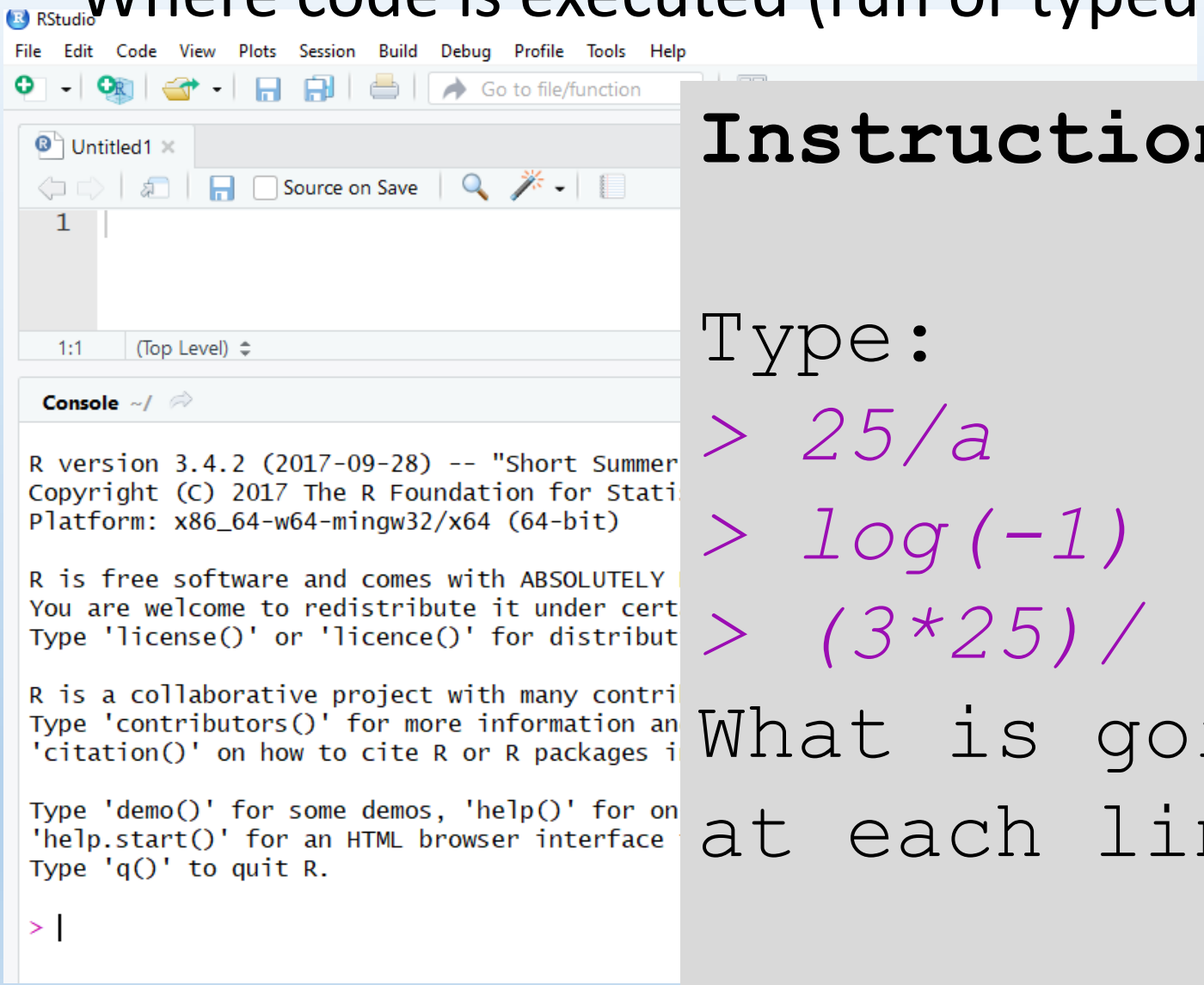
```
> 2+2
> 5*10
> 75/3
```

Hit enter after each line.

What are these functions?

II. Rstudio: the **console** pane

- Where code is executed (run or typed)



Instructions

Type:

```
> 25/a
> log(-1)
> (3*25)/
```

What is going on at each line?

II. Rstudio: the **console** pane

```
Console ~/
>
> 25/a
Error: object 'a' not found
>
>
> log(-1)
[1] NaN
Warning message:
In log(-1) : NaNs produced
>
>
> (3*25)/
+
+
+
```

