Class 1. Getting Started with R
Goals:
(1) get R installed and running,
(2) be comfortable getting around the R framework (console, editor, quartz, and help windows)
(3) understand the differences among R types and structures (numeric, character, factor, logical; vector, data frame, list, matrix, etc.)
(4) understand basic R notation and operators to be able to do simple functions,
(5) import your own data into R from a spreadsheet

(1) INSTALL R on your computer

1. Go to the R website at http://www.r-project.org/.
2. In the middle of the page, under Getting Started, click on download R.
3. Choose a nearby mirror site from the list (e.g., http://cran.cnr.Berkeley.edu)
4. From the Download and Install R box, choose Linux, MacOS X, or Windows, as appropriate.
5. Download the most recent package that your system can handle (currently 3.1.2).

(2) FOUR R WINDOWS in the R framework

The R interface has four key windows that you will use:

R Console: This is where you enter and execute commands, and where textual output of your analyses appear.

Editor Window: This is a good place to write your code, or open it from a saved file. Copy and paste from here into the console window. The advantage of writing the code here is that it provides automatic helpful hints to the formatting of functions. On a Mac, highlight the code you want and then use command-return to submit text directly from editor to console; in Windows, use control-R. (Caution: using Word or similar word processing software to write R code; it is problematic if it automatically creates curly quotes, or fixes capitals or spelling.)

Quartz Window: This is where graphical output goes by default. On a mac, use right and left arrows to scroll through the plots you make in a session. On a PC you need to (1) create a graphic, (2) with the graphic window active, go to the *History* tab, next to *File.* (3) Click on *Recording* and make sure it is checked. (4) Now each graph in the session is recorded; use PgUp dn PgDn to scroll.

R Help: R has very useful help functions. For instance, if you what help for the function plot, either

```
?plot
```

or

```
help(plot)
```

will open up an R Help Window with detailed, standardized instructions on how to use that function. The format for the help at first is a bit cryptic, but all the functions take the same format. Once you get it, you can quickly learn what you need for any function.

Alternatives to the built-in Editor:
On the mac, I usually use the R Editor Window because it provides helpful contextual support, and facilitates submitting code.

Many PC users as well as a lot of mac users prefer to use other editors, particularly RStudio. Go to http://rstudio.org/ and download R Studio for free. This provides some really nice ways to handle your files, organize windows, get help, load libraries, code completion, etc. Lots of R-users swear by it, and I think especially for PC users it has some great advantages. I find that the way I work with the different windows and use R on a mac I actually prefer using the basic interfaces. It is completely up to you if you want to use R studio, but we won't be focusing on using it in this course.

Optional X11 install for mac if you want to use Rcmdr
Starting with Mountain Lion (and newer) the mac OS does not automatically include the X11.app window system software. This is not needed for R, unless you want to run the Rcmdr R interface (we won't be using it in this class, but it provides many useful shortcuts and menu-driven analyses). If you want to install Rcmdr, first install XQuartz, the open-source version of X11 that Apple has now made available. You can install it from http://xquartz.macosforge.org/landing/.

An important caution about copying and pasting from word processors
Most word processors now default to "smart quote", the kind that angle in around whatever is being cited. R does not recognize those as quotes – it needs the vertical dumb quotes.

```
“smart quotes”
"dumb quotes"
```

You can turn this off in your word processor, just use the R editor (or an editor like Text Wrangler) or just be really careful to fix them. But basically don't use Word to write R code.

(3) BASIC TYPES, STRUCTURES, and OBJECTS in R

R has a variety of *Data Types* and *Data Structures*: unfortunately, it is ESSENTIAL to know what types and structures you are using, and what is required for particular functions.
The most common types are *numeric, character, logical,* and *factor (i.e., a category)*.
Homogeneous structures contain elements of only one type (e.g., all numeric). The most common homogeneous structures are *vectors* (1-dimensional) and *matrices* (2-D).
Mixed-type structures can combined different types of data into one structure. The most common is the *data frame*, which may mix factors (e.g., treatment names) and numerical data (e.g., measurements). A *list* is a structure that can contain many kinds of data; analytical output is usually organized into a list. A vector, matrix, data frame, or list is an *object* you can manipulate.

