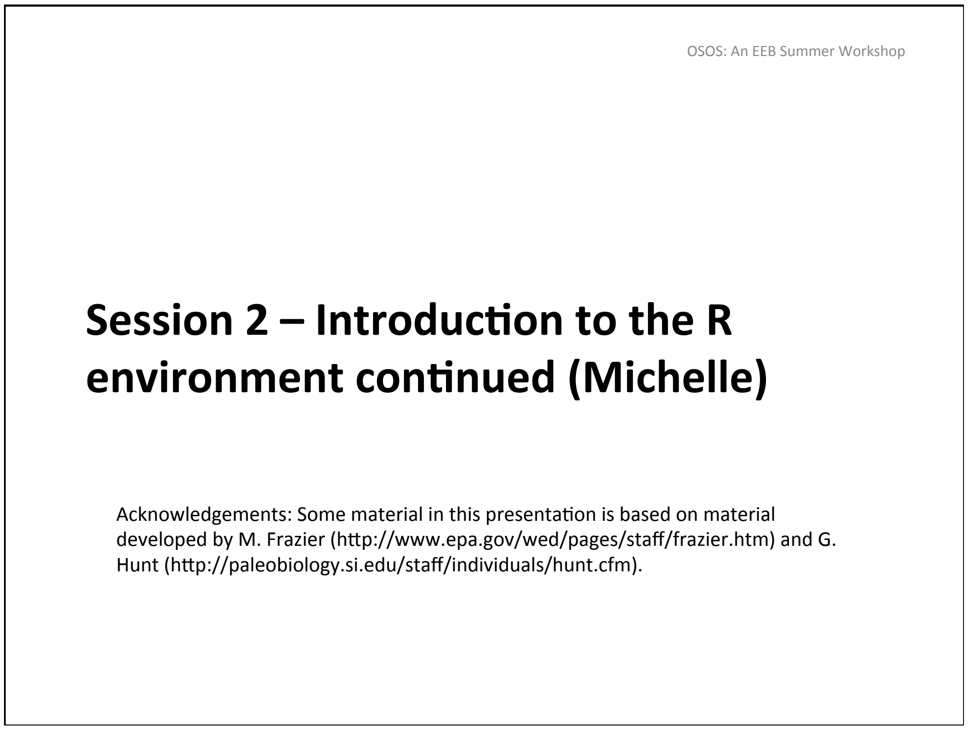
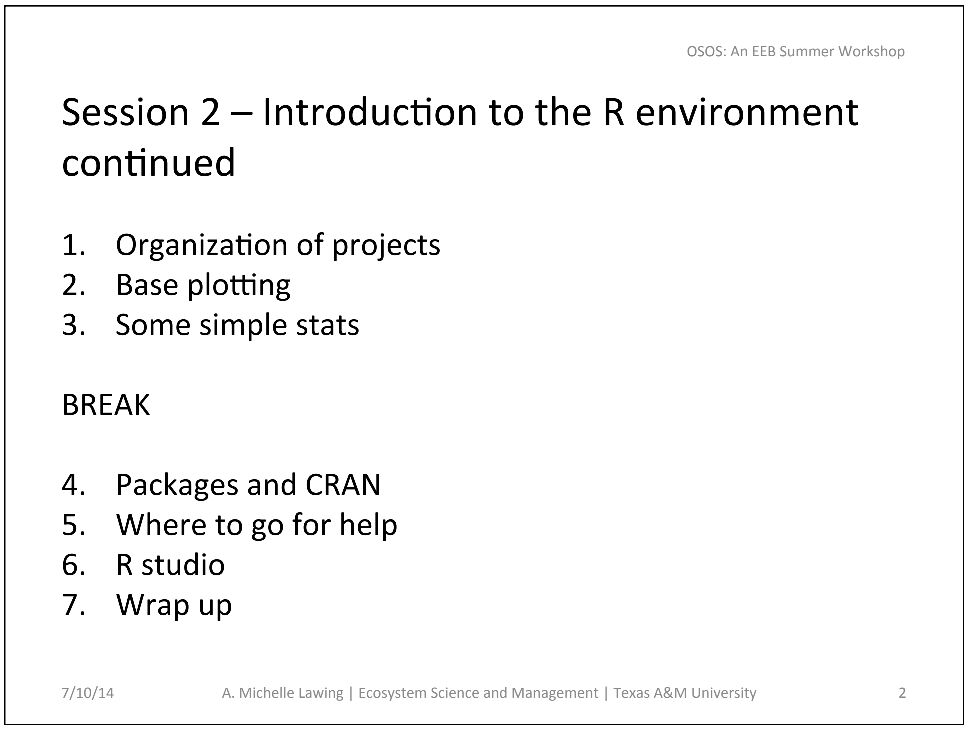


Session 2 – Introduction to the R environment continued (Michelle)

Acknowledgements: Some material in this presentation is based on material developed by M. Frazier (http://www.epa.gov/wed/pages/staff/frazier.htm) and G. Hunt (http://paleobiology.si.edu/staff/individuals/hunt.cfm).



Session 2 – Introduction to the R environment continued

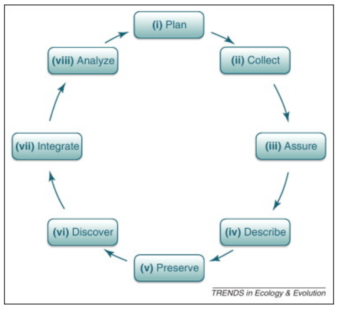
1. Organization of projects
2. Base plotting
3. Some simple stats

BREAK

4. Packages and CRAN
5. Where to go for help
6. R studio
7. Wrap up

Project Organization

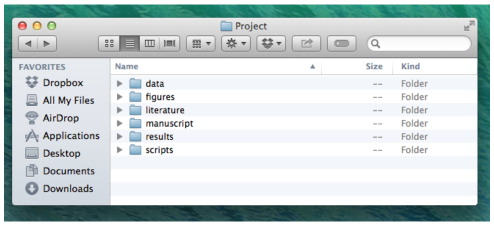
- Supports the data life cycle
- Reproducibility is important
- Adopt a good system for organization
- Facilitate future work
  - Easy for you to pick it up again (saves time!)
  - Easy to collaborate
  - Reproducible science



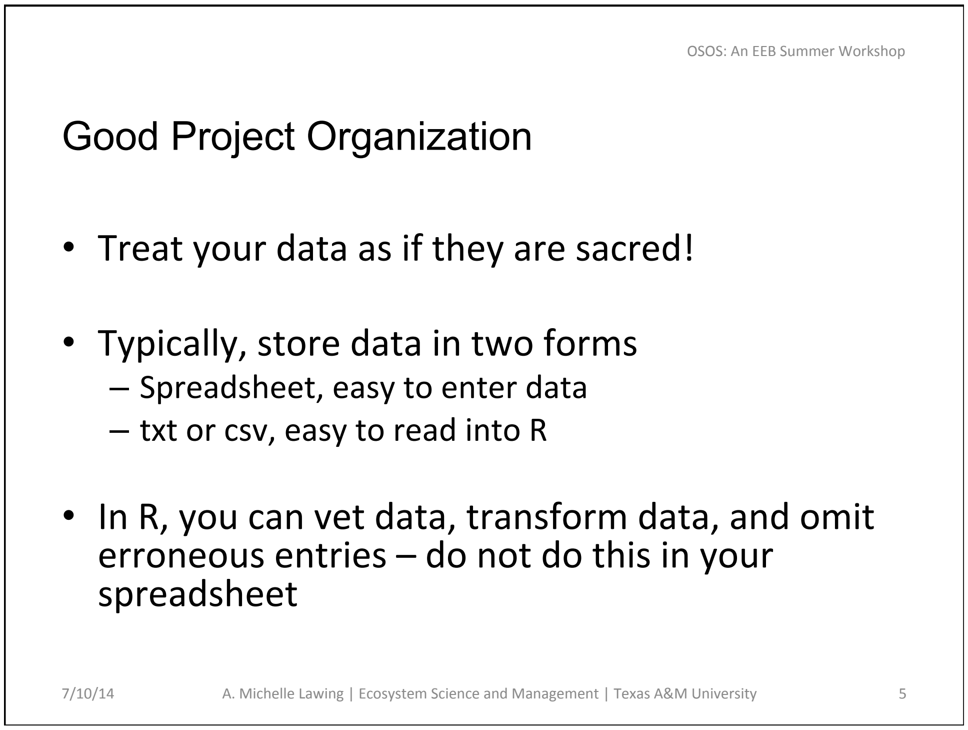
Michener and Jones 2012

Good Project Organization

- Keep everything in one file!
  - Every project has its own file
  - Decide on a file hierarchy

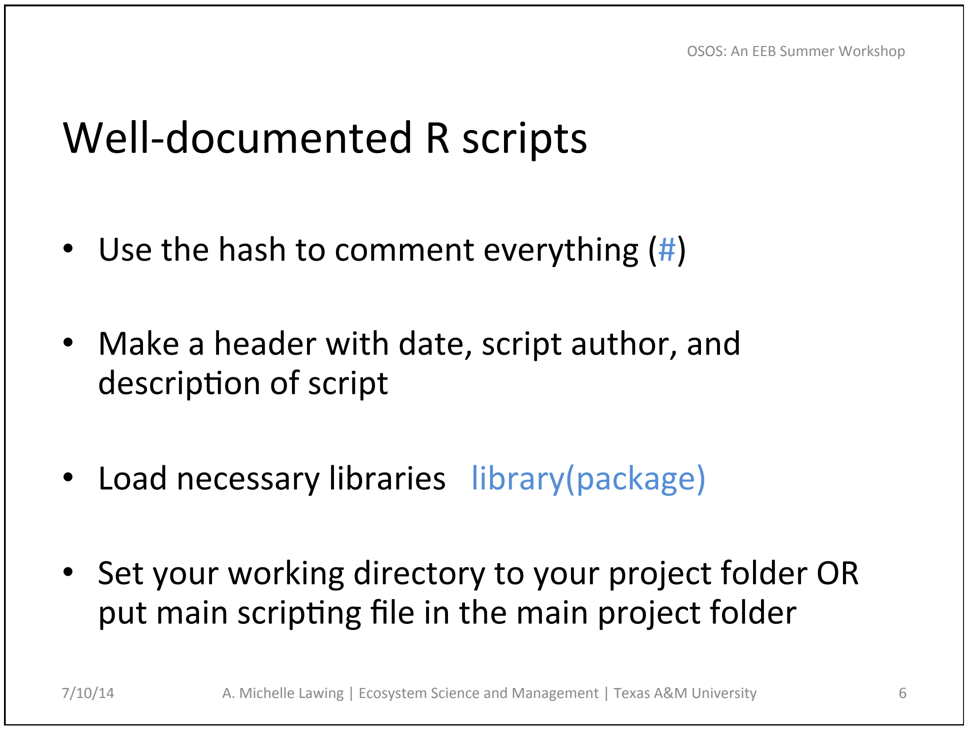


Name	Size	Kind
data	--	Folder
figures	--	Folder
literature	--	Folder
manuscript	--	Folder
results	--	Folder
scripts	--	Folder