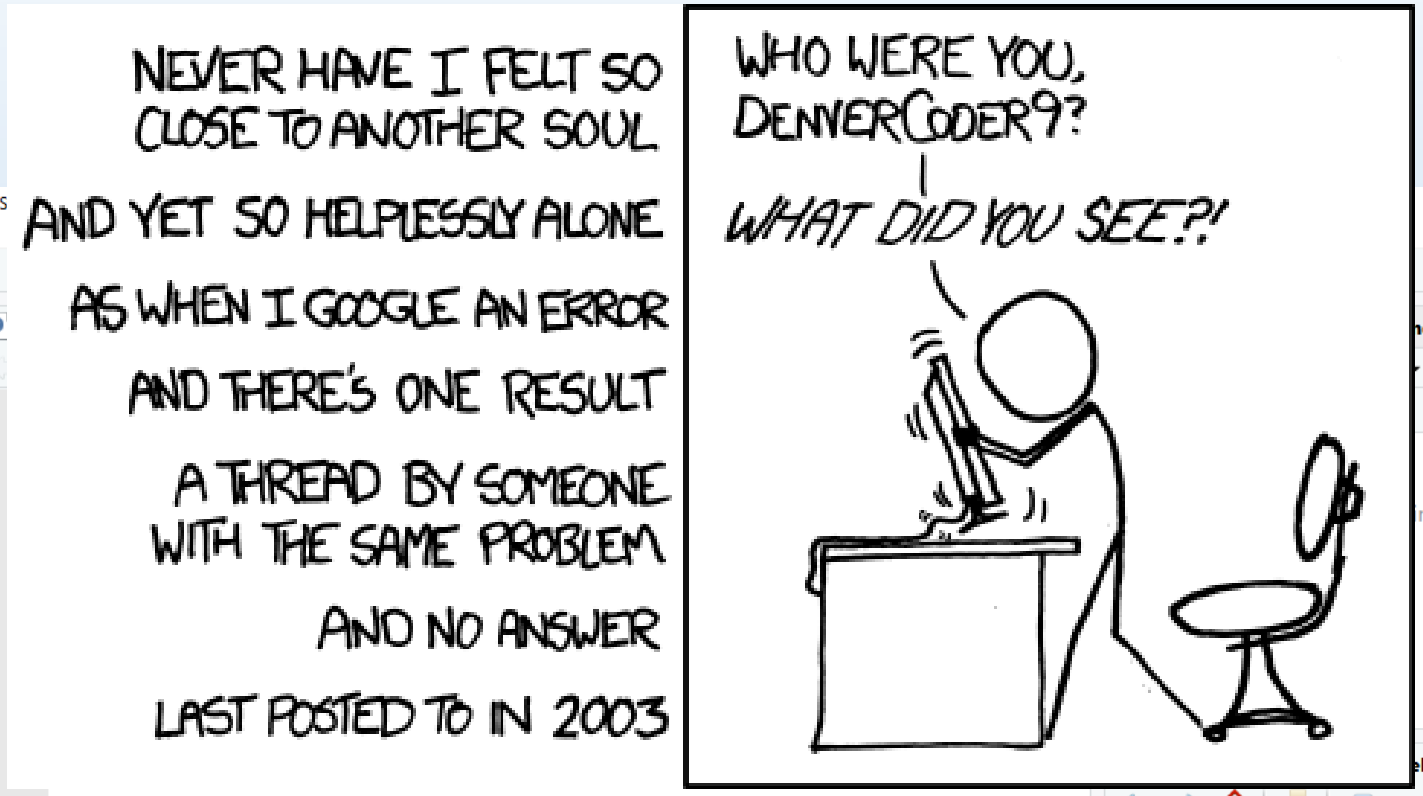
Error message: check and fix

Warning message: check and understand, perhaps fix

+ sign: R is expecting more from you

Hit ‘escape’ to escape! Also if R is generally bogged down…

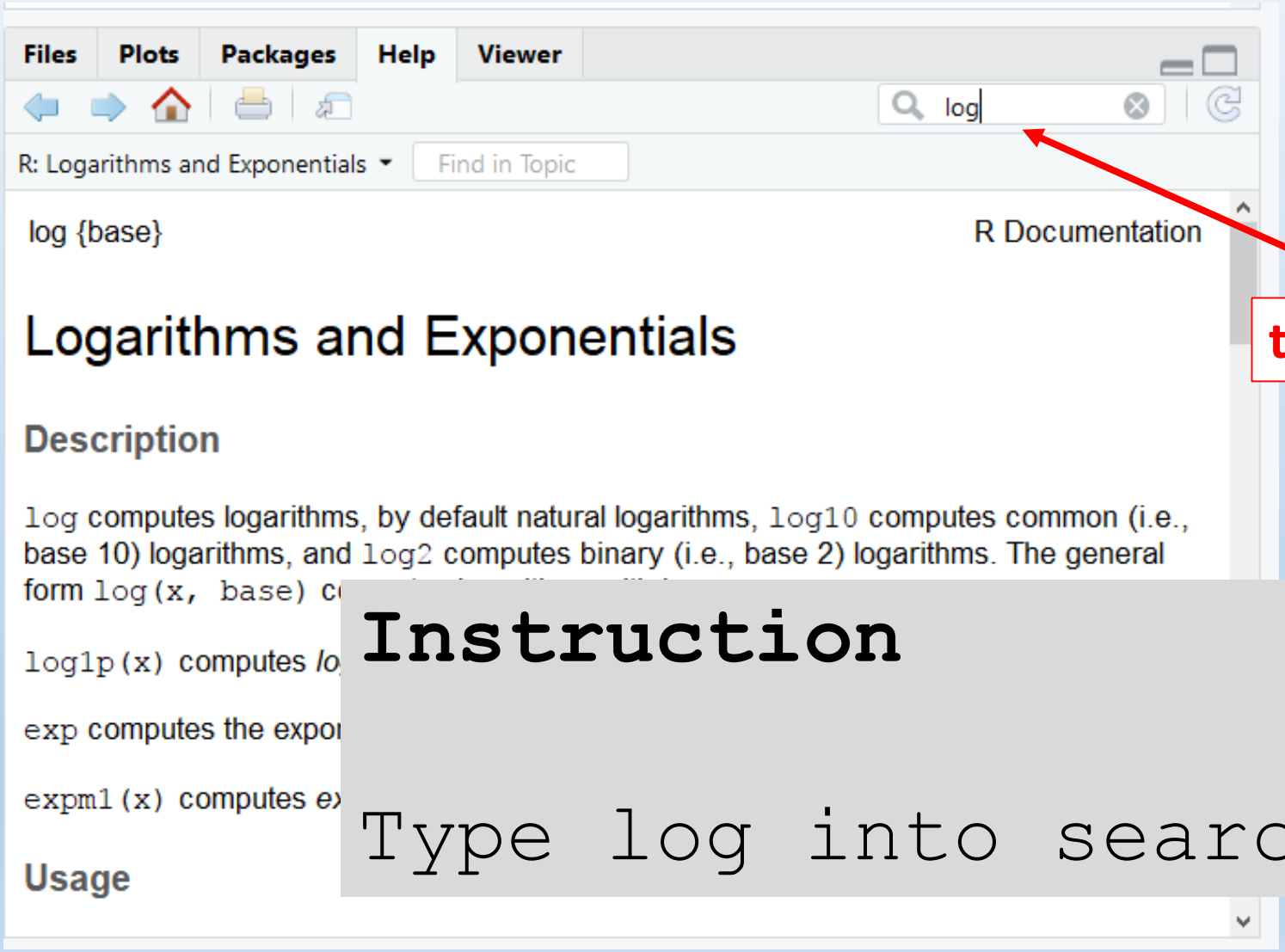
II. Rstudio: …Plots/Packages/**Help**… pane



Help pane
(also plots, packages tabs)

II. Rstudio: …Plots/Packages/**Help**… pane

- Use to get Help. Option 1: type in **Help** pane search bar



II. Rstudio: …Plots/Packages/**Help**… pane

R: Logarithms and Exponentials ▾ Find in Topic

log {base} R Documentation

Logarithms and Exponentials

Description

What the function does in general terms

`log` computes logarithms, by default natural logarithms, `log10` computes common (i.e., base 10) logarithms, and `log2` computes binary (i.e., base 2) logarithms. The general form `log(x, base)` computes logarithms with base `base`.

`log1p(x)` computes *log(1+x)* accurately also for |x| << *1* (and less accurately when *x is approximately -1*).

`exp` computes the exponential function.

`expm1(x)` computes *exp(x) - 1* accurately also for |x| << *1*.

Usage

How to use the function

```
log(x, base = exp(1))
logb(x, base = exp(1))
log10(x)
log2(x)

log1p(x)

exp(x)
expm1(x)
```

Arguments

What does the function need

`x`	a numeric or complex vector.
`base`	a positive or complex number: the base with respect to which logarithms are computed. Defaults to *e*=`exp(1)`.

Details

All except `logb` are generic functions: methods can be defined for them individually or via the `Math` group generic.

`log10` and `log2` are only convenience wrappers, but logs to bases 10 and 2 (whether computed *via* `log` or the wrappers) will be computed more efficiently and accurately where supported by the OS. Methods can be set for them individually (and otherwise methods for `log` will be used).

`logb` is a wrapper for `log` for compatibility with S. If (S3 or S4) methods are set for `log` they will be dispatched. Do not set S4 methods on `logb` itself.

All except `log` are primitive functions.

R: Logarithms and Exponentials ▾ Find in Topic

Value

What does the function return

A vector of the same length as `x` containing the transformed values. `log(0)` gives `-Inf`, and `log(x)` for negative values of `x` is `NaN`. `exp(-Inf)` is `0`.

For complex inputs to the log functions, the value is a complex number with imaginary part in the range *[-pi, pi]*: which end of the range is used might be platform-specific.

S4 methods

`exp`, `expm1`, `log`, `log10`, `log2` and `log1p` are S4 generic and are members of the `Math` group generic.

Note that this means that the S4 generic for `log` has a signature with only one argument, `x`, but that `base` can be passed to methods (but will not be used for method selection). On the other hand, if you only set a method for the `Math` group generic then `base` argument of `log` will be ignored for your class.

Source

`log1p` and `expm1` may be taken from the operating system, but if not available there are based on the Fortran subroutine `dlnrel` by W. Fullerton of Los Alamos Scientific Laboratory (see http://www.netlib.org/slatec/fnlib/dlnrel.f and (for small x) a single Newton step for the solution of `log1p(y) = x` respectively.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole. (for `log`, `log10` and `exp`.)

Chambers, J. M. (1998) *Programming with Data. A Guide to the S Language*. Springer. (for `logb`.)

See Also

Discover other related functions

`Trig`, `sqrt`, `Arithmetic`.

Examples

Sample code showing how it works

```
log(exp(3))
log10(1e7) # = 7

x <- 10^-(1+2*1:9)
cbind(x, log(1+x), log1p(x), exp(x)-1, expm1(x))
```

II. Rstudio: …Plots/Packages/**Help**… pane

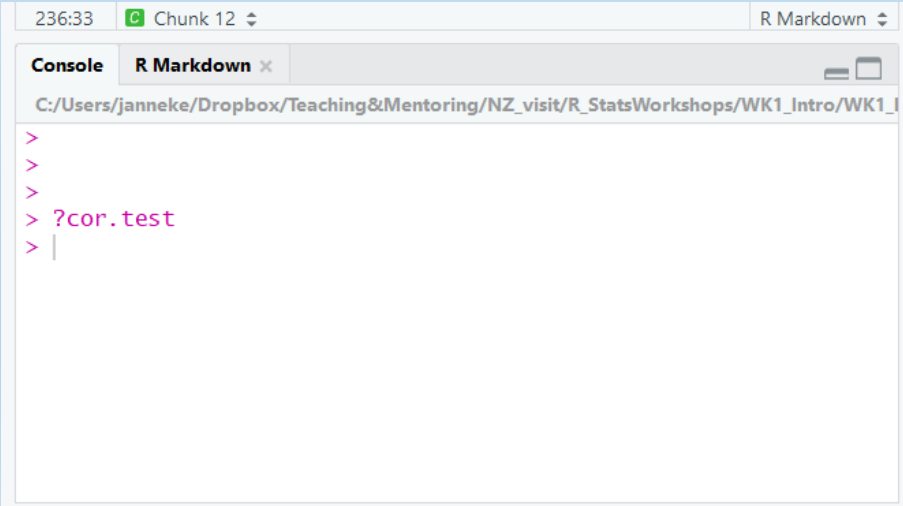
Option 2: type in **Console**, examine in **Help** pane