Vector: A 1-dimensional object containing some number of elements of the same type (e.g., a string of numbers). A single value is a vector of length one.

Matrix: Matrices organize data into rows and columns, where all the elements of a matrix must be of the same type (e.g., all numeric). Note that some functions require data to be in a data frame, whereas others require data to be organized as a matrix. A vector is a 1-dimensional matrix. R can also handle n-dimensional arrays.

Data frame: This is the most common way to organize data. Data frames organize data into columns (variables), with each row representing a single record (i.e., individual). Columns can be of different types (e.g., numeric, character strings, factors). Note that some functions require data to be in a data frame, whereas others require data to be organized as a matrix.

List: You can combine objects of any types into a single object, using lists, for easy output. For instance, if you want to combine into one object the data frame and output for regressions that you saved as objects regAbyB and transregAbyB, you can combined them into a single object Y as Y <-list(MyDF, regAbyB, transregAbyB). Typing "Y" will print them out in that order. The output of many functions are stored in lists; to see components of list Y, type names(Y). Note that a data frame itself is a kind of list because it includes both names and the data.

Tables: Tables are compact structures that summarize data (e.g., tabulations of counts of different groups). They may look superficially like a data frame, but are not, and different functions require one or the other. You can convert between them with the as.data.frame and table functions.

MORE USEFUL R TERMINOLOGY

Function: An R statement that does something to an object. It acts on whatever is in parentheses following the function name. There may be various modifying attributes included inside the parentheses, in addition to the data object (e.g., probs in the quantile function, below)

```
sqrt(25)   #the function sqrt acts on what is in the parentheses- returns 5.
a<-36 #create a vector of length one
a
sqrt(a) #here the function sqrt acts on the value of a
b<-c(4,8,16,24,30,36) #create a vector of six number
b
sqrt(b) #calculate the square root of all the elements of the vector
d<-matrix(data=seq(2,24,2),nrow=4,ncol=4) #create a 4x4 matrix
d
sqrt(d) #calculates 16 square roots from the matrix
```

quantile(d,probs=c(0.1,0.5,0.9)) # calculates the estimated 10%,50%, and 90% quantiles of d

Objects: Data are bundled into *objects*, and functions do something with the objects. Objects can have different *structures* (see above), and the output of a function depends on the structure of the object on which it acts. The description of function, above, illustrates this. The function sqrt (square root) returns a single number if a single number is included inside the parentheses; it returns a vector of six numbers if it acts on a vector of six numbers; it would return a matrix of 16 number, if given a 4x4 matrix to act on.

Working directory: The working directory is the default folder you want to use to read from and write to in a particular session. Specifying the path to the working directory allows you to use much shorter names to refer to specific files. You can set the working directory using the menu Misc:Change working directory (on PC, under File menu), or use the command line setwd("/Users/ggilbert/Documents/RPlotData") where the part inside the quotes shows the chain of names to the folder (in this case the folder RPlotData) (on a Windows machine this would be something like setwd("C:/Documents/RPlotData").

Libraries: Libraries are packages of functions designed to do specific things, written by various people in the R community. The base installation of R has a number of useful functions, but the real value of R is the development of different libraries. Install packages from the `Package & Data` menu. Choose `Package Installer`, type in the name of the package, and click `Get List`. Check `install dependencies`, and then click `Install Selected` to download the package. To use the package, you then need to load it into the session in which you use it. You can then use `Package & Data: Package Manager` to load the libraries you want. Or, use the command line `library(xxx)`, where xxx is the name of the library you wish to load.

Workspace: The objects and libraries you have open in a particular session. When closing an R session, if you choose to Save workspace image, they will all still be there the next time you open, ready to go. You can save and restore different workspace files (see workspace menu) for different projects.

