**Introduction to R, RStudio, and RMarkdown**

R is freely available statistical software for ecological analyses.

This book is a good and gentle intro to using R:

Beckerman, A.P., D.Z. Childs, and O.L. Petchey. 2017. *Getting Started with R: An Introduction for Biologists (2nd ed.)*. Oxford University Press.

It teaches you how to enter data, how to pull out and analyze certain variables from a large dataset, how to make simple plots (scatterplots, bar charts), and how to do some basic statistical analyses (t-tests, ANOVA). If you have never used R, I highly recommend it.

**Load R – the latest version is 4.0.3**

[http://cran.r-project.org](http://cran.r-project.org/)

Open the downloaded .exe file to install R on your desktop. (If you can, right-click and Run as Administrator.) If you are asked to pick a CRAN Mirror, just select one. (CRAN stands for Comprehensive R Archive Network.) I usually pick one from inside Texas or the U.S.

**Load RStudio – the latest free desktop version is 1.3.1093**

<https://rstudio.com/products/rstudio/download/>

Select the free desktop version. Open the downloaded .exe file to install RStudio. (If you can, right-click and Run as Administrator.)

(If you have already worked with R/RStudio before, then make sure to update your installed packages: in RStudio, select Tools/Check for Package Updates.)

**R, RStudio, and R Markdown: What’s the difference?**

R is a programming language and software environment that includes a diverse array of analyses employed in ecological studies and for exploratory data analysis. RStudio has a graphical user interface for using R. You can use R without RStudio, but you need R to run RStudio. R Markdown allows you to make text documents with embedded R code, in RStudio. It comes already installed in RStudio. You will use R Markdown to create your weekly homework assignments.

**R: A primer**

In all materials for this class, I will use Courier New font to indicate R code (also called an R script). Sections of R code and output will be set off in a colored box like these:

Example R code

and output

**Download data needed for this class:**

Create a folder somewhere on your computer where you will place all materials for this class. (For best computer hygiene, make sure that folder name has no spaces, numbers, or symbols.) This does NOT need to be in your R folder. Once that folder is created, then you’ll tell RStudio that that’s where it will find the data you’ll be analyzing, and that’s where it should put the output it will generate.

Go to the course website and there will be some data files as well as a Word document that lists where each data file came from; download them all to your folder:

butterfly\_pres\_abs.csv

butterfly\_sites.csv

brycesite.R – this is a zipped folder; save to your computer and then extract the brycesite.R file inside

bryceveg.R – this is a zipped folder; save to your computer and then extract the bryceveg.R file inside

div\_data.csv

GBbiol.csv

GBsite.csv

grassland.community.csv

Ground\_beetles\_abundance.csv

Ground\_beetles\_habitat.csv

plot.metadata.csv

data\_sources\_info.docx

Add more data files from Gardener (2014) to do more examples

**Working in R:**

Open RStudio. (If you had been working on something there earlier, it may pop up. If so, close any open scripts and sweep the working environment.)

In the upper left corner, select File / New File / R Script. A window will appear, which is where you will type your commands.

(Notice how there is an option to open a new RMarkdown file; do not do this yet.)

**Set your working directory:**

In your script window, you will set your working directory (note carefully the direction of the slashes!):

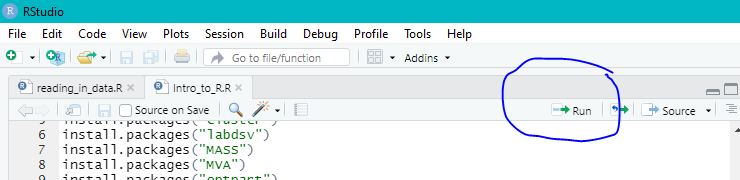
Setwd(“C://path/to/your/course/folder”)

**Important note:** If you try to copy/paste the pathname from Windows, **the slash marks in the path must be changed from \ to /**. (Mac and Linux should already have the slashes facing forward.) This is a legacy of R's development under UNIX, and you'll just have to remember it. (If you copy/pasted the URL of your working folder, you’ll have to change the direction of each slash!) And don’t forget the double slash after C://.