Good Project Organization

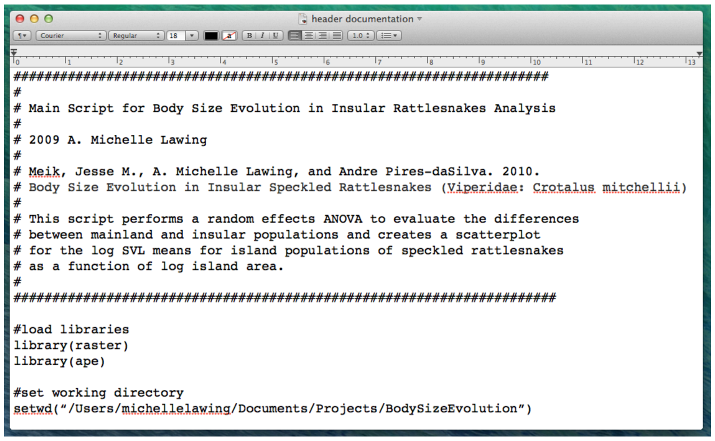
- Treat your data as if they are sacred!
- Typically, store data in two forms
  - Spreadsheet, easy to enter data
  - txt or csv, easy to read into R
- In R, you can vet data, transform data, and omit erroneous entries – do not do this in your spreadsheet



Well-documented R scripts

- Use the hash to comment everything (#)
- Make a header with date, script author, and description of script
- Load necessary libraries `library(package)`
- Set your working directory to your project folder OR put main scripting file in the main project folder

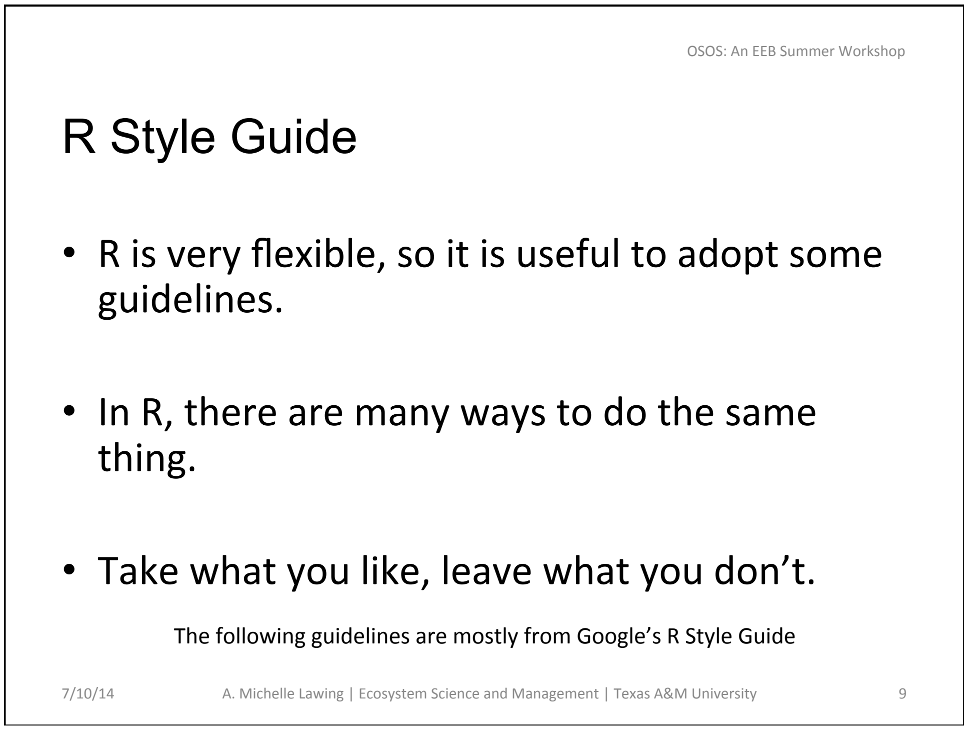
Header Documentation



```
########################################################################
#
# Main Script for Body Size Evolution in Insular Rattlesnakes Analysis
#
# 2009 A. Michelle Lawing
#
# Meik, Jesse M., A. Michelle Lawing, and Andre Pires-daSilva. 2010.
# Body Size Evolution in Insular Speckled Rattlesnakes (Viperidae: Crotalus mitchellii)
#
# This script performs a random effects ANOVA to evaluate the differences
# between mainland and insular populations and creates a scatterplot
# for the log SVL means for island populations of speckled rattlesnakes
# as a function of log island area.
#
########################################################################

#load libraries
library(raster)
library(ape)

#set working directory
setwd("/Users/michellelawing/Documents/Projects/BodySizeEvolution")
```

Well-documented R scripts

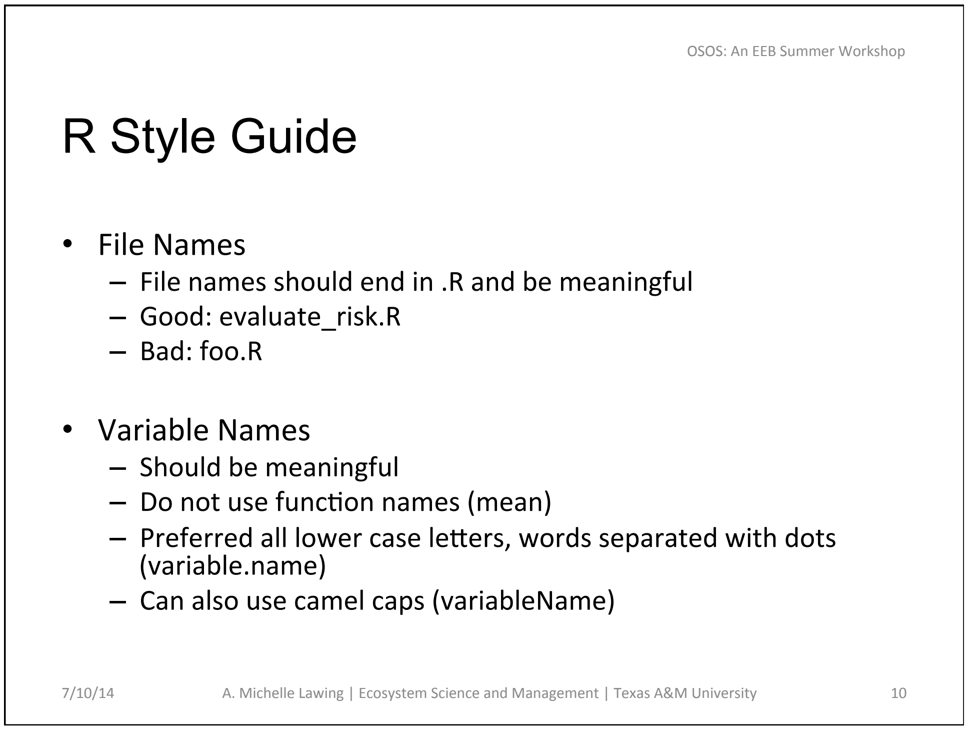
- Decide how many functions you put outside your main script file (for legibility)
- Read in project specific functions/scripts
- Load data, analyze data, make figures



R Style Guide

- R is very flexible, so it is useful to adopt some guidelines.
- In R, there are many ways to do the same thing.
- Take what you like, leave what you don't.

The following guidelines are mostly from Google's R Style Guide



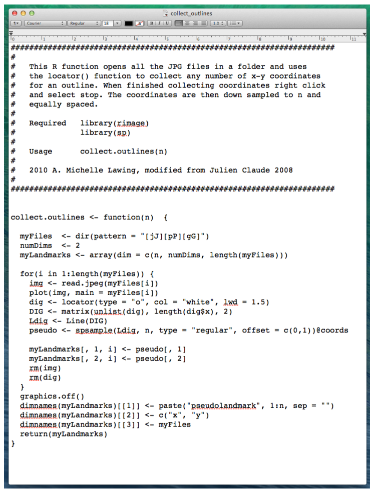
R Style Guide

- File Names
  - File names should end in .R and be meaningful
  - Good: evaluate_risk.R
  - Bad: foo.R
- Variable Names
  - Should be meaningful
  - Do not use function names (mean)
  - Preferred all lower case letters, words separated with dots (variable.name)
  - Can also use camel caps (variableName)