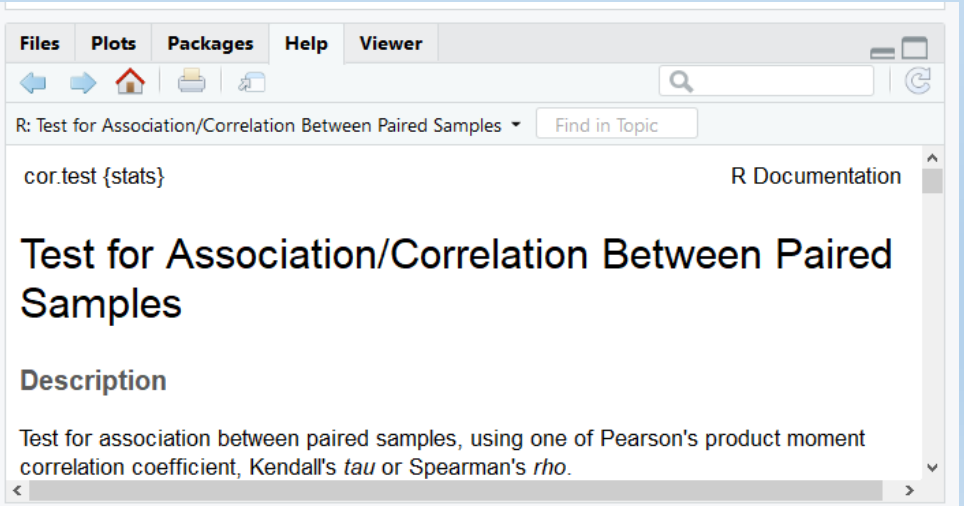
Instruction

Type in console

> ?cor.test

Examine output in Help pane





II. Rstudio: …Plots/Packages/**Help**… pane

Option 3: type in **Console**, examine in **Help** pane

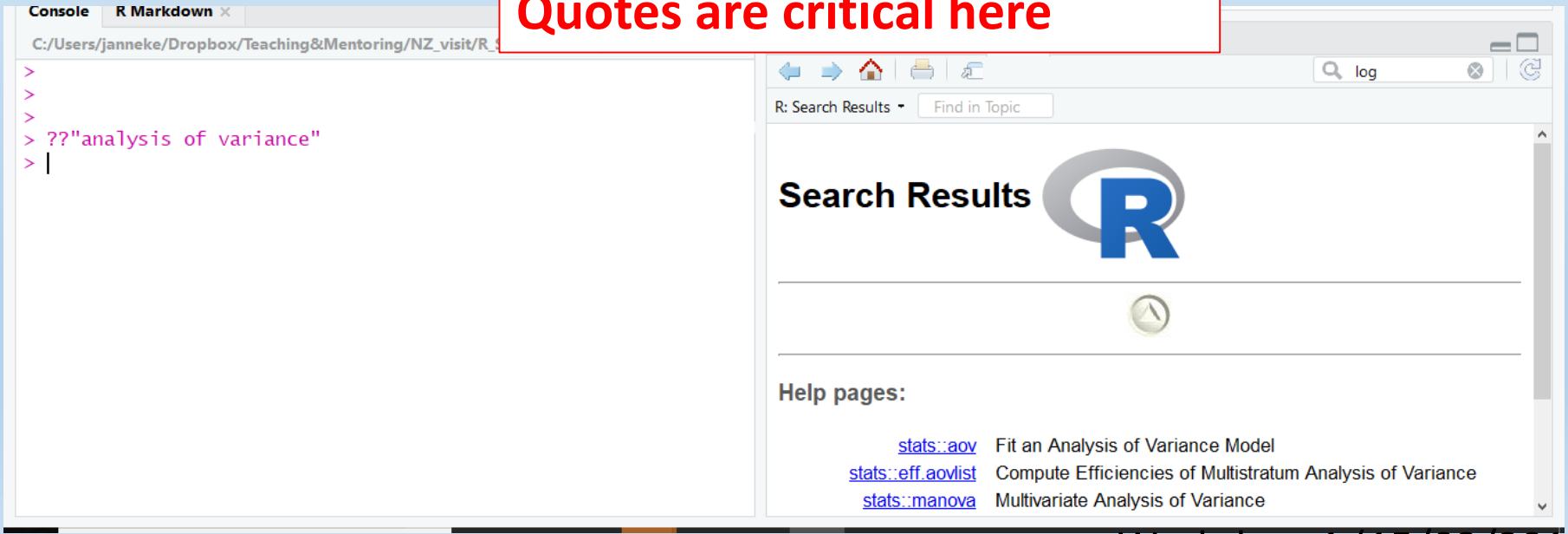
Instruction

Type in console

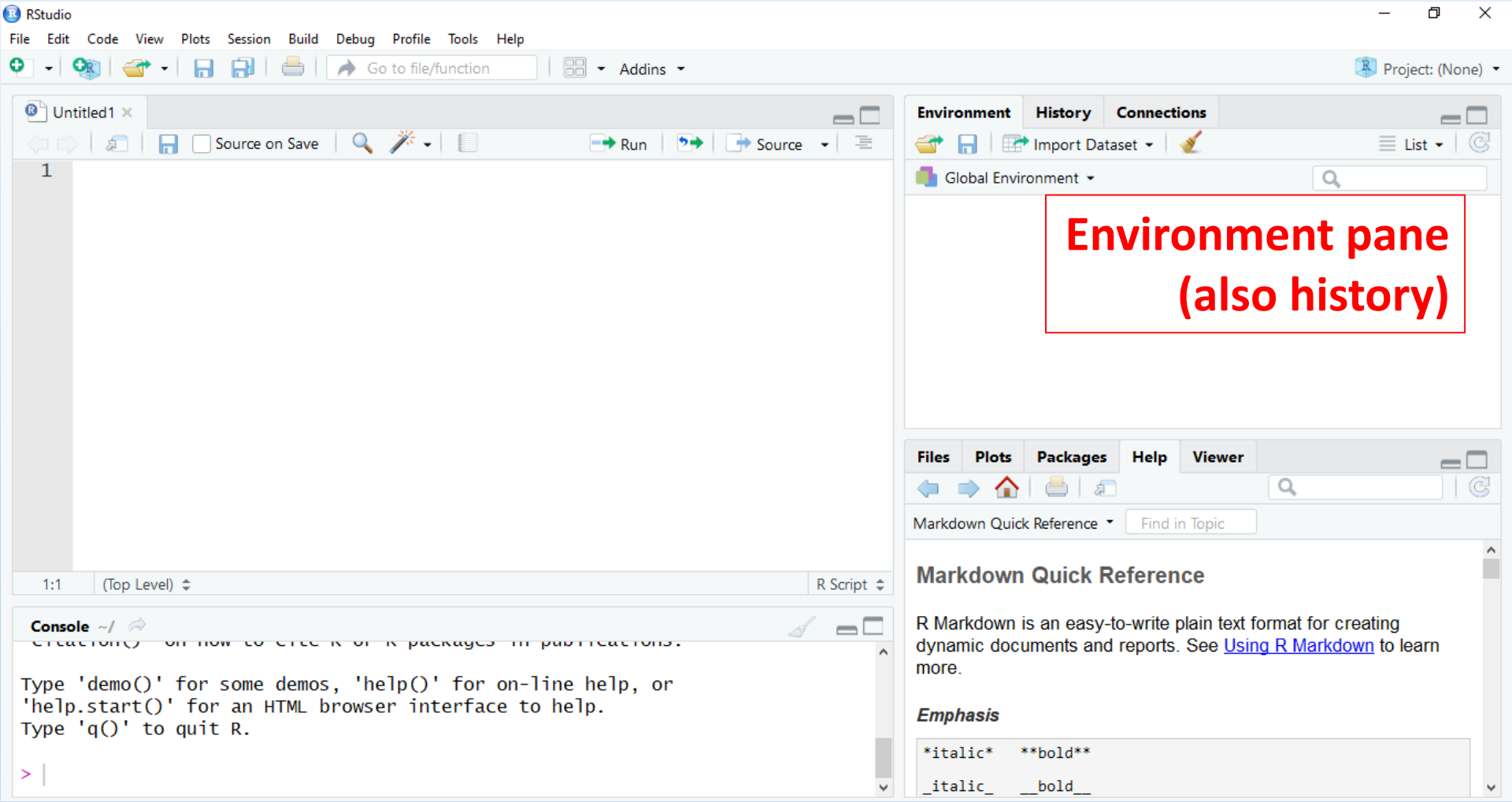
> ??"analysis of variance"

