Ecological Analysis with R (Harvard Forest ***R***EU course)

---

http://tinyurl.com/HFRwkshp-notes

Day 1

- [Getting setup]
- [Coding basics]
- [R as a second language]
- [R Community = Packages + help?]
- [Plotting]

[Getting setup]

- Install the newest version of **RStudio** from here
- You do not need RStudio to run R. We do suggest it however for the improvements it offers over the default R GUI (a more seamless integration of scripts, console and graphs) along with some other additional features.
- While RStudio offers some point-and-click GUI alternatives, we will not be using or teaching them in this class.

Set up your project space

- Create a directory on your Desktop called Rwksp
- Create the following folders:
- data (This is self explanatory)
- src (This is where your code lives)
- docs (This is where other files that aren't code or data should live)
- I also suggest creating a README file for you to put some general notes about the workshop

Say hello to the Console...

Open-up Rstudio (or whatever flavor of R you are using) and head to The Console. This is a command line where you can enter text as if you are "talking" to the computer.

Activity: Try out some math

```
2+2
```

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

```
2-2
```

```
## [1] 0
```

```
2*2
```

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

```
2/2
```

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

```
2%*%2
```

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

Operators, Objects and Functions

Operators are characters that do things like math, logical operations, equalities, etc.

Objects (aka. variables) are names that can "store" information, like data.

Functions are special types of objects that can do things, they all have the form of **name**(*arguments*). For example:

```
help('help')
```

Data types and Object classes

There are multiple data types, but for now we just want you to be aware that are different types: characters vs. numbers vs. logical

If you do not use "" (or ''), R will read whatever word/combination of characters you type as an object. Therefore R will read `ABCD` as an attempt to call an object that doesn't exist, but `"ABCD"` as a character string.

Activity: What can we do with objects?

```
x <- 4
y <- 6
x + y
```

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

```
z <- "Hello Harvard Forest"
print(z)
```

```
## [1] "Hello Harvard Forest"
```

We will go more in on object classes later in the workshop, but for now there is one important class you need to know and that is vectors. Vectors are objects that have multiple numbers (or strings, or logicals, etc) contained within. To create a vector do the following: `objectname <- c(1,2,3,4,5)` . `c()` is a function that stands for concatenate.

Activity: What if we want to store more than one value at a time?

```
v <- c(1,2,3,4,10)
sum(v)
```

```
## [1] 20
```

```
v2 <- c(10,15,20,30,40)
v + v2
```

```
## [1] 11 17 23 34 50
```

[R as a second language]

- Learning R (or any computer language) is similar to learning a foreign language.
- You have to learn more than just the words: you have to learn the grammar, punctuation and syntax of the language.
- Just like in a foreign language, R has many commands that are cognates of English words. This is helpful when learning, as you often can figure out what a function does from its name alone.
- Though be wary, just like in human languages there are sometimes false cognates.

Can you guess what the following functions do?

```
mean(x)
max(x)
plot(x)
Map(x)
```

About arguments...

- Most functions have default settings for most arguments
- You don't always have to state the argument name, R infers from the ordering of the arguments
- Like Las Vegas, what happens in a function, stays inside a function (most of the time...), so you don't have to worry about argument names that are inside of functions over-writing your objects outside of the function
- Use tab to try and complete the names of functions and arguments

Activity: Pick a function and explore the arguments!

1. help() or ?
2. Use tab completion

Scripting

A script is a file that you can store your code for later use!

So, open up a new script file and save it to your Rwksp/src directory.

Commenting can be done using # because R will ignore the entire line to the right of it.

Style and Annotation

Be sure to use style (i.e. formatting of your code) and annotation that will make your code usable later by others and yourself!

1. R script names should always end in .R
2. Try and keep names simple, i.e. 3-5 characters, lowercase
3. Put spaces around operators and after commas and before left parentheses except after function names
4. Use <- for assignment, rather than =
5. For more info on appropriate formatting, see http://stat405.had.co.nz/r-style.html

---

Day 2

REUs, climate change and the ninja-pirate-zombie apocalypse

1. Get the data
2. Check the data
3. Prep the data
4. Visualize patterns
5. A bit about statistics and some tests
6. Export results
7. Make and save final figures

Warm-Up

Activity: Write a script that will do some useful math!

Object Management

After spending some time playing with R you may realize that you've created a lot of objects with assorted names, shapes, and sizes. Luckily there are some very useful functions that will help you keep track and clean up your workspace.

```
ls()
rm()
rm(list = ls())
```

Go ahead and clear your workspace now for the activities ahead.

1. [Loading/Entering Data]

The first thing we need to do for most analyses is get our data into R so that we can do things with it.