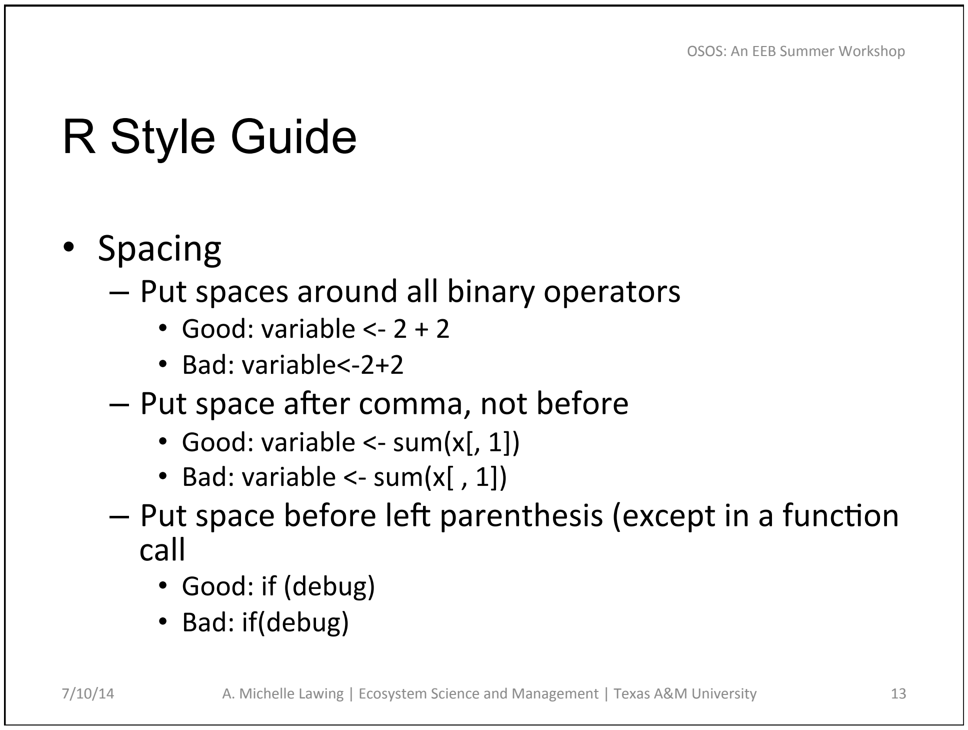
R Style Guide

- Function Names
  - Use action verbs
  - Capitalize first letter for each word
  - Good: GetWeights
  - Bad: getweights
- Indentation
  - For each indent, use two spaces
  - Do not use tab

Function Documentation

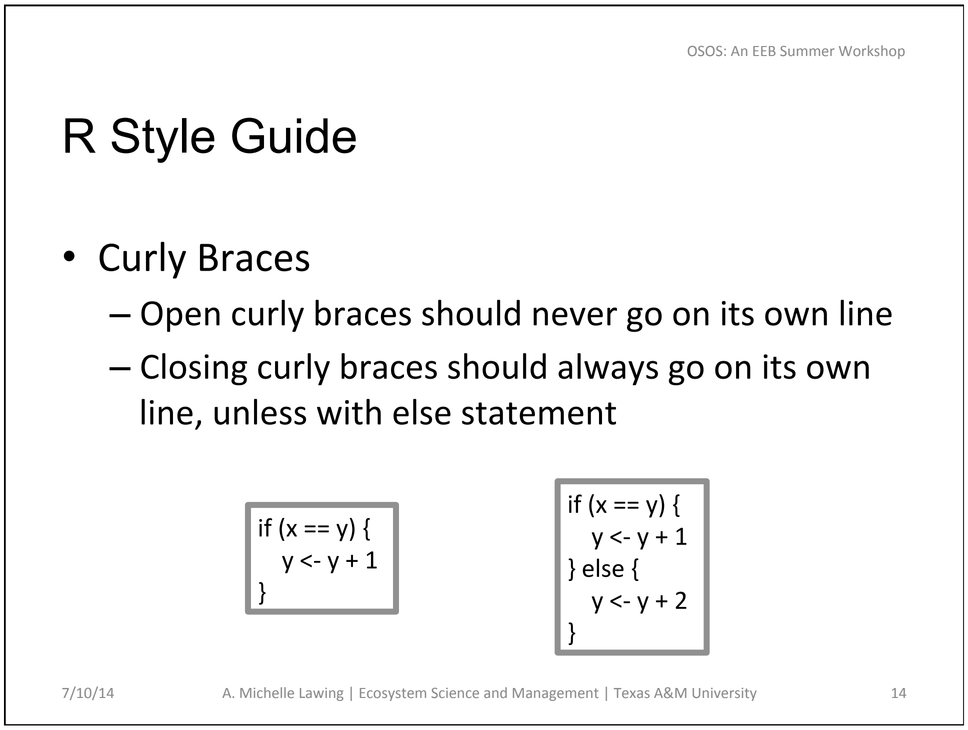


```
######################################################################
#
#   This R function opens all the JPG files in a folder and uses
#   the locator() function to collect any number of x-y coordinates
#   for an outline. When finished collecting coordinates right click
#   and select stop. The coordinates are then down sampled to n and
#   equally spaced.
#
#   Required   library(rimage)
#              library(sp)
#
#   Usage      collect.outlines(n)
#
#   2010 A. Michelle Lawing, modified from Julien Claude 2008
#
######################################################################


collect.outlines <- function(n)  {

  myFiles  <- dir(pattern = "[jJ][pP][gG]")
  numDims  <- 2
  myLandmarks <- array(dim = c(n, numDims, length(myFiles)))

  for(i in 1:length(myFiles)) {
    img <- read.jpeg(myFiles[i])
    plot(img, main = myFiles[i])
    dig <- locator(type = "o", col = "white", lwd = 1.5)
    DIG <- matrix(unlist(dig), length(dig$x), 2)
    Ldig <- Line(DIG)
    pseudo <- spsample(Ldig, n, type = "regular", offset = c(0,1))@coords

    myLandmarks[, 1, i] <- pseudo[, 1]
    myLandmarks[, 2, i] <- pseudo[, 2]
    rm(img)
    rm(dig)
  }
  graphics.off()
  dimnames(myLandmarks)[[1]] <- paste("pseudolandmark", 1:n, sep = "")
  dimnames(myLandmarks)[[2]] <- c("x", "y")
  dimnames(myLandmarks)[[3]] <- myFiles
  return(myLandmarks)
}
```



R Style Guide

- Spacing
  - Put spaces around all binary operators
    - Good: variable <- 2 + 2
    - Bad: variable<-2+2
  - Put space after comma, not before
    - Good: variable <- sum(x[, 1])
    - Bad: variable <- sum(x[, 1])
  - Put space before left parenthesis (except in a function call
    - Good: if (debug)
    - Bad: if(debug)

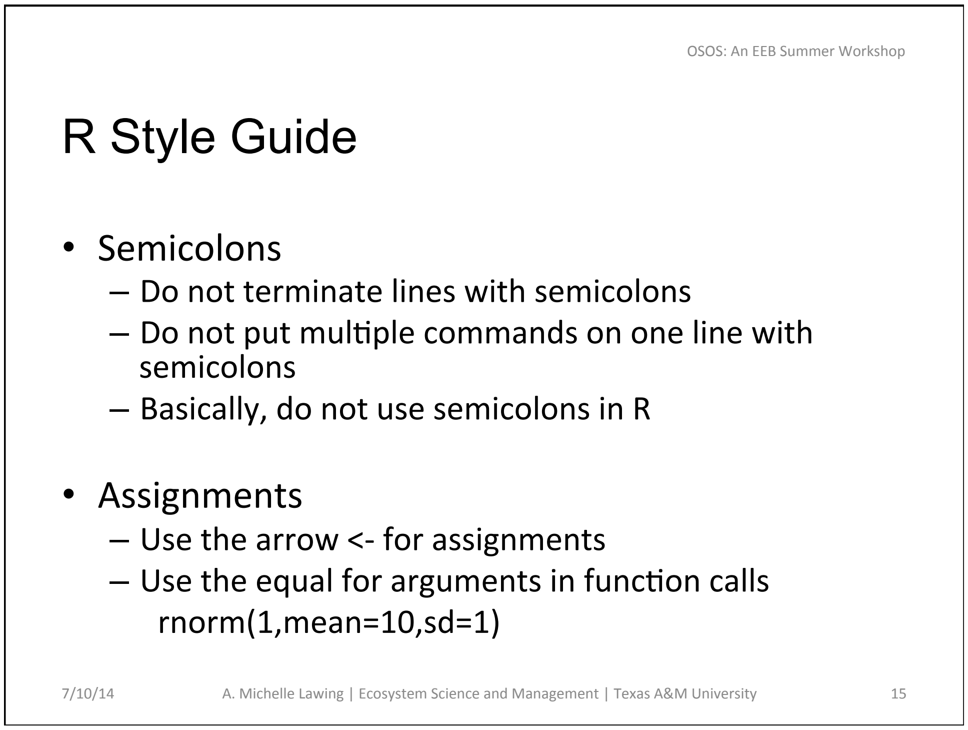


R Style Guide

- Curly Braces
  - Open curly braces should never go on its own line
  - Closing curly braces should always go on its own line, unless with else statement

```
if (x == y) {
  y <- y + 1
}
```

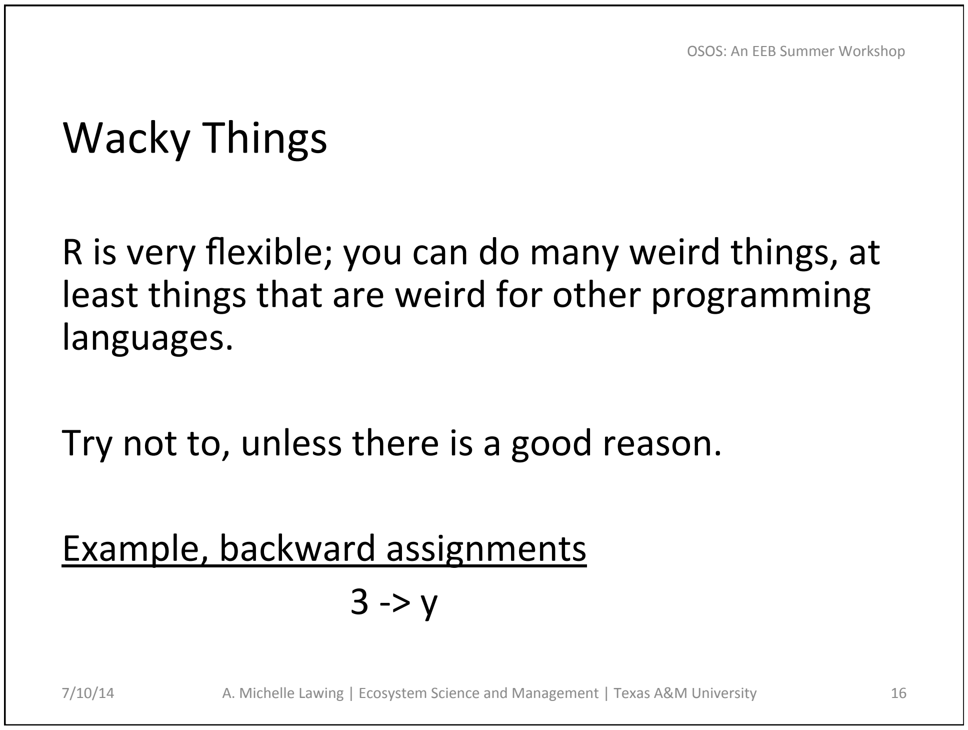
```
if (x == y) {
  y <- y + 1
} else {
  y <- y + 2
}
```