Examine in Help pane

Quotes are critical here

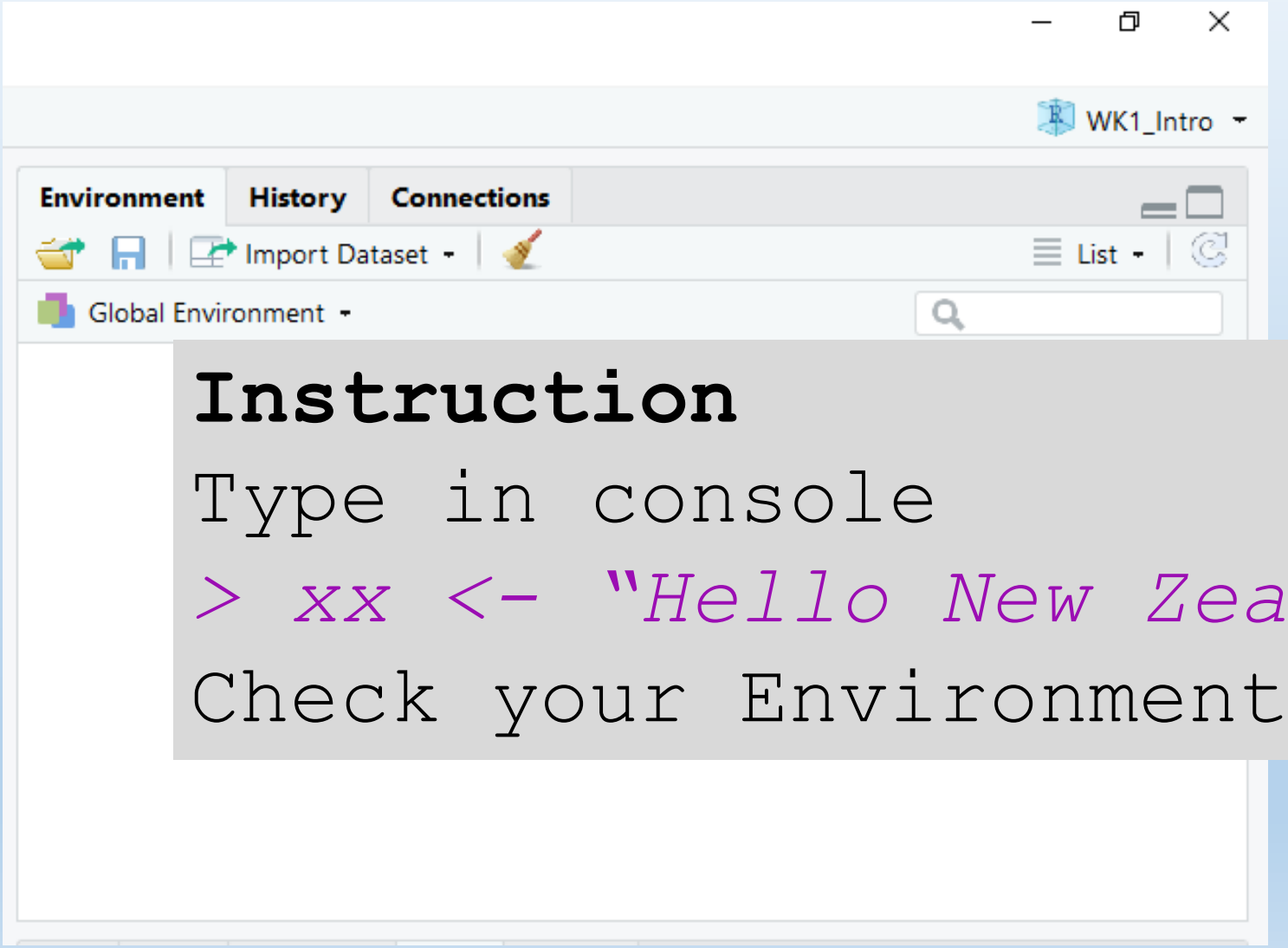


II. Rstudio: The **Environment** pane



II. Rstudio: The **Environment** pane

- All objects you've identified, created show up here



Instruction

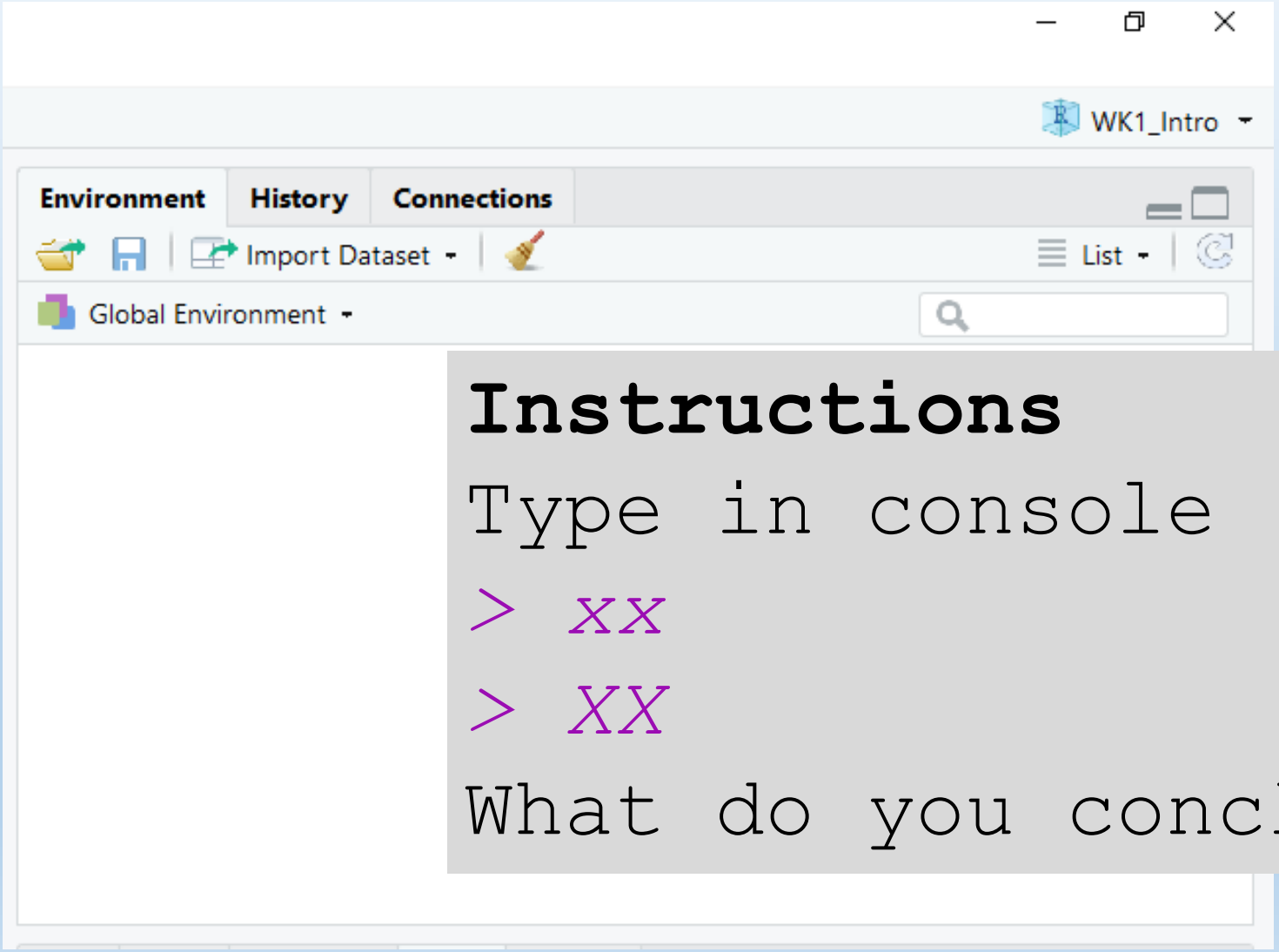
Type in console

```
> xx <- "Hello New Zealand"
```

Check your Environment pane

II. Rstudio: The **Environment** pane

- All objects you've identified, created show up here



Instructions

Type in console

```
> xx
> XX
```

What do you conclude?