(4) SOME BASIC R NOTATION

Anything following the "#" and before a carriage return is treated as a comment, and is not executed. It is very useful to annotate your code liberally so that other people (or you, a month from now) can easily see what the code does.

```
library(Rcmdr)     #Loads R commander
```

? ?x, where x is the function you want to learn about opens a new R Help window with detailed, standardized instructions on how to use that function. Also, help(x).

```
?plot  #shows the help page for the plot function
```

<- This is the most general assignment operator in R. It assigns the variable a to be equal to whatever is on the right side of the "<-" symbol.

```
a<-9   # sets a to be equal to 9
b<-a*3   #sets b to be equal to a times 3
```

(Note: this is equivalent to "=", but the "=" operator in R can only be used at the top level of the command line prompt or the top level within brackets)

( ) Round brackets. Functions act on whatever is inside the parentheses.

```
log10(100)  # returns the base-10 log of 100, that is, 2
```

c( ) Concatenate function. Creates a vector of multiple elements.

```
d<-c(1,2,4,6)  # sets d to be a 4-element vector with elements 1 2 4 6
```

[ ] Square brackets. Indicates the position inside a vector.

```
d<-c(1,2,4,6) # sets the vector d to have the elements 1, 2, 4, and 6
d   # returns the full vector 1 2 4 6
d[3]  # returns the third element of the vector, that is, 4
f[1:5,2:3]  #returns rows 1 to 5 of columns 2 and 3 of data frame f
f[1:10,] #returns rows 1 to 10, and all columns of data frame f
f[,c(2,5,6)] #returns all rows of columns 2, 5, and 6 of data frame f
```

{ } Curly brackets. Marks the start and stop of a loop or complex function

```
b=0 #sets b to 0
a<-c(2,4,6,8,10)   # creates vector a with 5 even elements
for (i in 1:5) {b<-b+a[i]}
#for a loop of 5 steps, add the first element of a to b,
#then the second element of a to b, etc.
b  # returns 30 (that is, starts with 0 then adds 2+4+6+8+10)
```

; functions as a carriage return, and indicates the end of a function. You can use it to string together several simple functions on the same line.

```
c<-3; d<-c*2; c; d;  #prints 3 on one line, and 6 on the next line.
```

$ The $ is used to indicate a column (variable) within a data frame.

```
MyDF$mort  #returns all rows of column mort from data frame MyDF
```

SOME USEFUL COMMON OPERATORS

Arithmetic operators

+	add
-	subtract
*	multiply
/	divide
^	to the power of e.g., x^3 is x cubed
%%	modulo, the remainder of division, e.g., 8%%3 is 2
%/%	integer division, e.g., 9%/%2 is 4

Compare operators

<	less than e.g., x <y ; x is less than y
>	greater than e.g., x >y ; x is greater than y
==	equal to e.g., x ==y ; x is equal to y
<=	less than or equal to e.g., x <=y ; x is less than or equal to y
>=	greater than or equal to e.g., x >=y ; x is less than or equal to y
!=	not equal to e.g., x!=y ; x is not equal to y
%in%	included in the set e.g., var %in% c(1,3,5)

Logic operators

&	and
|	or

Basic types

logical	TRUE or FALSE	a<-6; b<-a==6; b #returns TRUE
integer	whole numbers	a<-c(1,2,3,4,5)
double	real numbers	a<-c(3.4,-3.4,6.3,9)
numeric	type that includes both integer and double	
character	character strings	a<-c("ACERMA","PSEUME","SEQUSE")
factor	a category	R by default makes characters into factors

Math operators (most are what you expect; use ?abs to see details on abs, for example)

`"abs", "sign", "sqrt", "ceiling", "floor", "trunc", "cummax", "cummin", "cumprod", "cumsum", "log", "log10", "log2", "log1p", "acos", "acosh", "asin", "asinh", "atan", "atanh", "exp", "expm1", "cos", "cosh", "cospi", "sin", "sinh", "sinpi", "tan", "tanh", "tanpi", "gamma", "lgamma", "digamma", "trigamma", "round", "signif"`

LOOKING AT AN OBJECT (the object myData (could be vector, matrix, or data frame)

str(myData) #returns the structure of the object myData, including the names and types of all variables, the length of the variables, and a snapshot of the values.