R Style Guide

- Semicolons
  - Do not terminate lines with semicolons
  - Do not put multiple commands on one line with semicolons
  - Basically, do not use semicolons in R
- Assignments
  - Use the arrow <- for assignments
  - Use the equal for arguments in function calls

    rnorm(1,mean=10,sd=1)



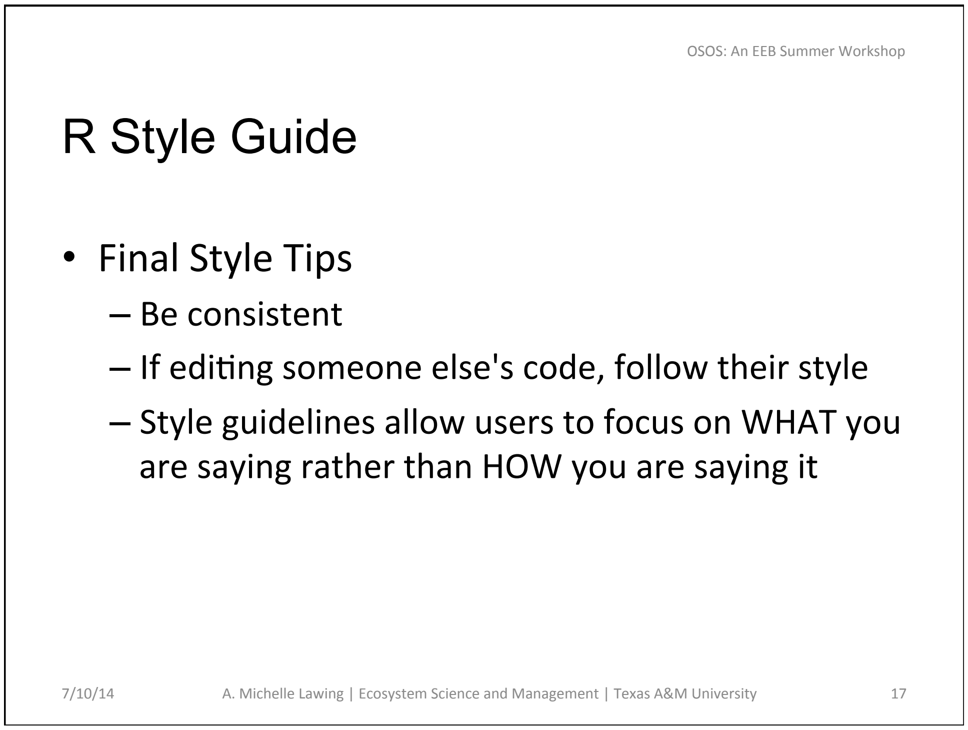
Wacky Things

R is very flexible; you can do many weird things, at least things that are weird for other programming languages.

Try not to, unless there is a good reason.

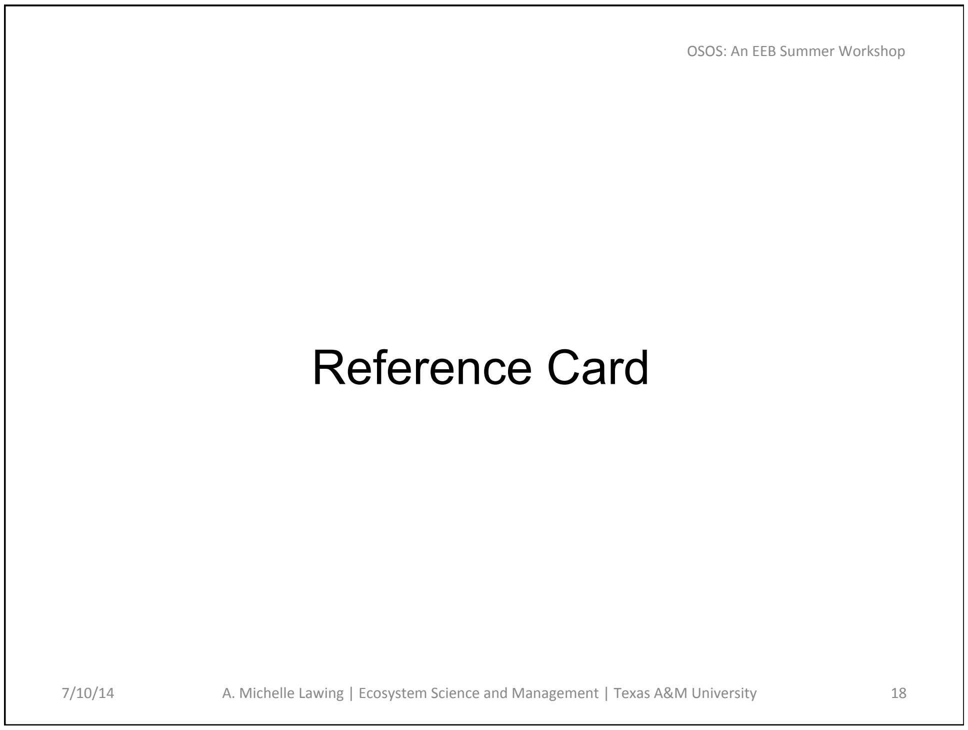
<u>Example, backward assignments</u>

3 -> y

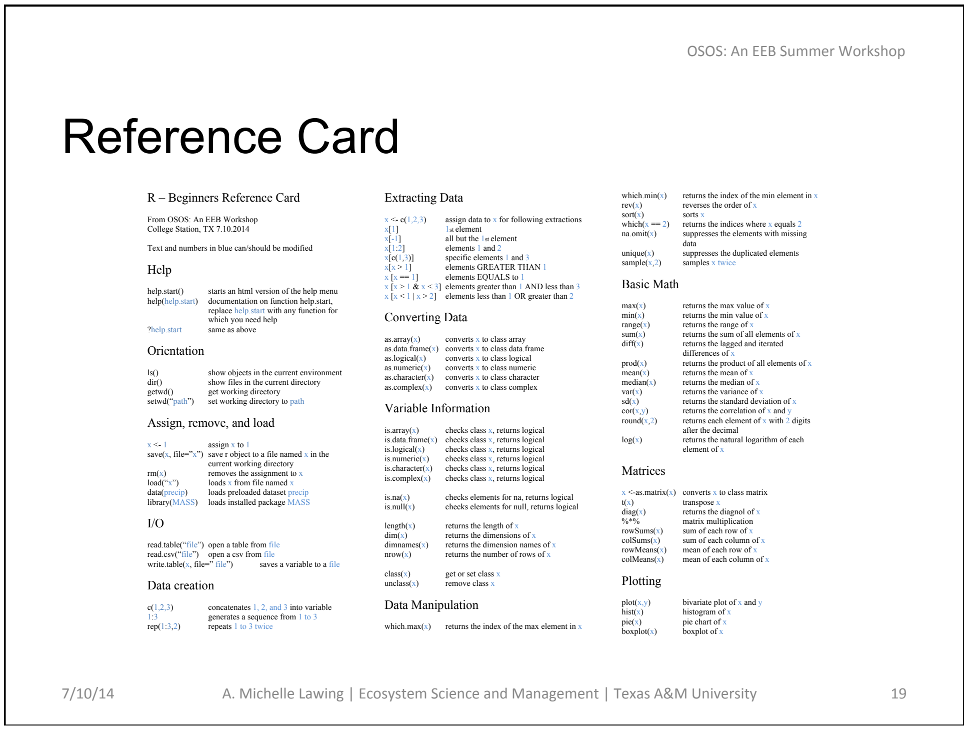


R Style Guide

- Final Style Tips
  - Be consistent
  - If editing someone else's code, follow their style
  - Style guidelines allow users to focus on WHAT you are saying rather than HOW you are saying it



Reference Card



Reference Card

R – Beginners Reference Card

From OSOS: An EEB Workshop
College Station, TX 7.10.2014

Text and numbers in blue can/should be modified

Help

help.start()	starts an html version of the help menu
help(help.start)	documentation on function help.start, replace help.start with any function for which you need help
?help.start	same as above

Orientation

ls()	show objects in the current environment
dir()	show files in the current directory
getwd()	get working directory
setwd("path")	set working directory to path

Assign, remove, and load

x <- 1	assign x to 1
save(x, file="x")	save r object to a file named x in the current working directory
rm(x)	removes the assignment to x
load("x")	loads x from file named x
data(precip)	loads preloaded dataset precip
library(MASS)	loads installed package MASS

I/O

read.table("file")	open a table from file
read.csv("file")	open a csv from file
write.table(x, file=" file")	saves a variable to a file

Data creation

c(1,2,3)	concatenates 1, 2, and 3 into variable
1:3	generates a sequence from 1 to 3
rep(1:3,2)	repeats 1 to 3 twice