II. R & Rstudio: functions and data objects

- A **function** is a chunk of code that conducts a specific task. R comes with many built in functions (e.g. cor.test, log, print, etc).

Instructions

Use help to look up these functions:

```
<-
read.csv
write.csv
plot
print
```

- Eventually, you may want to write your own **functions** (for tasks you repeat often)

II. R & Rstudio: functions and data objects

- Data comes in many types (i.e. **modes**) recognized by R and Rstudio. You'll use the first three most.

Data Type	Also know as:	Example
Numeric	float	42, 3.14, -19.2
Character	string or text	"a", "block1","red", "Pinus palustris"
Logical	boolean	TRUE, FALSE
Integer		
Complex		
Raw		

II. R & Rstudio: functions and data objects

- A **vector** is a collection of items, that are all of the same type (the latter is important). We can create vectors and assign them to names using c():

Instructions

Write in your console:

```
> example_vector <- c("beer","cheese")
> mode(example_vector)
> example_vector
```

- Notice example_vector has now been added to your Environment.
- What does the c in c() refer to?

II. R & Rstudio: functions and data objects

- You can access the different elements within **vectors** with square brackets and numbers.

Instructions

Write in your console:

```
> example_vector[1]
> example_vector[2]
```

- You can add to and replace elements in **vectors** using the <- assignment operator:

Write in your console:

```
> example_vector[1] <- "wine"
> example_vector[3] <- "crackers"
> example_vector
```

II. R & Rstudio: functions and data objects

- You can also create **vectors** with functions called rep and seq (stands for repeat and sequence).

Instructions

Write in your console, and examine:

```
> rvect <- rep(1, times=10)
> svect1 <- seq(1,10)
> svect2 <- seq(1,20,by=2)
```

Instructions

Create the following using seq and rep (and c() as little as possible):

- Odd integers between 1 and 99
- The numbers 1,1,1,2,2,2,3,3,3

II. R & Rstudio: functions and data objects

- **Matrices** are 2-dimensional collections of data. Like vectors, they only have 1 data type in them.

Instructions 1

Write in your console:

```
> example_matrix <- matrix(svect1,nrow=5,ncol=2)
> example_matrix
```

- Extract rows and columns (vectors), and cells from **matrices** using square brackets [row,col]:

Instructions

Extract from example_matrix (in your console)the 2nd column, the 5th row, and cell that is in the 3rd row and 1st column in 3 commands

II. R & Rstudio: functions and data objects

- **Data frames** are 2-dimensional, like matrices, but can hold several types of data. Try the following:

Instructions

Write in your console:

```
> spp <- c("Anemone", "Avalanche Lily",
"Subarctic Lupine", "Mountain Daisy")
> flowercol <- c("white","white","purple","pink")
> visits <- c(3, 5, 25, 7)
> PlantDat <- data.frame(spp,flowercol,visits)
```

- As with matrices, rows / cols can be extracted with brackets. Columns can also be extracted by name:

Write in your console:

```
> PlantDat$visits
```

II. R & Rstudio: functions and data objects

- When you read a .txt or .csv file into R, it will be read in as a **data frame**, from which you can extract columns of data (vectors) as response and explanatory variables. These can also be transformed (e.g. if visits were monitored over 15 minutes and we want a per hour estimate):

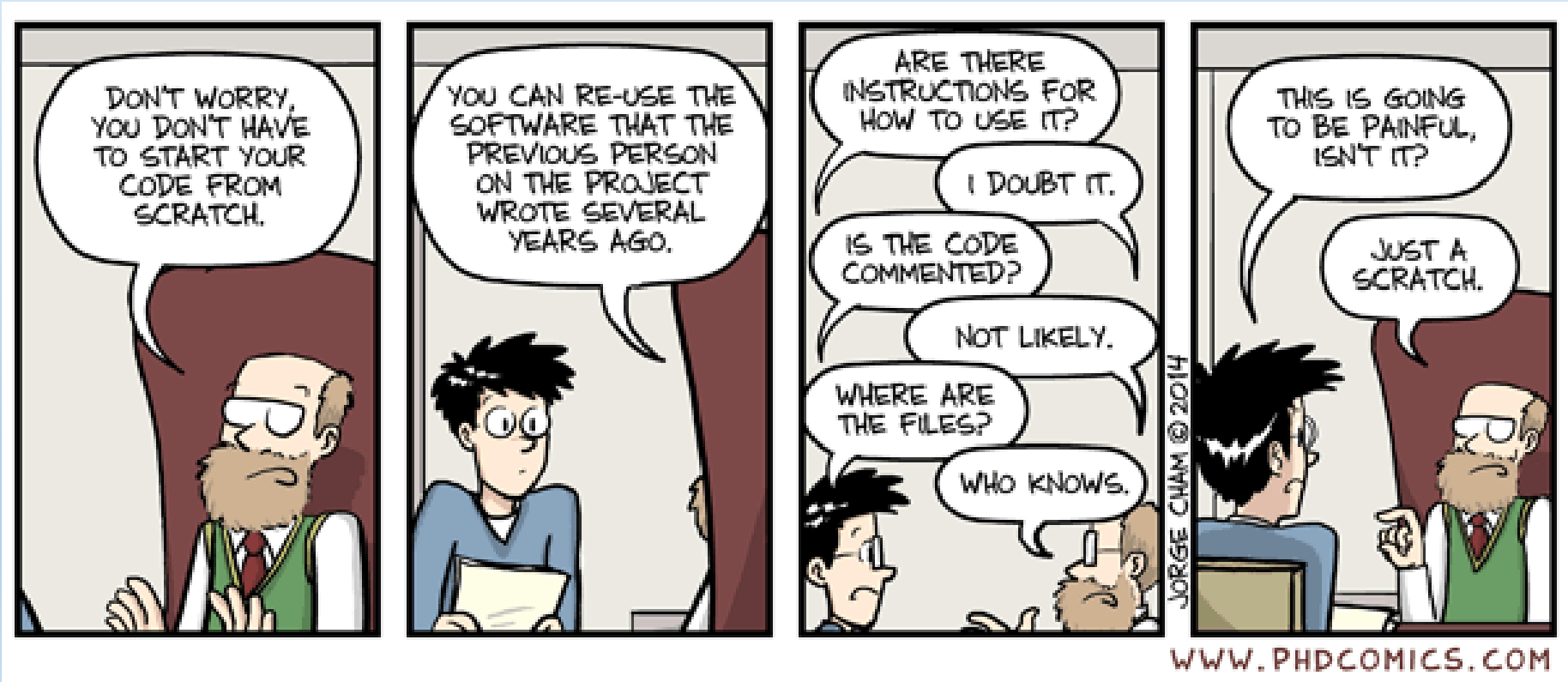
Instructions

Create a vector called visitsperhour from PlantDat.

```
> visitsperhour <- PlantDat$visits * 4
```

III. Organization = reproducibility (& clarity)

- Organize your files (R Projects)
- Organize your code (comments, structure)
- Presumption: you've organized your data (we'll talk about that if we have time)…