**Important note:** If you copy/paste text (e.g. from this document) into your R session, be advised that **some formatting cannot be read by R**. If you have a line of code that will not run (has a little red stop sign with an X in it next to the line of code in your script), first, check it for typos. If there are no typos, then ask yourself whether you copy/pasted anything. If so, type the line out manually. **Typos** and **formatting** are the two most common nitpicky problems in R.

**To execute (i.e., run) a line of code:**

Click anywhere on that line, or select multiple lines of code, and hit the Run button:



**Save your R script:**

Note how as soon as you type anything into your script window, the filename Untitled1 in the upper left tab will turn red and be marked with an asterisk. As this point, select File / Save and **save your script as filename.R** (changing “filename” to something appropriate, but with NO spaces, NO symbols) in your working directory folder. In future, you can just double click on that file and RStudio will open, or you can open RStudio and select File / Open / Open File and navigate to your \*.R file.

**Download packages:**

In R, a package is code developed to conduct specific analyses. There are hundreds of packages available at the CRAN website; you can browse some at

<https://cran.r-project.org/web/packages/available_packages_by_name.html>

And for ecology specifically:

<https://cran.r-project.org/web/views/Environmetrics.html>

Each package should have a web or PDF documentation file that explains it, also available at the CRAN website.

We don’t have time to cover every time of community ecology analysis in this course! Here is a list of some R packages that are commonly used for topics that we won’t cover in this course:

Species distribution modeling

* *SSDM*
* *HMSC*
* *Dismo*
* *Raster*
* *MIAmaxent*
* Tutorial: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.190.9177&rep=rep1&type=pdf>

Phylogenetic analyses

* *ADIV*
* *PGLMM*
* *Caper*
* *Phytools*
* *Treeio*

Trait-based functional community analyses

* *FD*
* *Vegan*
* *Ade4*
* *Picante*

Metacommunities

* *Metacom*

Bioinformatics and/or data mining

* *Dredge*
* *MuMIn*

Install.package(“vegan”)

# This installs the R package named vegan on my computer.

# You only need to install a package once.

Library(vegan)

# This opens the installed package.

# You may have to use this command for each new R session.

# Notice that you don’t need the quote marks for the library

# statement.

The code above introduces an important convention: **any line that starts with # is NOT read by R**. Thus, # is used to annotate one’s code. This is necessary for anyone else to be able to understand your code. And keeping notes about your analyses will make your life easier in the long run! For example:

# Last updated: 10 July 2020

# Data collected by Nancy McIntyre in Colorado Springs in

# summer 2019.

# All measurements were made using an Acme model T1000 pH meter.

It's important to distinguish between INSTALLING a package (which puts a copy on your computer and LOADING a package, which loads an installed package into your copy of R. First you install (install.package()), then you load (library()).A library refers to the place where the package is contained (a folder on your computer), whereas a package is the actual code that runs functions. A package is thus a like a book and a library is like a library; you use library() to check a book out of the library.

By convention, package names are *italicized*. They are also case-sensitive!

Use the install.package() and library() functions to download the following packages that we will use in this course (note that names are case-sensitive!!):

*labdsv* – good package for multivariate analyses, developed by a botanist

*MASS* – data and functions for many modern statistical applications

*MVA* – for various multivariate analyses

*optpart* – we will use in cluster analysis

*picante* – for phylogenetic community analyses (also does other analyses with other, bundled packages)

*stats* – basic stats package for R

*vegan* – currently the best all-around package for community ecology analyses

**Later in the semester we will also be using:**

*cluster* – for cluster analyses

*BiodiversityR* – does many of the same analyses as *vegan* plus suitability mapping

*plyr* – tools for manipulating data sheets (arranging, splitting, combining, etc.)

**Other packages that may be of interest/use to you in your research on ecological communities:**

*ade4* – for various types of ordination

*ape* – especially useful when dealing with phylogenetic data

*bdvis* – for gathering data from collective databases such as the Global Biodiversity Information Facility (GBIF)

*coenoflex* - simulates the composition of samples of vegetation according to gradient-based vegetation theory

*fso* – fuzzy set ordination to relate the composition of samples to possible explanatory variables

And there are many others!