Extracting Data

x <- c(1,2,3)	assign data to x for following extractions
x[1]	1st element
x[-1]	all but the 1st element
x[1:2]	elements 1 and 2
x[c(1,3)]	specific elements 1 and 3
x[x > 1]	elements GREATER THAN 1
x [x == 1]	elements EQUALS to 1
x [x > 1 & x < 3]	elements greater than 1 AND less than 3
x [x < 1 | x > 2]	elements less than 1 OR greater than 2

Converting Data

as.array(x)	converts x to class array
as.data.frame(x)	converts x to class data.frame
as.logical(x)	converts x to class logical
as.numeric(x)	converts x to class numeric
as.character(x)	converts x to class character
as.complex(x)	converts x to class complex

Variable Information

is.array(x)	checks class x, returns logical
is.data.frame(x)	checks class x, returns logical
is.logical(x)	checks class x, returns logical
is.numeric(x)	checks class x, returns logical
is.character(x)	checks class x, returns logical
is.complex(x)	checks class x, returns logical
is.na(x)	checks elements for na, returns logical
is.null(x)	checks elements for null, returns logical
length(x)	returns the length of x
dim(x)	returns the dimensions of x
dimnames(x)	returns the dimension names of x
nrow(x)	returns the number of rows of x
class(x)	get or set class x
unclass(x)	remove class x

Data Manipulation

which.max(x)	returns the index of the max element in x
which.min(x)	returns the index of the min element in x
rev(x)	reverses the order of x
sort(x)	sorts x
which(x == 2)	returns the indices where x equals 2
na.omit(x)	suppresses the elements with missing data
unique(x)	suppresses the duplicated elements
sample(x,2)	samples x twice

Basic Math

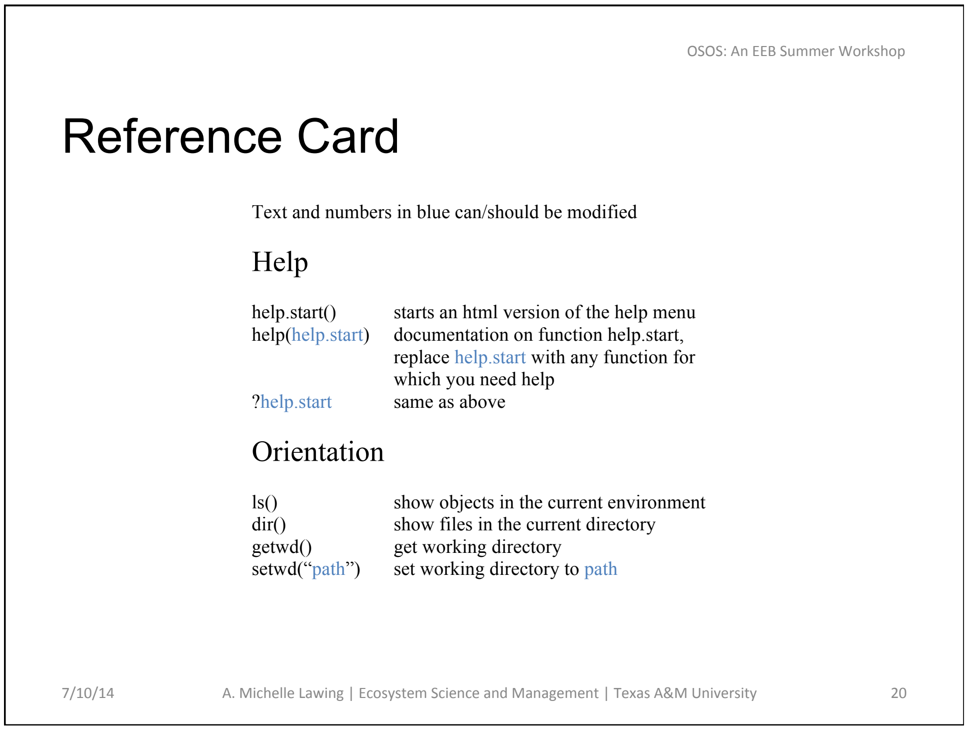
max(x)	returns the max value of x
min(x)	returns the min value of x
range(x)	returns the range of x
sum(x)	returns the sum of all elements of x
diff(x)	returns the lagged and iterated differences of x
prod(x)	returns the product of all elements of x
mean(x)	returns the mean of x
median(x)	returns the median of x
var(x)	returns the variance of x
sd(x)	returns the standard deviation of x
cor(x,y)	returns the correlation of x and y
round(x,2)	returns each element of x with 2 digits after the decimal
log(x)	returns the natural logarithm of each element of x

Matrices

x <-as.matrix(x)	converts x to class matrix
t(x)	transpose x
diag(x)	returns the diagnol of x
%*%	matrix multiplication
rowSums(x)	sum of each row of x
colSums(x)	sum of each column of x
rowMeans(x)	mean of each row of x
colMeans(x)	mean of each column of x

Plotting

plot(x,y)	bivariate plot of x and y
hist(x)	histogram of x
pie(x)	pie chart of x
boxplot(x)	boxplot of x



Reference Card

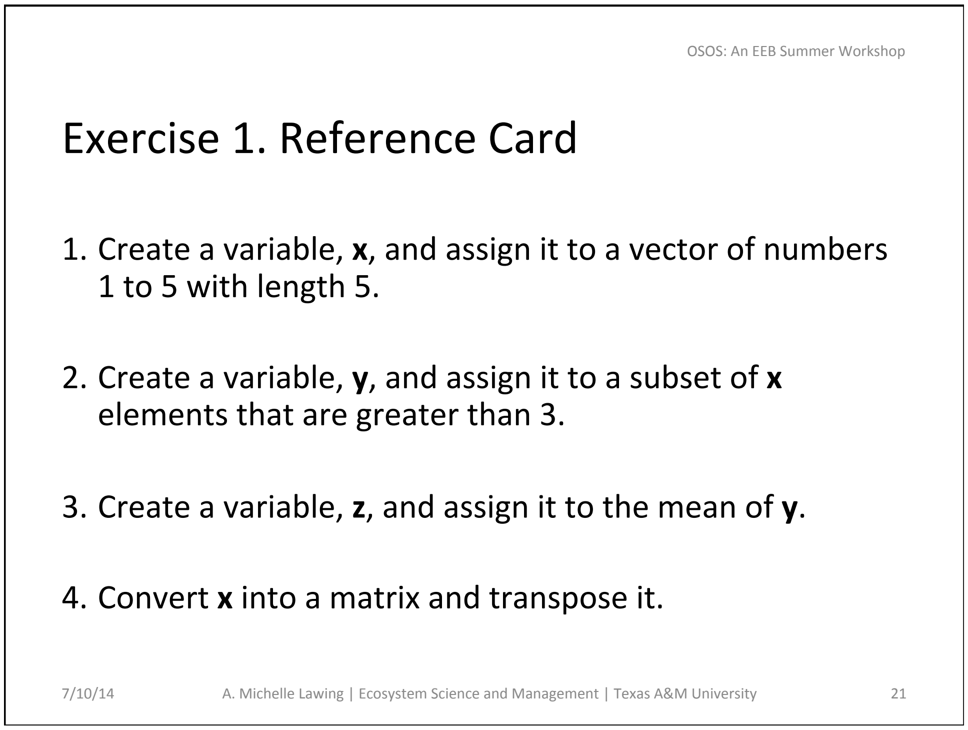
Text and numbers in blue can/should be modified

Help

help.start()	starts an html version of the help menu
help(help.start)	documentation on function help.start, replace help.start with any function for which you need help
?help.start	same as above

Orientation

ls()	show objects in the current environment
dir()	show files in the current directory
getwd()	get working directory
setwd("path")	set working directory to path



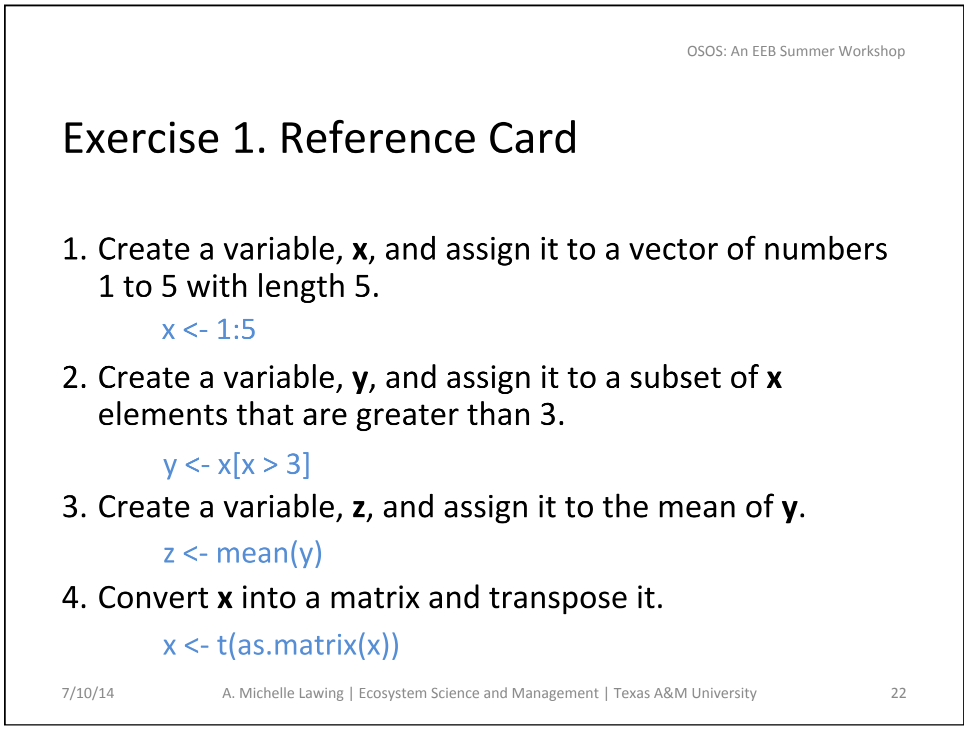
Exercise 1. Reference Card

1. Create a variable, **x**, and assign it to a vector of numbers 1 to 5 with length 5.

2. Create a variable, **y**, and assign it to a subset of **x** elements that are greater than 3.

3. Create a variable, **z**, and assign it to the mean of **y**.

4. Convert **x** into a matrix and transpose it.



Exercise 1. Reference Card

1. Create a variable, **x**, and assign it to a vector of numbers 1 to 5 with length 5.
```
x <- 1:5
```
2. Create a variable, **y**, and assign it to a subset of **x** elements that are greater than 3.
```
y <- x[x > 3]
```
3. Create a variable, **z**, and assign it to the mean of **y**.
```
z <- mean(y)
```
4. Convert **x** into a matrix and transpose it.
```
x <- t(as.matrix(x))
```