One way to do this is to enter it by hand. We can do this in a few ways: * Create multiple objects with our data:

```
temperature <- c(70,80,50,53,60,45) # in Fahrenheit
precipitation <- c(.01,1,0,2,.8,1.5) # in inches
```

- Maybe more useful is to create a dataframe:

```
weather <- data.frame(date = c("3/18","3/19","4/10","4/11","4/18","5/10"),temperature = c(70,80,50,53,6
                      precipitation = c(.01,1,0,2,.8,1.5) )
weather
```

```
##   date temperature precipitation
## 1 3/18          70          0.01
## 2 3/19          80          1.00
## 3 4/10          50          0.00
## 4 4/11          53          2.00
## 5 4/18          60          0.80
## 6 5/10          45          1.50
```

From this point on we will be using the data we collected from you all. http://tinyurl.com/REUdata-csv

Often data will already be entered into other formats or programs and we'll need to read it in so we can work with it.

Note: There are many functions for reading-in different types of files.

Remember: We have to make sure to save it as an object!

```
REUdata <- read.csv("../data/REUdata.csv")
```

2. [Checking Data]

The first thing we need to do after we read in our data is to look at it. This accomplishes a couple things:

1. We can get to know our data.
2. We can see if R has accidentally changed the data (often column names will be different if they were ented in Excel or other programs), and sometimes even the data may have been reformatted.
3. We can look for errors : often data will have quality issues. There can be typos, misclassifications, and missing data.

There are all kinds of tools available to us for data exploration. Here are some common ones that we'll use today:

- `summary(),hist(),table(), is.na(), complete.cases(), names(), head(),tail()`

Note: We will cover selection/indexing/subsetting of data in much more detail in the next class. For now, the main thing to pay attention to is that using the `[]` brackets after our object name and then the column name comes in quotations and after a comma: `ObjectName[,"ColumnName"]`

```
REUdata[,"Name"]  ####THIS IS HOW YOU SELECT A COLUMN
```

```
##  [1] K A N A K S M L A K A K P S R A I R M S A M
## Levels: A I K L M N P R S
```

```
head(REUdata)
```

```
##   Name I.have.used.R Programming.Experience X.Stats.Classes Zombocalypse
## 1    K         FALSE                    0.0              1          6.8
## 2    A         FALSE                    3.5       none yet          8.0
## 3    N         FALSE                    0.0              1          5.0
## 4    A         FALSE                    0.0              1          5.0
## 5    K         FALSE                    0.0              1          3.0
```

```
## 6    S         FALSE                   0.0            none          5.0
##   Pirates.or.Ninjas
## 1           Pirates
## 2            Ninjas
## 3           Pirates
## 4              Both
## 5           Pirates
## 6            Ninjas
```

```
table(REUdata[,"I.have.used.R"])
```

```
##
## FALSE  TRUE
##    11    11
```

Activity: Play with the data...

Wait. We're missing an important step.

1. Get the data
2. Check the data
3. Prep the data
4. Visualize patterns
5. A bit about statistics and some tests
6. Export results
7. Make and save final figures

Wait. We're missing an important step.

Day 3

Warm-Up

Copy/download the script we posted. Run this script and make sure you understand what each line is doing.

Remember our workflow

0. Get a question and decide on analyses
1. Get the data
2. Check the data
3. Prep the data
4. Visualize patterns
5. A bit about statistics and some tests
6. Export results
7. Make and save final figures

0. [Questions?]

What are some questions that people are curious about?

3. [Prep the data]

Let's look at the question:

Do poeple prefer ninjas or pirates?

```
table(REUdata$Pirates.or.Ninjas)
```

```
##
##                         Both                def. Pirates
##                            1                           1
##               Ninja Pirate!                       Ninja!
##                            1                           1
##                       Ninjas      Ninjas (no gingevitas)
##                            6                           1
##                      Pirates          Pirates, obviously
##                            8                           1
## Pirates! (of the caribbean)                     Pirates?
##                            1                           1
```

As we can see there are many responses that don't fall under the category Pirate or Ninja.

For our first pass, we want to get rid of any that aren't just clearcut "Pirates" or "Ninjas".