**Variables:**

Variables in R are assigned a value with an arrow or an equals sign. For example,

number.species <- 137

number.species = 137

In the above code, you created a variable named “number.species” that you assigned a value of 137.

More examples of variable assignments (numeric, scientific notation, categorical, and Boolean logic true/false, respectively):

x <- 1.2345

small.value <- 1.0e-10

species.name <- 'Pinus contorta'

conifer <- TRUE

In exponential notation, 1.0e-10 means 1.0-10. Character variables, called "strings," should be entered in quotes (single or double, it doesn't matter as long as they match). Finally, note that the word TRUE is NOT surrounded by quotes. That’s because this is not the WORD TRUE, but rather the VALUE TRUE. Logical variables can only take the values TRUE or FALSE.

You do not have to tell R what kind of value (integer, real, or character) a variable will contain; it can tell when the variable is assigned. R will only allow the appropriate operations to be performed on a variable. For example:

species.name + 37

Error in species.name + 37 : non-numeric argument to binary operator

R did not allow me to add 37 to species.name because species.name was a character variable.

**Read in an existing dataset:**

Now you can read in a data file (an existing spreadsheet). We will work more on this next time, but the spreadsheets we will work with will be Excel or text files. (R can also read in other filetypes.)

Today we will start with an example data set called bryceveg.R, available on the course website. This is an ASCII text file of a site x species database containing plant species abundance data from sites (sampling plots) in Bryce Canyon National Park, Utah. (You can save ASCII files with the .R extension, or you can keep them with their original .txt. R can read either.) We will learn more about this dataset in the next lesson.

We will convert this text file containing the data to an R object, which we’ll name veg. To open the file, specify the path to it.

veg <- read.table("C://your/file/path/bryceveg.R",header=TRUE)

Let’s take a look at this dataset:

veg

# shows you the contents of the R object veg

dim(veg)

# to get the dimensions of this dataset

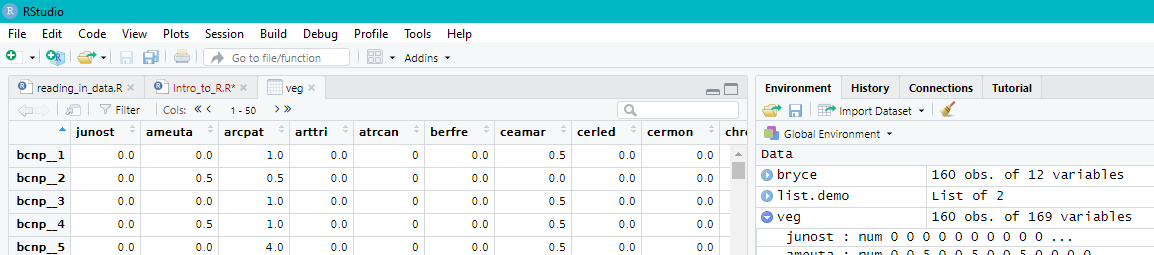
names(veg)

# to get the columns names (species in our case)

row.names(veg)

# to get the row names (plots in our case)

You can click on veg in the Global Environment window, and the actual data table will open (look for the tab to the left):



This site x species data table is set up in exactly the way most community analyses require: **sites (or sampling plots, sampling units, or observation) as rows, and species (or environmental variables) as columns**.

**Data structures:**

In R there are 4 primary data structures; I will go through each in turn:

1. **vectors** --- vectors are one-dimensional ordered sets composed of a single data type, such as numbers or strings (character variables).
2. **matrices** --- matrices are two-dimensional ordered sets composed of a single data type.
3. **data frames** --- data frames are one- to multi-dimensional sets and can be composed of different data types (although all data in a single column must be of the same type). Each column and row in a data frame may be given a label or name to identify it. Data frames are equivalent to database or spreadsheet.
4. **lists** --- lists are compound objects of associated data. Like data frames, they need not contain only a single data type but can include strings (character variables), numeric variables, and even matrices and data frames. In contrast to data frames, list items do not have a row-column structure, and items need not be the same length; some can be a single values and others a matrix. You can think of a list as a named box to put related objects into.