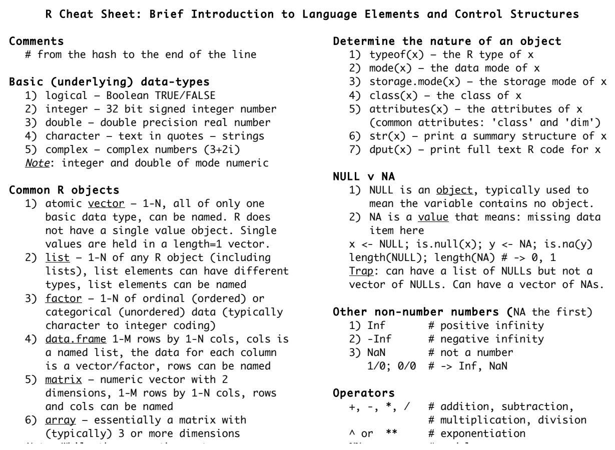
References for Reference Cards

The files below are available from CRAN Contributed Documentation:

- R Reference Card, by Jonathan Baron
- R Reference Card, by Tom Short
- Time Series Reference Card, by Vito Ricci
- Regression Reference Card, by Vito Ricci
- R Language Reference Card
- R Metrics Reference Card
- VGAM Reference Card
- R Reference Card for Data Mining, by Yanchang Zhao
- R Reference Card on memrise, by Daniele Amberti

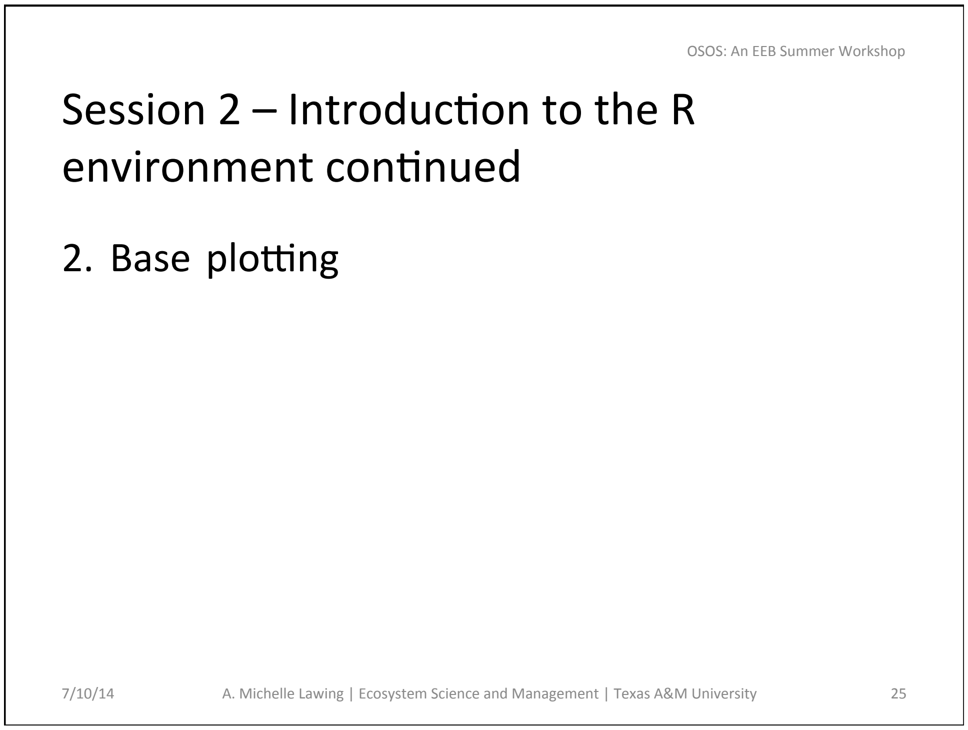
Google is my reference card!

R Cheat Sheets



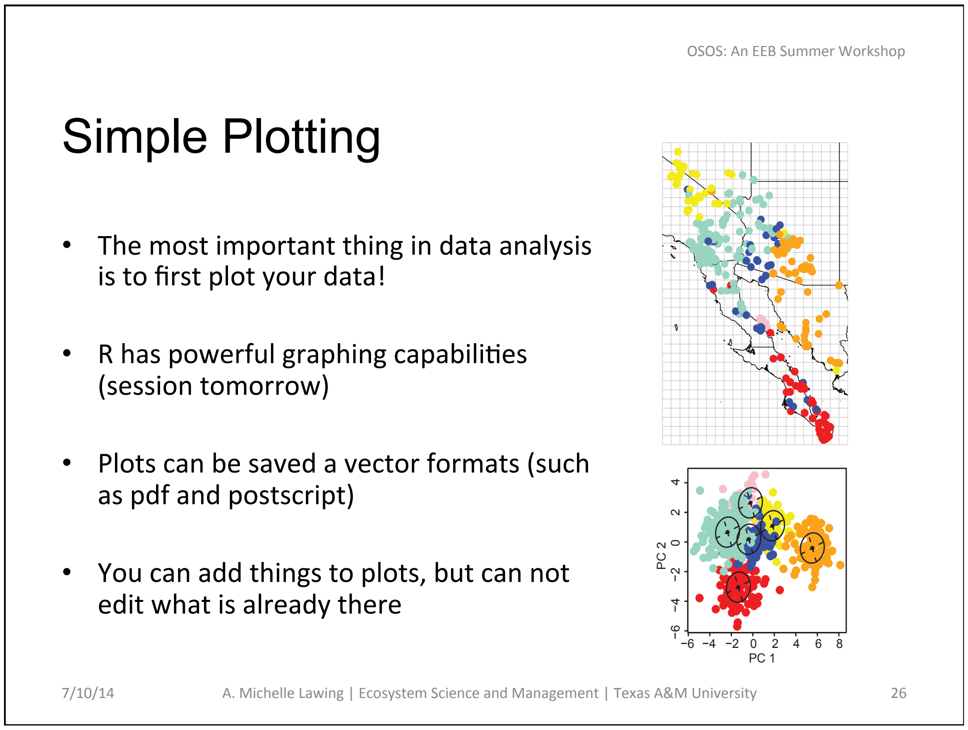
```
R Cheat Sheet: Brief Introduction to Language Elements and Control Structures

Comments
  # from the hash to the end of the line

Basic (underlying) data-types
  1) logical – Boolean TRUE/FALSE
  2) integer – 32 bit signed integer number
  3) double – double precision real number
  4) character – text in quotes – strings
  5) complex – complex numbers (3+2i)
  Note: integer and double of mode numeric

Common R objects
  1) atomic vector – 1-N, all of only one
     basic data type, can be named. R does
     not have a single value object. Single
     values are held in a length=1 vector.
  2) list – 1-N of any R object (including
     lists), list elements can have different
     types, list elements can be named
  3) factor – 1-N of ordinal (ordered) or
     categorical (unordered) data (typically
     character to integer coding)
  4) data.frame 1-M rows by 1-N cols, cols is
     a named list, the data for each column
     is a vector/factor, rows can be named
  5) matrix – numeric vector with 2
     dimensions, 1-M rows by 1-N cols, rows
     and cols can be named
  6) array – essentially a matrix with
     (typically) 3 or more dimensions

Determine the nature of an object
  1) typeof(x) – the R type of x
  2) mode(x) – the data mode of x
  3) storage.mode(x) – the storage mode of x
  4) class(x) – the class of x
  5) attributes(x) – the attributes of x
     (common attributes: 'class' and 'dim')
  6) str(x) – print a summary structure of x
  7) dput(x) – print full text R code for x

NULL v NA
  1) NULL is an object, typically used to
     mean the variable contains no object.
  2) NA is a value that means: missing data
     item here
  x <- NULL; is.null(x); y <- NA; is.na(y)
  length(NULL); length(NA) # -> 0, 1
  Trap: can have a list of NULLs but not a
  vector of NULLs. Can have a vector of NAs.

Other non-number numbers (NA the first)
  1) Inf        # positive infinity
  2) -Inf       # negative infinity
  3) NaN        # not a number
     1/0; 0/0   # -> Inf, NaN

Operators
  +, -, *, /    # addition, subtraction,
                # multiplication, division
  ^ or  **      # exponentiation
```

13 different cheat sheets → markthegraph.blogspot.com.au



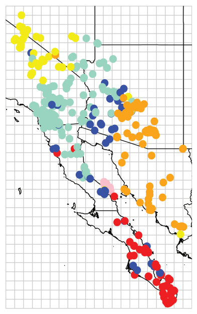
Session 2 – Introduction to the R environment continued

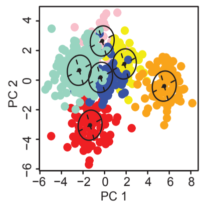
2. Base plotting



Simple Plotting

- The most important thing in data analysis is to first plot your data!
- R has powerful graphing capabilities (session tomorrow)
- Plots can be saved a vector formats (such as pdf and postscript)
- You can add things to plots, but can not edit what is already there