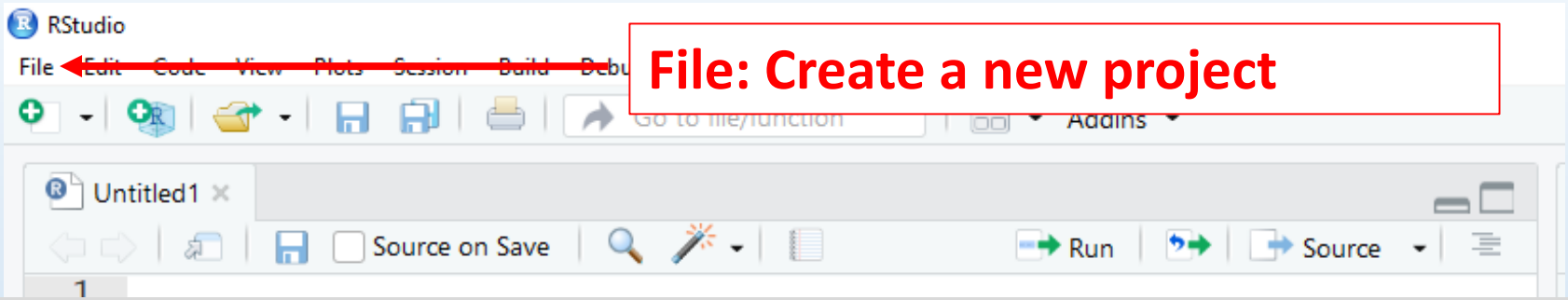
III. Getting Started: Project Management

SUGGESTIONS

- Create a directory for each project, naming it something informative (to you).
- Keep your data files, R script(s) and output files (e.g. figures, simulation results, summary tables) together in that directory (potentially w/ subfolders called data, results, etc).
- Treat data as read only, output as disposable (i.e. R generates everything from your script and data files)

Potential Tool to do this: create an R project

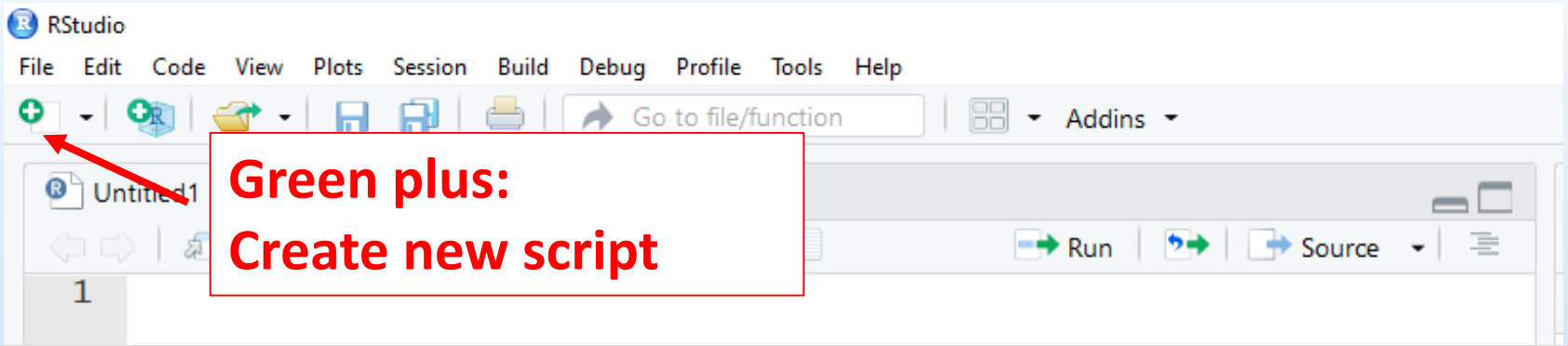
III. Getting Started: Creating an R Project



Instructions

- Go to File / New Project
- Choose New Directory / New Project
- Choose a directory / folder name (e.g. Workshop1) to write in top box
- Choose a location for this directory
- Copy all files for this workshop there

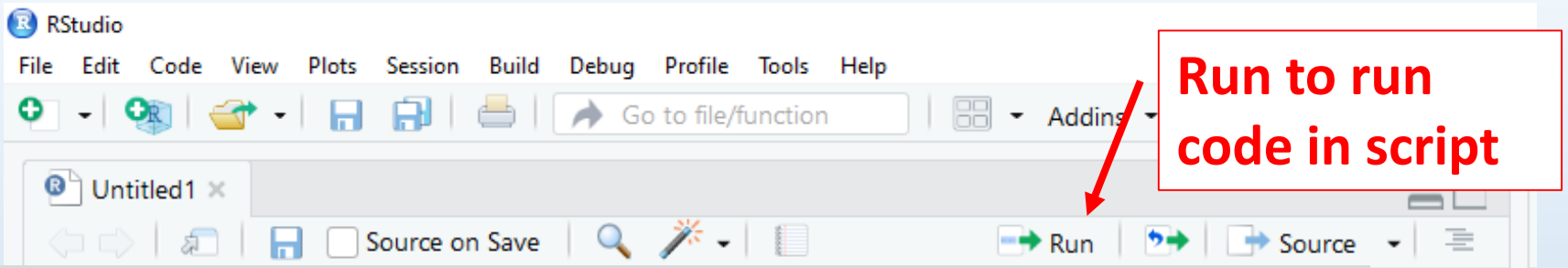
III. Getting Started: Starting a new script



Instructions

- Create a new script
- Save it as SimpleScript.R
- Notice that SimpleScript.R has shown up in the R project directory you created…
- NOTE - You can close your R project (under File / Close Project) when done, and reopen by clicking on the R project file (ends in .Rproj)

III. Getting Started: writing/running a script



Instructions

1. Type into SimpleScript.R:

```
2+2
5*10
75/25
4^3
5<6
5>6
```

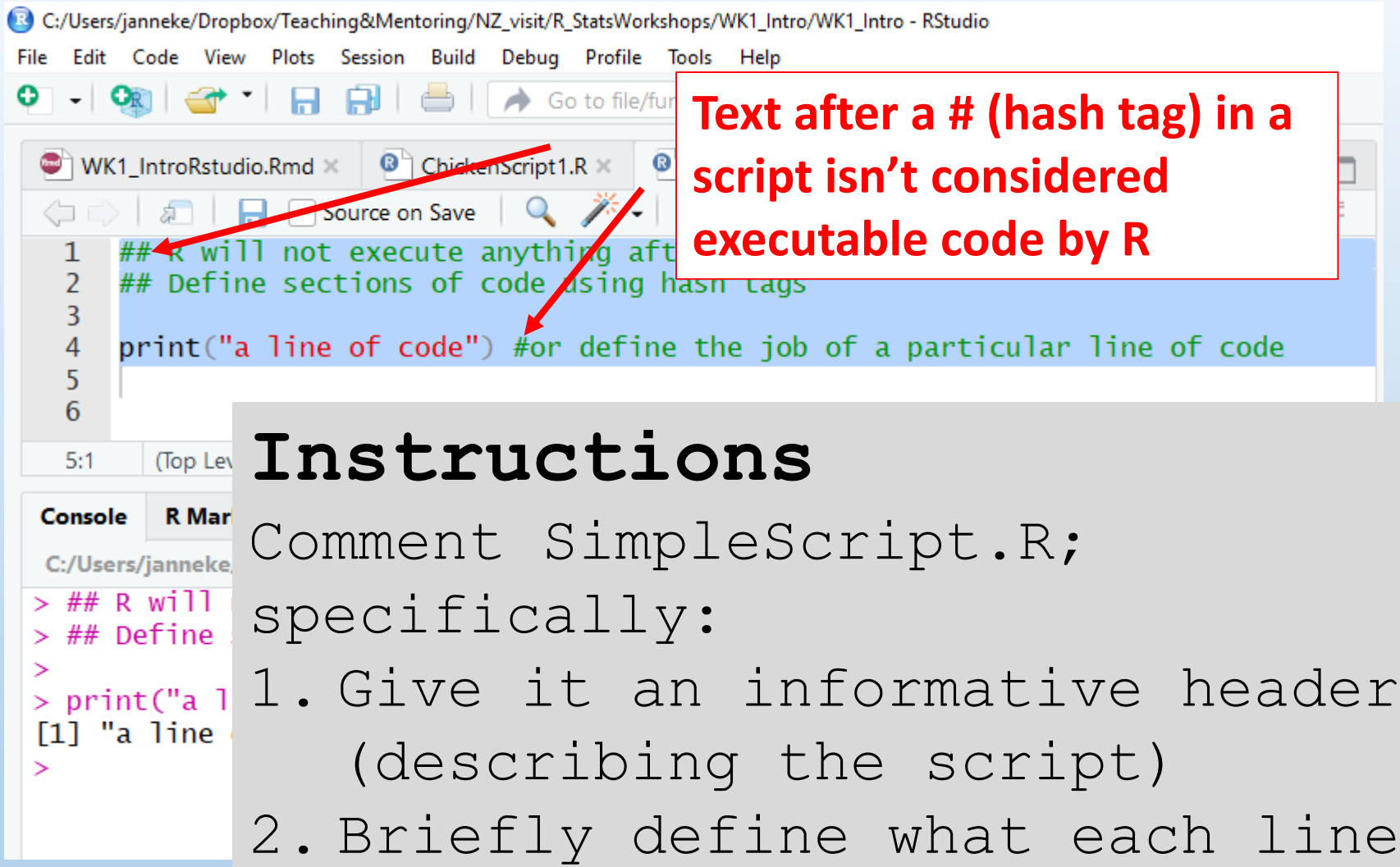
2. Save
3. Highlight the text and hit run