Vectors, matrices, data frames, and lists should be identified by a name that is unique and long enough to clearly identify the contents of the structure. Names can consist of letters, numbers, and the character "." but they cannot start with a number or include the character "$" or any arithmetic symbols, as these have specific meanings in R.

**Vectors and matrices:**

Vectors are often read in as data or produced as the result of analysis, but you can produce one simply using the c() function, which stands for "combine." For example:

demo.vector <- c(1,4,2,6,12)

produces a vector named demo.vector of length 5 with the values 1, 4, 2, 6, 12.

Individual items within a vector or matrix can be identified by a number (or numeric variable) within square brackets. For example, if the number of species per plot is stored in a vector you named A, then A[37] would equal the number of species in plot 37.

Matrices are specified in the order "row, column" such that veg[23,48] equals the datapoint that exists in row 23, column 48 in matrix veg. **Since you have a matrix called veg, what is the value of the datapoint at that location?**

Individual rows or columns within a matrix can be referred to by implied subscript, where the value of the desired row or column is specified, but other values are omitted. For example, veg[,3] represents the third column of matrix veg, as the row number before the comma was left blank. Similarly, veg[5,] represents row 5, as the column after the comma was left blank. In addition, a number of specialized subscripts can be used:

veg[]

# returns all rows and columns of matrix named veg

veg[a:b,c:d]

# returns a submatrix of veg from row a to b and column c to d

# where a, b, c, and d are numbers

It's also possible to specify specific subsets of rows and columns that are not adjacent:

veg[c(1,7,10),c(3,6,12)]

# returns a submatrix consisting of rows 1,7, and 10, and # columns 3, 6, as 12 from the matrix veg

Notice how if your note goes beyond one line, each line needs to start with #.

**Data frames:**

Now read in a different ASCII data file called brycesite.R. This file contains information about the sampling sites and is in the form of a data frame. Use it to create a new R object called bryce:

bryce <- read.table(“C://your/file/path/brycesite.R”, header=TRUE)

Examine it by clicking on it from the Global Environment window.

Here’s a few rows and columns to show the different data formats present, which is what separates a data frame from a matrix:

plot elev aspect slope text

1 1300 240 30 loam

2 1640 170 20 clay.loam

3 1840 NA 24 silty.clay.loam

Multi-word variables like "clay loam" need to be connected. The R convention is to connect with a period, as shown above, or with an underscore (clay\_loam).

**Why is veg a matrix but bryce a data frame?**

Data frames can be accessed like matrices, but can also be accessed by data frame and column or field name, without knowing the column number for a specific data item. For example, in the Bryce Canyon site dataset, there is a column labeled "elev" that has the elevation of each sample plot. This column can be accessed as bryce$elev, where "bryce" is the name you gave the data frame, "elev" is the name of the field or column of interest, and the "$" is a separator to distinguish data frame from field.

NA stands for missing data. We will learn more about missing data in a future lesson.

**Lists:**

A list is a compound object composed of associated data. Items within a list are generally referred to as components. Similar to data frames, components in a list can be given a name, and the component can be specified by name at any time. In addition, components can be specified by their position in the list like in a vector, but with double brackets [[ ]] rather than single ones [] like with vectors. Here’s a simple demo using the list() function:

list.demo <- list(species\_names=names(veg))

list.demo

$species\_names

[1] "junost" "ameuta" "arcpat" "arttri" "atrcan" "berfre"

[7] "ceamar" "cerled" "cermon" "chrdep" "chrnau" "chrpar"

[13] "chrvis" "eurlan" "juncom" "pacmyr" "pruvir" "purtri"

. . . . . .

. . . . . .

. . . . . .