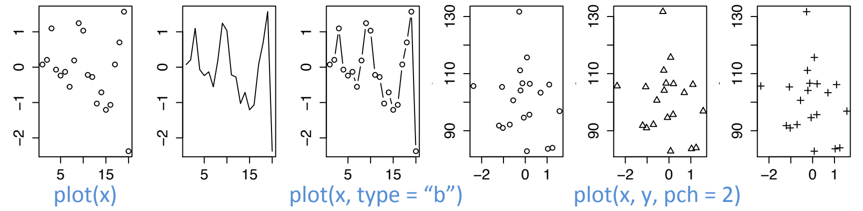


Exercise 2. Make some plots

1. Generate some data

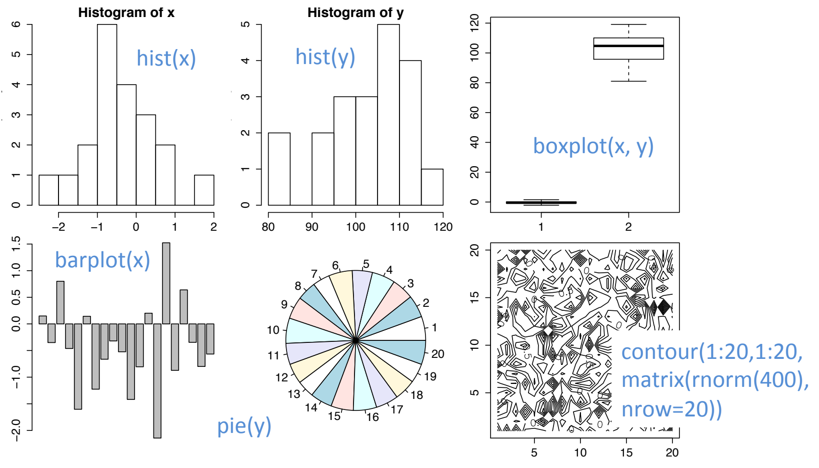
```
x <- rnorm(n=20,mean=0,sd=1)        #random normal numbers
                                    #same as rnorm(20,0,1)
                                    #same as rnorm(20)

y <- rnorm(n=20,mean=100,sd=10)
```

2. Plot it out

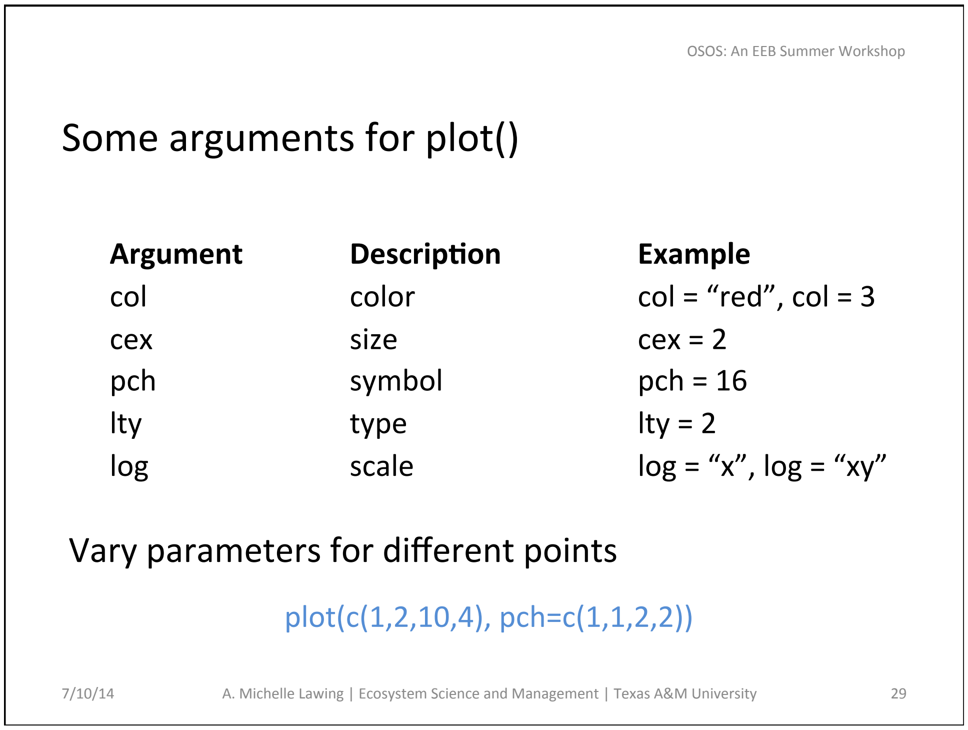


```
plot(x)
plot(x, type = "l")
plot(x, type = "b")
plot(x, y)
plot(x, y, pch = 2)
plot(x, y, pch = 3)
```

Other Simple Plotting Functions



```
hist(x)
hist(y)
boxplot(x, y)
barplot(x)
pie(y)
contour(1:20,1:20,
matrix(rnorm(400),
nrow=20))
```

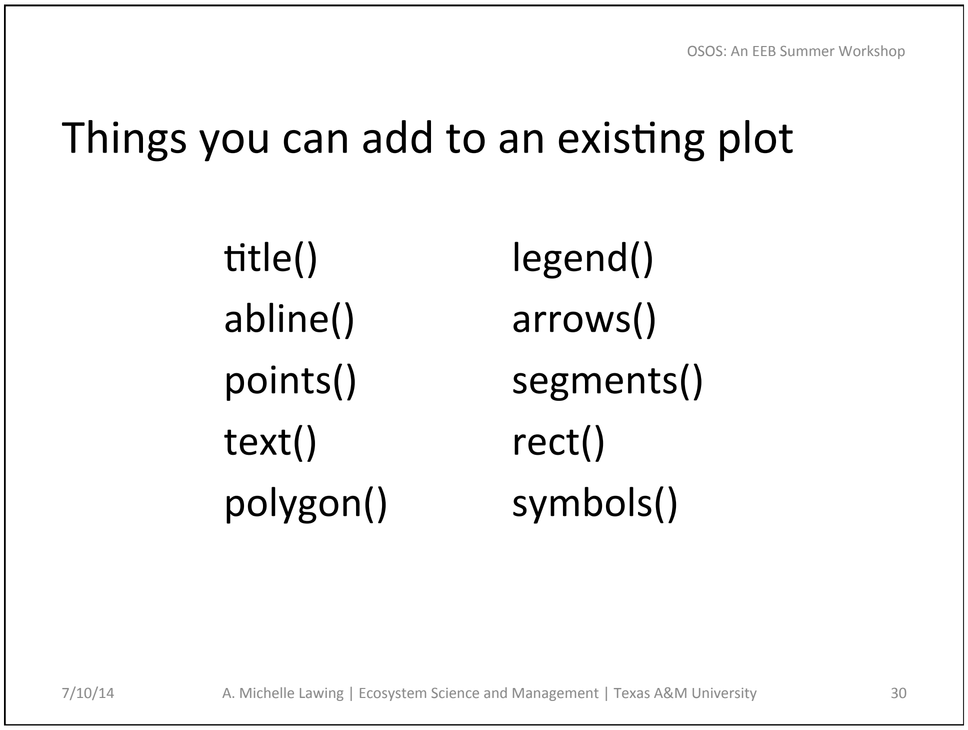


Some arguments for plot()

Argument	**Description**	**Example**
col	color	col = “red”, col = 3
cex	size	cex = 2
pch	symbol	pch = 16
lty	type	lty = 2
log	scale	log = “x”, log = “xy”

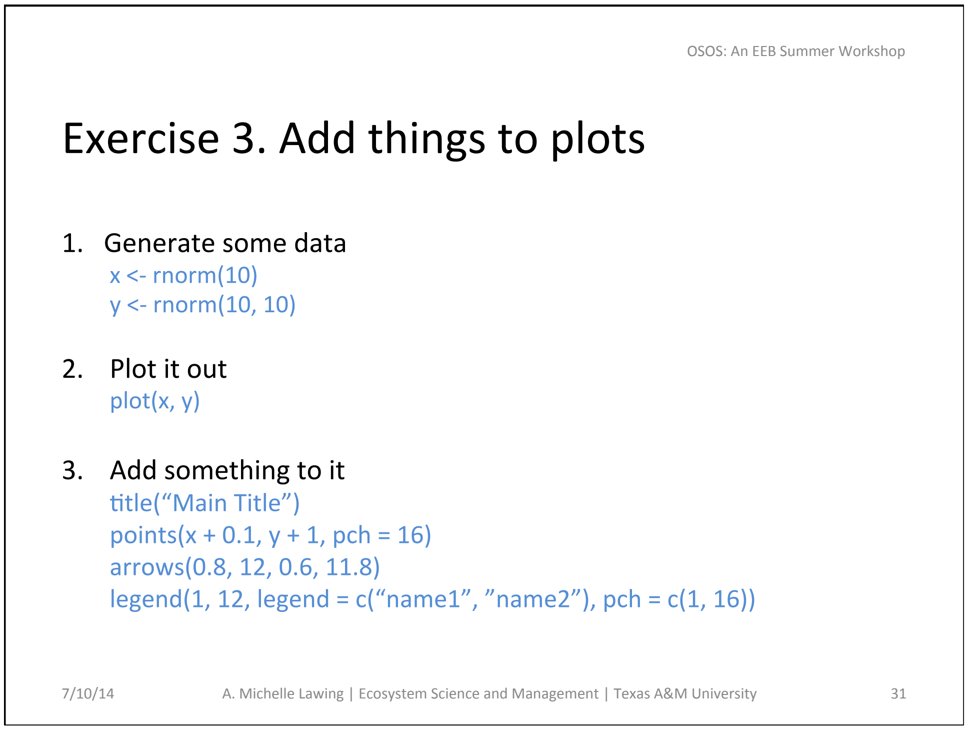
Vary parameters for different points

```
plot(c(1,2,10,4), pch=c(1,1,2,2))
```