myData #simply typing the name of the object will print out the full contents of the data frame myData

head(myData) #shows the first 6 lines of myData
head(myData,10) #shows the first 10 lines of myData
tail(myData,7) #shows the last 7 lines of myData

length(myData) #for vectors only. Returns the number of elements
dim(myData) #returns the number of rows and number of columns in a 2-dimensional object called myData. If you just want to capture the number of rows, it is dim(myData)[1]; The number of columns is dim(myData)[2].

myData[1:10,] # show the first 10 lines and all columns of myData,. Structure is dataframe[rows,columns]. Nothing after the comma means show all columns

myData[,2:3] #shows all rows of just columns 2 and 3 of myData. Structure is dataframe[rows,columns]. Nothing before the comma means show all the rows

myData[1:5,c(1,3)] #shows the first five rows of columns 1,3 in myData

myData[1:5,c('temp','precip')] #shows the first five rows of the columns named temp and precip, in a 2-dimensional myData.

MORE ON REFERENCING PARTS OF DATA FRAMES, VECTORS, AND MATRICES

Vectors

```
v1<-runif(50,0,1) #make vector of 50 uniform random numbers between 0 and 1
v1 # look at the vector
v1[1:10]    #shows 1st 10 elements
head(v1,7)   #shows the 1st 7 elements
tail(v1,5) # shows the last 5 elements
v1[-3]  #shows all except element 3
v1[-1:-5]  #shows all except the first 5 elements
```

Matrices

#Matrix notation takes the form MatrixName[rows,columns]

```
m1<-matrix(runif(25,0,1),nrow=5,ncol=5) #make a 5x5 matrix of random numbers
m1   #look at the matrix
m1[1:3,1:2]  #look at the first three rows and first two columns
m1[c(1,3,5),c(2,4)]  #look at the odd rows and even columns
m1[,1:3]   #shows all rows (nothing before comma) and cols 1 through 3
m1[1:2,]  #shows rows 1 and 2, and all columns (nothing after comma)
rownames(m1)<-c("a","b","c","d","e")  #give names to the rows
colnames(m1)<-c("a","b","c","d","e") #give names to the columns
m1  #show matrix with row and column names
m1[c("a","d","e"),c("b","c","d")]  #refer to elements by their row and
           column names
```

Data frames

#Read in a data frame from a web source

```
df1<-
read.table("http://people.ucsc.edu/~ggilbert/Rclass_docs/RegressionDataset.csv",sep=",", header=TRUE)
```

df1 #look at it
str(df1) #get an idea of the structure
head(df1) #look at the first 6 rows
df1[1:10,1:2] #look at first 10 rows and 2 columns
df1[1:4,] #look a the first 4 rows and all the columns
df1[1:8,c("temp","num_spp")] # first 8 rows of two specific columns, called by name
df1$temp[1:12] #first 12 values of the temperature column of data frame df1

Attach and detach

If you are going to focus analyses one data frame, you can avoid the $ structure (e.g., df1$temp), and refer just to the column name, by first calling to attach.

```
attach(df1)   # attach data frame df1 so all subsequent commands refer to it
precip   #shows the values of df1$precip, without including the df1$ part
detach(df1)  # detache df1 when you want to use other objects
```

#BE CAREFUL: if you have another variable somewhere called precip, this can be very messy!