[157] "sclwhi" "senmul" "sphcoc" "stapin" "steten" "strcor"

[163] "swerad" "taroff" "thafen" "towmin" "tradub" "valacu"

[169] "vicame"

**R vector and matrix operators:**

It is possible to perform fairly sophisticated routines in R with a little programming. For example:

logveg <- log(veg+1)

# creates a new matrix called logveg with all values as the

# logarithm of the respective values in veg

# (+1 to avoid log(0), which is undefined)

# (i.e., logveg is the log-transformed version of veg)

In addition, R supports logical subscripts, where the subscript is applied whenever the logical function is true. Logical operators include:

* > for "greater than"
* >= for "greater than or equal to"
* < for "less than"
* <= for "less than or equal to"
* == for "equal to"
* != for "not equal to"
* & for "and"
* | for "or"

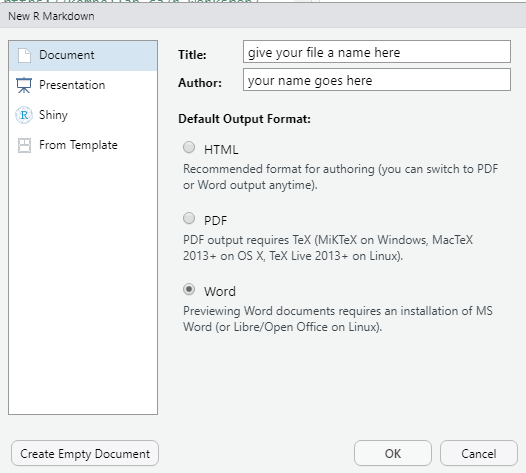
We will use some of these things in the coming weeks.

**Create your RMarkdown file:**

RMarkdown is a file format (filename.RMD) for making dynamic documents that include text as well as executable chunks of embedded R code. This is how you will generate the documents you will submit as your weekly homework assignments. A good RMarkdown document should provide a reproducible log of your code, with comments and relevant output so the reader knows what you did and how you did it. It is included automatically in RStudio; it can be installed as a package (*rmarkdown*) in regular R if you don’t want to use RStudio.

You should currently be working from your filename.R script. (You may have noticed that you can start with File / New File / RMarkdown, so you may try that in future. For this class, you’ll need to have RMarkdown files, but normally you only work from R scripts. That’s why I had you start this lesson in the way I did, with creating an R script!)

Select File / New File / RMarkdown. A window will open with areas for you to populate. Select to save your file as a Document (left column) in Word format (bottom button option) or HTML (the default; top button option):



A new window will appear called Untitled1\* in red. It will have the info you entered for title and author as above, as well as the date and some RMarkdown instructions. Save your file (give it a name) and it will automatically save as an RMD (RMarkdown) file, not a Word (or HTML) file.

You can now select all of the text from your R script, copy and paste it into the RMD file below all of the intro stuff (which you can delete if you wish starting after the title/author/date/output info).

In your R script, any line that started with # was not executed. In your RMD file, you don’t have to use those and can now add any additional information you wish in plain text. This allows you more room to provide explanations.

For sections of your code, however, you must set them apart with the following (three backwards apostrophes [usually found on the key left of your 1 key on your keyboard] and r in curly brackets):

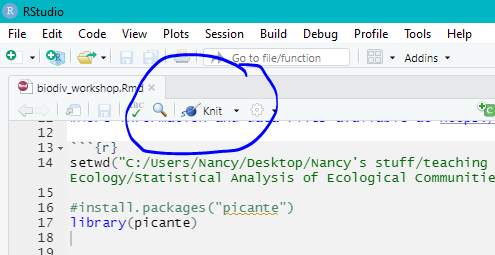
```{r }

*your code goes here*

```

Notice how when you add the ```{r}, all of the subsequent text is shaded. That shaded text indicates a chunk of executable R code. To end the chunk, remember to put ``` after it. Then you can type more explanatory text and go on to the next chunk.

Now to run your code to produce the output tables or graphs and save it as a Word document, click on the Knit button:



**Important oddity: RMarkdown will not save to Word if you have chunks with install.packages()!!!** You will get a red error message in your console instead. (I don’t know why, this is just some peccadillo of RMarkdown.) You can have those in your script file but mark them with #.

Your RMarkdown file will now be saved as a Word document in your working directory. This is what you will turn in via email for your weekly assignments.