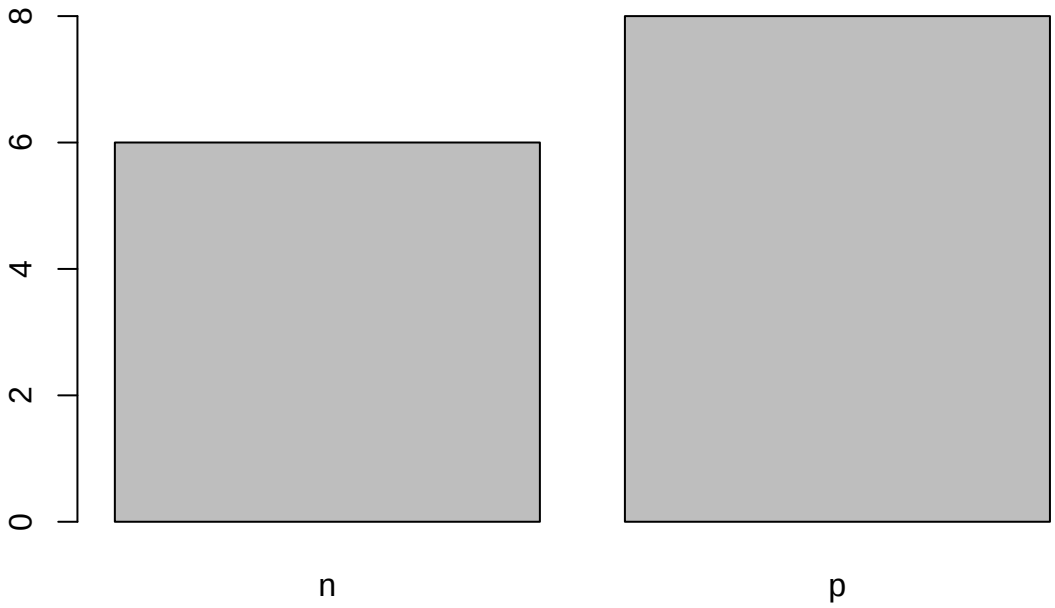
To do that we need to make a clean version of it without the other values.

```
pirates.or.ninjas <- c('p','n','p',NA,'p','n',
     'n','p',NA,NA,NA,'p','p',NA,
     'p',NA,'p','n',NA,'n','n',NA)
```

```
table(pirates.or.ninjas)
```

```
## pirates.or.ninjas
## n p
## 6 8
```

```r
barplot(table(pirates.or.ninjas))
```



Creating new columns

We can then append this new clean "Pirates or Ninjas to our dataframe".

Another way to think of this is that we're going to add a new column to our data frame that will have ONLY 2 possible values.

```r
names(REUdata)
```

```
## [1] "Name"                   "I.have.used.R"
## [3] "Programming.Experience" "X.Stats.Classes"
## [5] "Zombocalypse"           "Pirates.or.Ninjas"
```

```r
REUdata[,"clean.Pirates.Ninjas"] <- pirates.or.ninjas
names(REUdata)
```

```
## [1] "Name"                   "I.have.used.R"
## [3] "Programming.Experience" "X.Stats.Classes"
## [5] "Zombocalypse"           "Pirates.or.Ninjas"
## [7] "clean.Pirates.Ninjas"
```

Notice that we are using the same syntax that we used when we wanted to select a column beforehand. However, we are now asking for a column that doesnt exist, and then using the assignment operator `<-` to assign values to this column. This creates a new column with the values that we give it.

Logicals and Element Reassignments

But what about some of the other responses in the original Pirates or Ninjas column? Even though they didn't exactly match the response we were looking at some of them are still clearly either Pirates or Ninjas, and by not including them we're throwing away data points that we worked hard to colect! Instead of adding a whole new column for these few points, it makes more sense to change individual values in the column we just created. We will do this by selecting a row in addition to a column. Remember howe put a comma before our column names before? This is so that R knew it was a column and not a row you were selecting! The format will look like this: `ObjectName[rowNumber,"columnName"]`.

Logicals and Element Reassignments

```
REUdata$Pirates.or.Ninjas
```

```
##  [1] Pirates                     Ninjas
##  [3] Pirates                     Both
##  [5] Pirates                     Ninjas
##  [7] Ninjas                      Pirates
##  [9] Ninja Pirate!               Pirates?
## [11] Pirates! (of the caribbean) Pirates
## [13] Pirates                     Ninja!
## [15] Pirates                     Ninjas (no gingevitas)
## [17] Pirates                     Ninjas
## [19] def. Pirates                Ninjas
## [21] Ninjas                      Pirates, obviously
## 10 Levels: Both def. Pirates Ninja Pirate! Ninja! ... Pirates?
```

```
##Looking at the output we can see that the 14th value is "Ninjas!".
##We did not include this before but I think it's safe to
##assume that this should count as a 'n'.

REUdata[14, ]
```

```
##    Name I.have.used.R Programming.Experience X.Stats.Classes Zombocalypse
## 14    S         FALSE                    1.5              2            7
##    Pirates.or.Ninjas clean.Pirates.Ninjas
## 14            Ninja!                 <NA>
```