(5) IMPORT YOUR DATA INTO R FROM A SPREADSHEET

Here are several ways to import data into R. We will focus on moving data from a spreadsheet format into an R data frame called "df1".
Note 1. Save your data as comma-delimited (.csv) or tab-delimited (.txt) file.
Note 2. You need to tell R whether your data file has headers or not.

```
#Use dialog box to read data from comma-delimited file with headers
df1<-read.table(file.choose(),sep=",", header=TRUE) #specify csv
df1<-read.csv(file.choose()) #alternate version

#Use dialog box to read data from tab-delimited file with headers
df1<-read.table(file.choose(),sep="\t", header=TRUE)

#Use dialog box to read data from a comma-delimited file
#without headers, and then add headers
df1<-read.table(file.choose(),sep=",", header=FALSE)
#look at data
df1
#add in header names after the fact
colnames(df1)<-c("temp","precip","num_spp")
df1 #look again

#Read data from the clipboard, with headers
#copy and paste this text into the R console, but do not hit return
#In Excel, select your data, including headers, and choose copy
#move back to R console, and hit return
df1<-read.table("X11_clipboard",sep="\t", header=TRUE)

#Read data from comma-delimited file with headers,
# with data posted on the web
df1<-
read.table("http://people.ucsc.edu/~ggilbert/Rclass_docs/RegressionDataset.c
sv",sep=",", header=TRUE)

#Read data from comma-delimited file with headers,
# with specific pathway on your computer
df1<-read.table("/Users/family/Documents/Greg
docs/Rclass_2011/RegressionDataset.csv",sep=",", header=TRUE)

#set working directory to shorten pathway, then read file
setwd("/Users/family/Documents/Greg docs/Rclass_2011/")
#Could instead use menu Misc: Change Working Directory to do this
#On PCs Change Working Directory is under the File menu
df1<-read.table("RegressionDataset.csv",sep=",", header=TRUE)
```

Some examples for beginning use of R

```
a<-9          #assigns the value 9 to  vector a
sqrt(a)       #returns the square root of a
[1] 3

root_a<-sqrt(a)     #assigns the square root of a to vector root_a.
root_a        #shows the value of root_a
[1] 3

x<-c(2,4,6,8,9,10)  #creates a vector x with 6 elements
y<-c(3,4,6,7,8,9)   #creates a vector y with 6 elements
x             #shows you the contents of x
 [1]  2  4  6  8  9 10

xsquared<-x^2   #create vector xsquared with square of each value in x
xsquared
[1]   4  16  36  64  81 100

mean(x)             #returns the mean of the values in vector x
[1] 6.5

length(x)           #returns the number of elements in x
[1] 6

z<-x-y              #creates vector z: the difference between x and y
z                   #show contents of z
[1] -1  0  0  1  1  1

str(x)              #shows the structure of x
num [1:6] 2 4 6 8 9 10    #numeric type, six elements, and shows the first 6 elements

plot(x,y)           #makes a scatterplot of y by x in the Quartz window

xy<-cbind(x,y)      #bind x and y together into an object xy
xy                  #show the contents of xy
      x y
[1,]  2 3
[2,]  4 4
[3,]  6 6
[4,]  8 7
[5,]  9 8
[6,] 10 9

dim(xy)             #get the dimensions of xy
[1] 6 2          #xy has 6 rows and 2 columns
```

```
xy[1:3,]            #show just the first three rows of xy
x y
[1,] 2 3
[2,] 4 4
[3,] 6 6

xy[,2]              #show all the rows, but just the second column of xy
[1] 3 4 6 7 8 9

xy[1:4,"x"]         #show rows 1 to 4, of column x
[1] 2 4 6 8

xy[3,2]             #show the third row of the second column of xy
y
6

txy<-t(xy)          #get the transpose of xy
sxy<-xy%*%txy       #matrix multiplication of xy and txy
eigen(sxy)          #get the eigen values and vectors of sxy

d<-as.data.frame(xy)  #make a data frame from xy
sex<-c("male","male","female","female","male","female") #character vector
d$sex<-sex  #add it as a column to the dataframe
d #show the whole dataframe  or use head(d) to see first 6 lines
   x y    sex
1  2 3   male
2  4 4   male
3  6 6 female
4  8 7 female
5  9 8   male
6 10 9 female

str(d)              #look at the structure of d
'data.frame': 6 obs. of  3 variables:
 $ x  : num  2 4 6 8 9 10
 $ y  : num  3 4 6 7 8 9
 $ sex: chr  "male" "male" "female" "female" ...

d[1:2,c("sex","x")]     #show the first two values of variables sex and x
  sex x
1 male 2
2 male 4

ls()       #use this to see what objects you have in your work space
[1] "a"    "d"   "root_a"   "sex"    "x"   "xsquared" "xy"    "y"   "z"
```