Next time, you can just start with an RMarkdown file rather than an R script if you wish. You just have to remember to set aside executable code with ```{r} and ```. You’ll need to figure out whether you feel more comfortable working in .R and then in .RMD, or from .RMD from the start. The two tutorials on RMarkdown that I have links to on the course website have lots of other ways of including materials, making them look nice, etc.

**When you exit R:**

R will ask whether you want to save your Workspace; if you say yes, save the file as a .RData file. This will allow you to pull up all your work done thus far. You can also save just your code as a .R file. You can also just save your results (figures, tables); I suggest putting them into a subfolder within your working directory folder. Thus, you would have:

1. Your\_workspace\_name.RData - the R workspace containing all results
2. Your\_script\_name.R - a text file with all R commands you ran to get the results
3. results/ - a directory with all the figures and results tables that were generated
4. Your\_RMarkdown\_file.RMD
5. Your\_RMarkdown\_file.docx - Word document.

**Assignment:** due 0800 Monday, 8 February

This assignment gives a demo of a few more things that R can do and gives you a bit more practice in using RStudio and RMarkdown.

This assignment will perform some simple univariate statistical plotting and tests. If you don’t know what a median, quartile, or p-value are, then look those up first.

R itself contains some datasets; one is on irises.

Data(iris)

#loads the iris dataset

(If you want to find a list of datasets available in R, listed by package, type:

data()

A list of datasets in R is also maintained at <https://vincentarelbundock.github.io/Rdatasets/>.)

To find out more about the iris dataset:

?iris

This ? command is VERY useful in RStudio!

Head(iris)

#shows the top (the “head” of the dataset)

**Q1. Based on what you can see of the dataset, which of the four forms of data is iris and how can you tell?**

Summary(iris)

#gives you the summary statistics for each column of data (min,

#max, median, mean, and quartile values as well as sample sizes)

#To plot a histogram of sepal width:

hist(iris$Sepal.Width)

A histogram is a plot of the frequency of occurrence (Y axis) of various data values (X axis), in this case, various sepal widths. The highest bars of the histogram indicate the sepal widths that were most commonly observed in the dataset.

**Q2. Now you write code to make a histogram of petal width.**

**Q3. Describe the differences you see in the data distributions of sepal width and petal width.**

#To make a boxplot of petal length by species: boxplot(Petal.Length ~ Species, data = iris, xlab = "Species", ylab = "Petal length")

A boxplot displays the data range (in this case, for petal length by species), with the thick bars indicating the median value, the gray-shaded boxes indicating the interquartile range (i.e., values between Q1 and Q3 [the 25th and 75th percentiles]), the “whiskers” indicating the data range (min to max) excluding outliers, and circles outside the whiskers as outliers (values that fall outside 1.5 times the interquartile range above the upper quartile and below the lower quartile). These plots allow you to quickly compare distributions across categories (in our case, species). You can use ?boxplot to learn more.

**Q4. Interpret what the boxplots mean for the iris data in terms of which species has the longest petals, is there overlap in the data distributions by species, etc.**

**Q5. Now you build a boxplot of sepal width by species and interpret it in words.**

#Use a linear model to test whether sepal width differs

#statistically among species:

iris.lm <- lm(Sepal.Width ~ Species, data = iris)

#analysis of variance for significance of species effect model:

anova(iris.lm)

(Remember that you can use ?anova if you’re unfamiliar with what an analysis of variance is.)

**Q6. Interpret the output. Does it support or contradict the boxplot you made?**

#Plot model diagnostics (residuals, Q-Q plots, etc.) that

#are used to determine model goodness of fit; for more info

#on these, check out #https://data.library.virginia.edu/diagnostic-plots/

plot(iris.lm)

(You can look up more about interpreting model diagnostics at the website referenced in the note in the code above.) The model diagnostics look OK, which means we can be reasonably confident in the fit of our models and in our results.

Make an RMarkdown Word file of your analyses on the iris data and turn that in. Be sure to include your answers to the questions asked!

Turn in your assignment as a Word document via email to [iroro.tanshi@ttu.edu](mailto:iroro.tanshi@ttu.edu) no later than 8:00 a.m. on Monday of next week. In your email, please include the following as the Subject line:

Assignment on Intro to R