Logicals and Element Reassignments

```
REUdata[14,"clean.Pirates.Ninjas"]
```

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

```
table(REUdata$clean.Pirates.Ninjas)
```

```
##
## n p
## 6 8
```

```
REUdata[14,"clean.Pirates.Ninjas"] <- "n"
REUdata[14,"clean.Pirates.Ninjas"]
```

```
## [1] "n"
```

```
table(REUdata$clean.Pirates.Ninjas)
```

```
##
## n p
## 7 8
```

Activity: Replace another element of the Pirates/ninjas data

Now, while this works, it's a bit slow to have to manually select each element you want to replace. This is where logicals come in.

Logicals: realational operations

```
x <- 10
y <- 73
```

Logicals: realational operations

```
x < y
```

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

```
x > y
```

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

```
x <= y
```

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

```
x >= y
```

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

```
x == y
```

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

```
x != y
```

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

Logicals: vectors

```
x <- 1:10
x < 3
```

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

```
x == 3
```

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

Logicals: vectors

```
REUdata[,"Pirates.or.Ninjas"] == 'Ninjas'
```

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

```
REUdata[,"Pirates.or.Ninjas"] == 'Pirates'
```

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

```
!(REUdata[,"Pirates.or.Ninjas"] == 'Ninjas')
```

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

```
!(REUdata[,"Pirates.or.Ninjas"] == 'Pirates')
```

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

```
REUdata[,"Pirates.or.Ninjas"] == 'Pirates' | REUdata[,"Pirates.or.Ninjas"] == 'Ninjas'
```

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

```
REUdata[,"Pirates.or.Ninjas"] == 'Pirates' & REUdata[,"Pirates.or.Ninjas"] == 'Ninjas'
```

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

Activity: Make anything not Pirate or Ninja into an NA

```
REUdata[!(REUdata[,"Pirates.or.Ninjas"] == "Pirates" |
   REUdata[,"Pirates.or.Ninjas"] == "Ninjas"),"Pirates.or.Ninjas"] <- NA
```

Statistics: Testing for generality

- Statistics are used to make sense of data
- Statistical tests are used to generalize the results obtained from a sample
- Frequentist statistics is one school that defines probabilities as long run frequencies and uses the probability of "null" hypotheses to answer statistical questions

Testing for a difference in pirate/ninja preference

```
counts <- table(REUdata[,"clean.Pirates.Ninjas"])
pn.chi2test <- chisq.test(counts)
```

Export data

While in general it's preferable to output figures or summary files, there are times when you'll want to save your cleaned and processed data. One way to do this is to output in the same format that read it in with`write.csv()`.

Another option is to use `save()` which can save your R object as is. This has some uses if you have a format that's not amenable to writing to a .csv or other common format.

Export figures

One of the great things about R is that you can make highly customizeable and presentation/publication ready figures with ease. But once we make them we'll need to get them out of R.

One easy way to do this is to right-click on the figure itself (in either R or Rstudio) and paste it wherever you want. Alternatively you can use commands to output figures: `{r,eval = F} pdf() # or jpeg() or many other common formats plot()#Code to create figure dev.off() #this will close and save the image.`

Export Results

```
save(pn.chi2test,file='../results/pn_chi2.Rdata')
write.csv(unlist(pn.chi2test),file='../results/pn_chi2.csv')
pdf(file='../results/pn_barplot.pdf')
barplot(table(pirates.or.ninjas),
   names=c('Ninjas','Pirates'),ylab='REU Student Votes')
dev.off()
```

```
## pdf
##   2
```

Activity: more questions!

Pick one of the questions we came up with and do your best to answer it!

Go through the full workflow we've learned and see if you can make it all the way from a question to figures and stats!

Writing functions and modularizing your code

If you're doing an operation a lot in your script, think about making it a function and put it in a another file that you can call with source('helpers.R').

```
x <- rnorm(n=100,mean=20,sd=2)
mu <- function(x='some numbers'){sum(x)/mean(x)}
mean(x)
```

```
## [1] 20.14878
```

```
mu(x)
```

```
## [1] 100
```

Life-long leaRning

- Packages, CRAN and the R universe
- Find a good cheat sheet
- Google your errors away
- Find a good script editor with syntax highlighting
- Hackathons! Coding for the greater good...

Resources

- http://www.yeswecode.org/
- https://www.codeforamerica.org/
- Gaming to Make a Better World

. . . and beyond!

- The art of modularizing code
- Subsetting data
- Character searching
- String manipulation
- Running a leaflet or shiny app
- Loops and apply functions
- Lists and flexible data structures