III. Getting Started: what is ‘clean’ coding?

Instructions

1. Open ChickenScript1.R. It contains 57 lines of code.
2. Does it look easy to understand?
3. Imagine that you created this code, and reopened it after a field season (3 months later). Do you think you would understand it right away? How could you have written it to make it easier to use? **Discuss with your neighbor.**

III. Getting Started: comments



Instructions

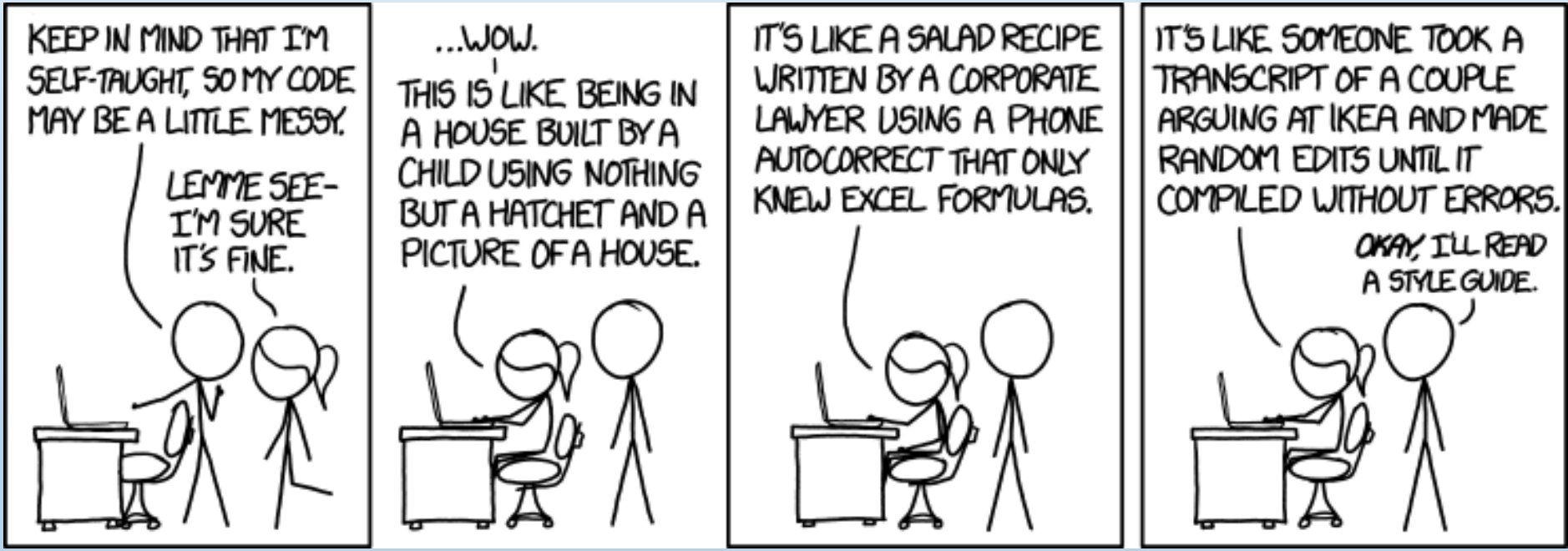
Comment SimpleScript.R; specifically:

1. Give it an informative header (describing the script)
2. Briefly define what each line of code is doing.
3. Run your code.

III. Getting Started: Code formatting

SUGGESTIONS: Develop your own style guide; i.e.

- Create a script template, and use it for each new script (e.g. see HRL_ScriptTemplate.R for an example).



III. Getting Started: Code formatting

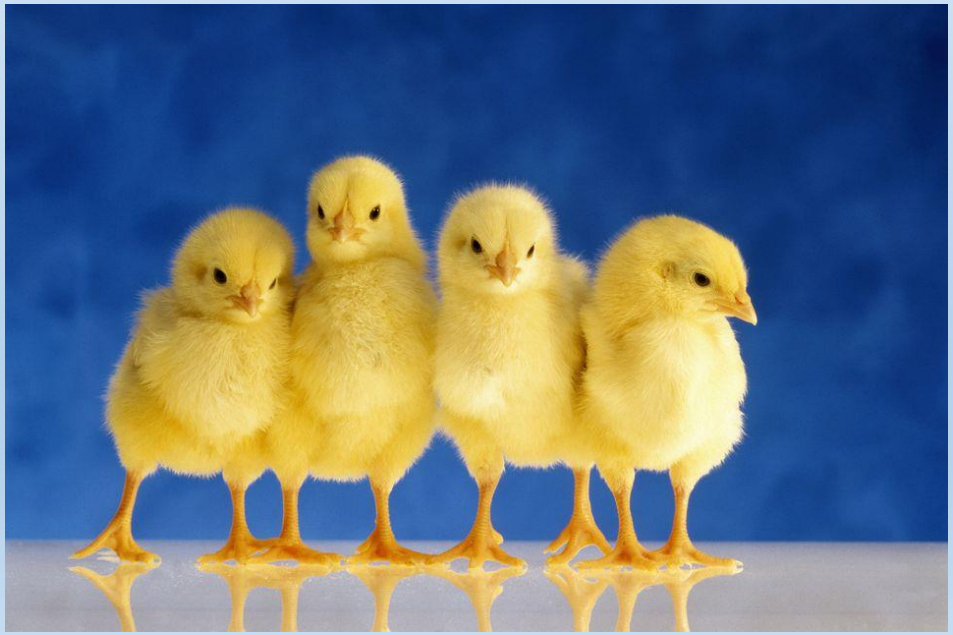
SUGGESTIONS: Develop your own style guide; i.e.

- Create a script template, and use it for each new script (e.g. see HRL_ScriptTemplate.R for an example).
- Comment liberally and informatively (this is useful for sharing, especially with a *future* you…). Aim to describe what and why, more than how
- Modularize your code (i.e. create chunks); use indents and paragraph breaks to keep your code visually appealing
- Use consistent and informative names (for data, functions, results, etc).

III. Getting Started: Data Wrangling I

What is data wrangling?

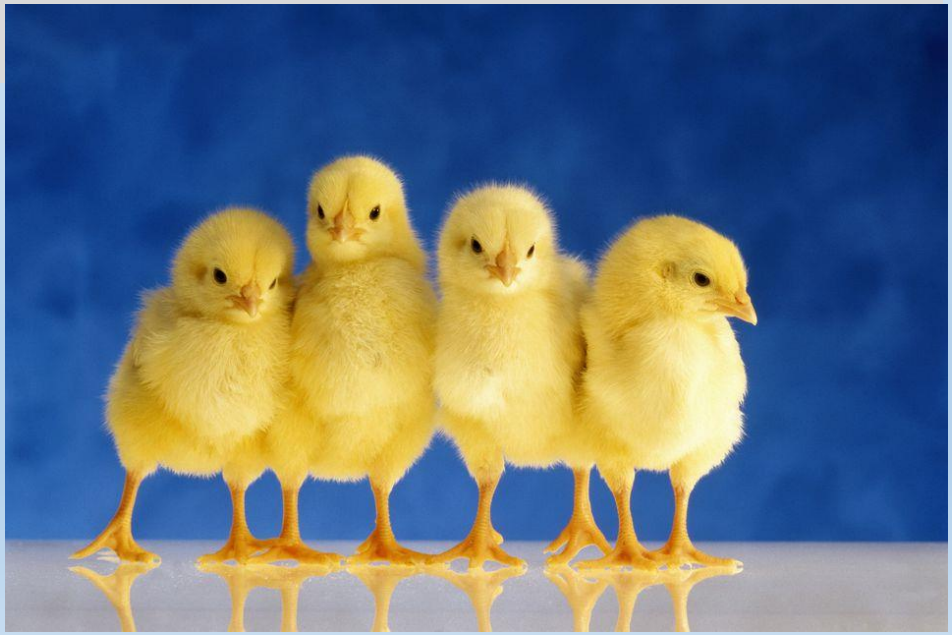
- Reading in data, examining the data file (ensure no errors), manipulating data to create summaries, different explanatory variables, exploratory plots.
- You will learn this by examining existing code, and trying to recreate similar code for another dataset…
- Chicks; 50 chicks weighed daily for 21 days
- Fed: Soybean, Sunflower, Linseed and Meatmeal (ugh)



III. Getting Started: Data Wrangling I

Instructions

1. Open ChickenScript2.R, and run the code line by line. Try to understand what the code is doing at each step.
2. Work with a partner
3. Raise your hand if you run into any problems.



III. Getting Started: Data Wrangling II

Nutrient **Net**work Experiment: global, distributed, expt

- Nutrient addition of N, P, K (all combos; i.e. control, N, P, K, NP, NK, KP, NPK)
- Exclusion of large herbivores (only control, NPK)
- ≥3 reps per treatment / site (30 plots)
- Biomass, by functional group, collected yearly
- You'll be working with data from Smith Prairie, site in Washington State 2 nutnet files).

III. Getting Started: Data Wrangling II

Instructions

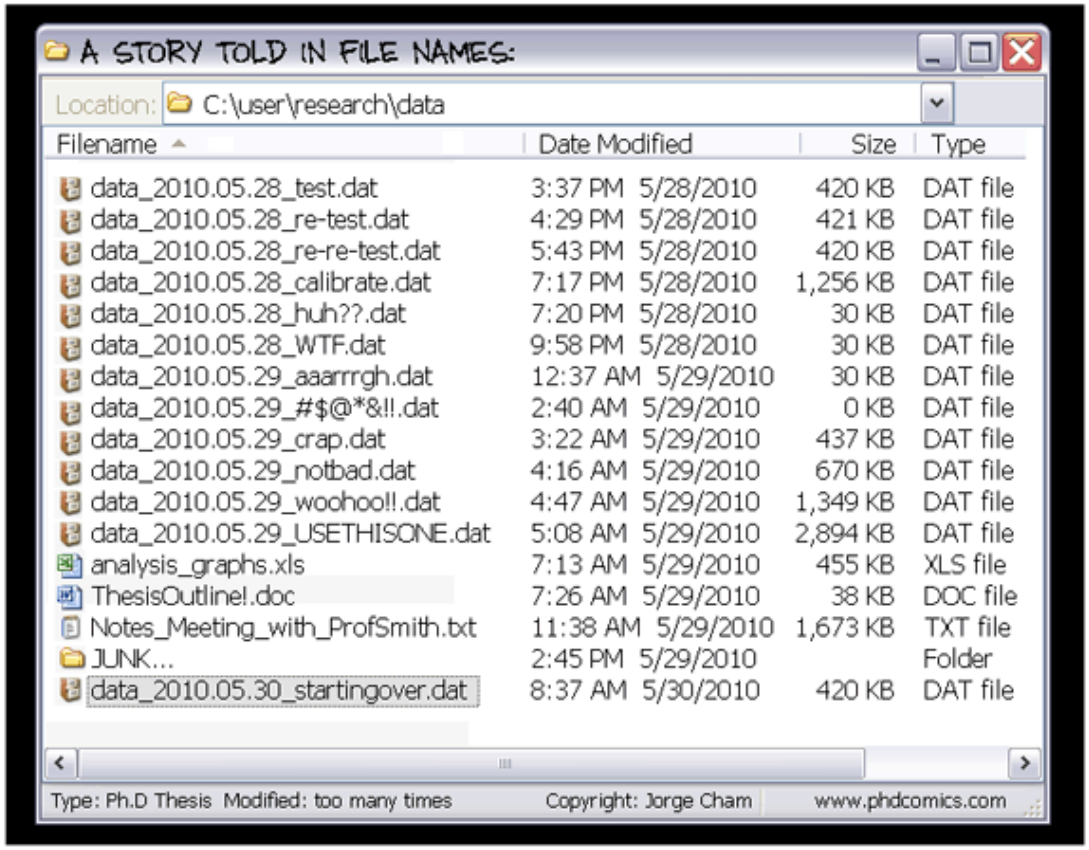
Write a script that does the following:

1. Reads in both data files & merges
2. Explore properties of the dataset
   A. # years, functional groups sampled.
   B. Exploratory plots: biomass vs. time; biomass vs. functional group; biomass vs. treatments
3. Subset data (e.g. final year, legumes only), and re-explore (as in #2)
4. A challenge: calculate the proportion of biomass that was legumes, explore by treatment (use tapply, merge, subset, create a new data frame)

IV. Further Topics

Data management

- Have a plan for data collection -> data entry -> error checking -> ‘clean’ data



IV. Further Topics

Data management

- Have a plan for data collection -> data entry -> error checking -> 'clean' data
- Include 'metadata'; in same file (e.g. worksheet in excel). Should include description of columns.
- Data structure: each row should be its' own observation, each column its' own variable.
- Anticipate the future (e.g. include year, 1st year)
- Order columns (explanatory, response), sort data
- No-no's: mixing data types (numbers + letters; NA's an exception); gaps / spaces, long names with spaces for column headers.
- Special consideration: consider NA's and true zeros

IV. Further Topics

Code sharing / reproducibility: Git and Github

- Git is a free online program that provides version control. Github is the webhosting version of Git.
- Keeps track of all versions of code, allows you to associate comments with changes, go back in time (to a previous version), and (if coding collaboratively) view changes by coder.
- More and more people are sharing code (in publications) by posting a link to a git repository.

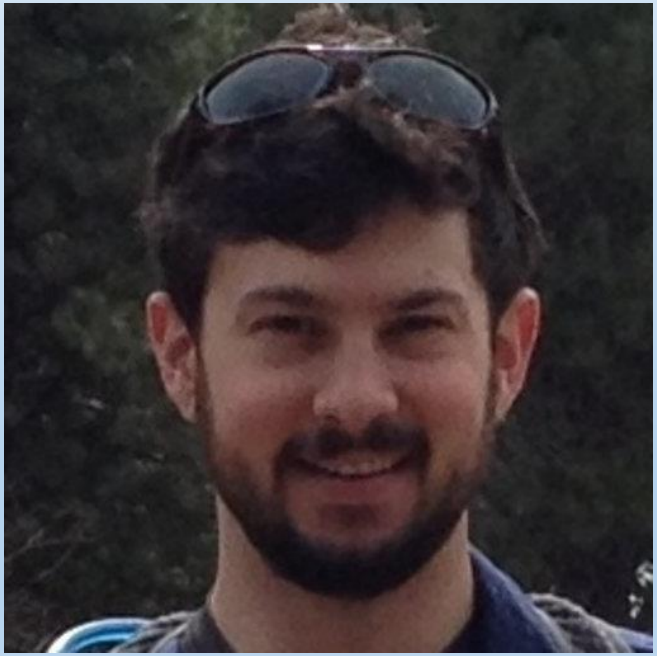
IV. Further Topics

Resources

- R and R studio online learning (hit home button in help)
- In-depth tutorial (web browser): Pirate-themed TryR via Code School
- Rstudio one page cheatsheets
- Software Carpentry has great workshops (free or virtually free), also online tutorials
- Books: I like Mick Crawley's **The R Book**. Native NZ son Hadley Wickham's book **R for Data Science** is also meant to be good – he also has a set of packages (check the Tidyverse) that are excellent for munging, merging, data.

Acknowledgments

Trevor Branch
UW SAFS – R course (SAFS 552, 553)



Clay Wright
UW Biology, R course (SAFS 552, 553) & Other

Workshop 1 (15/03/2018)