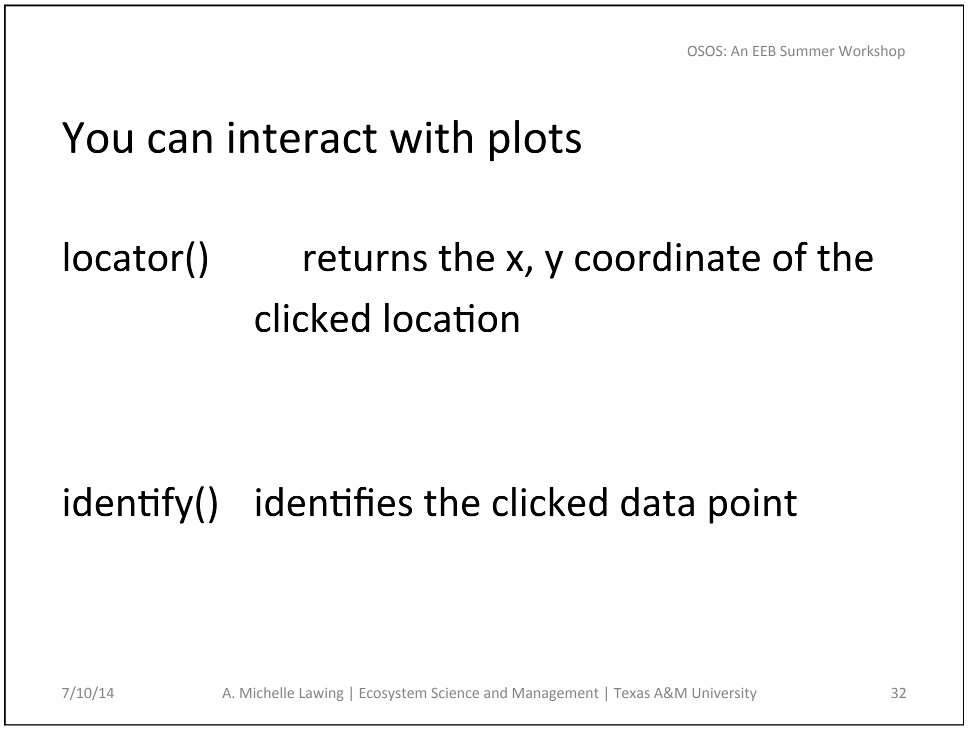
Things you can add to an existing plot

title()	legend()
abline()	arrows()
points()	segments()
text()	rect()
polygon()	symbols()



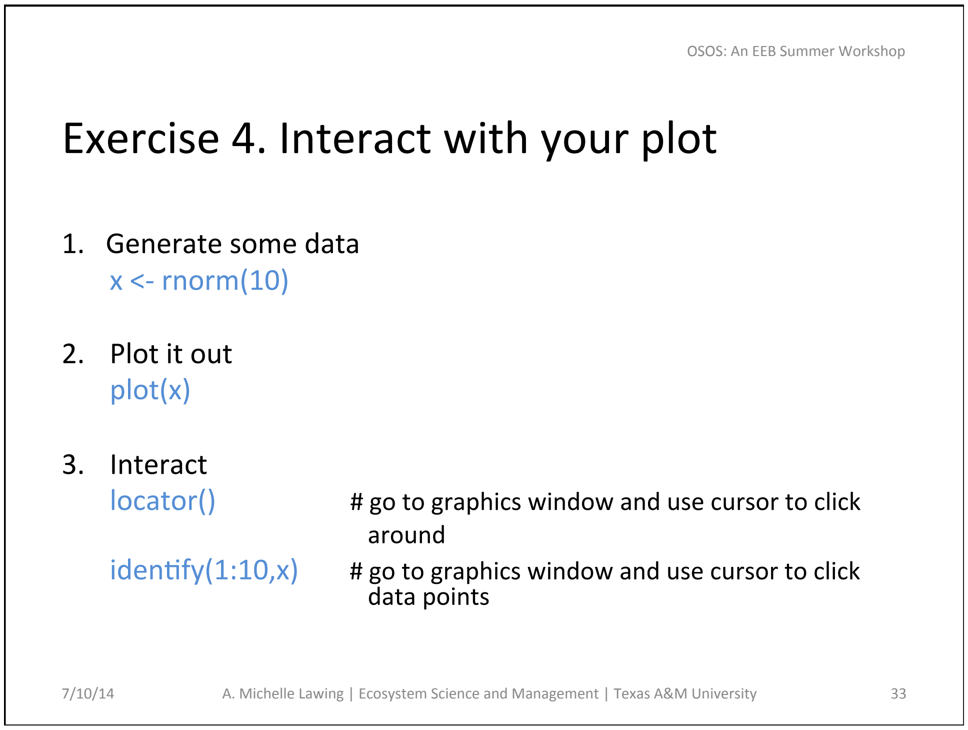
Exercise 3. Add things to plots

1. Generate some data
```
x <- rnorm(10)
y <- rnorm(10, 10)
```
2. Plot it out
```
plot(x, y)
```
3. Add something to it
```
title("Main Title")
points(x + 0.1, y + 1, pch = 16)
arrows(0.8, 12, 0.6, 11.8)
legend(1, 12, legend = c("name1", "name2"), pch = c(1, 16))
```



You can interact with plots

locator()	returns the x, y coordinate of the clicked location
identify()	identifies the clicked data point



Exercise 4. Interact with your plot

1. Generate some data

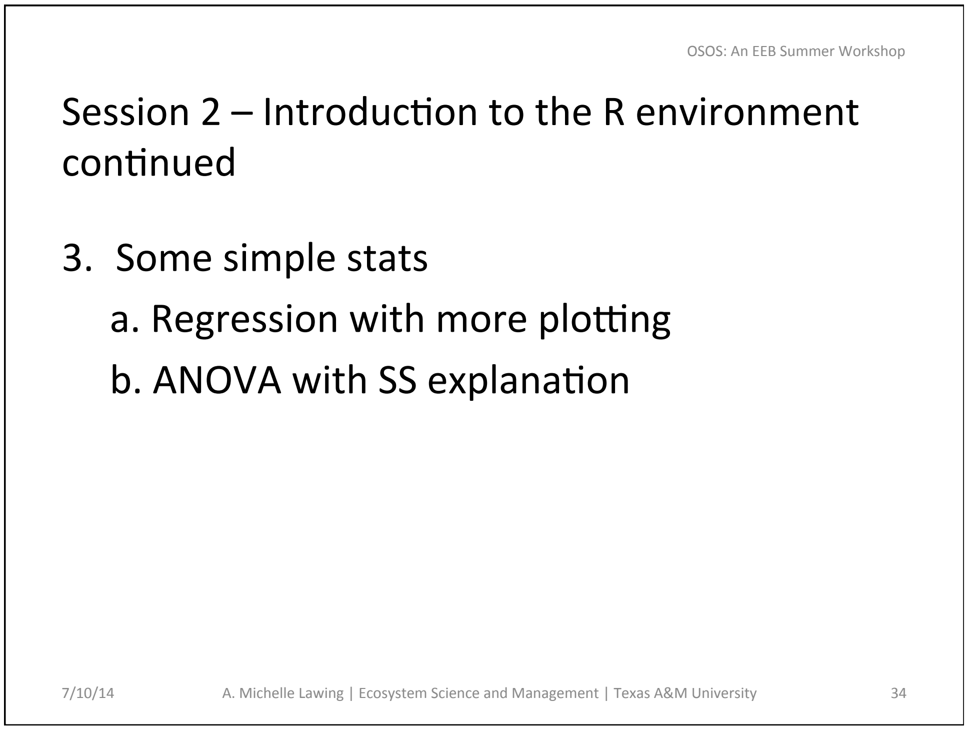
```
x <- rnorm(10)
```

2. Plot it out

```
plot(x)
```

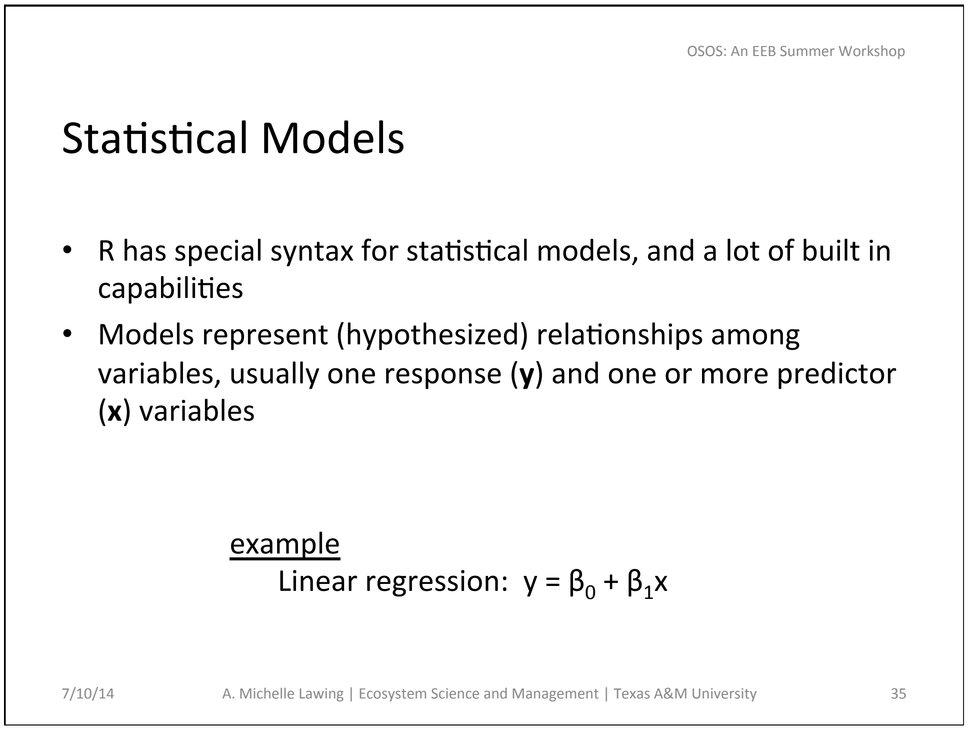
3. Interact

```
locator()           # go to graphics window and use cursor to click
                      around
identify(1:10,x)    # go to graphics window and use cursor to click
                      data points
```



Session 2 – Introduction to the R environment continued

3. Some simple stats
   a. Regression with more plotting
   b. ANOVA with SS explanation

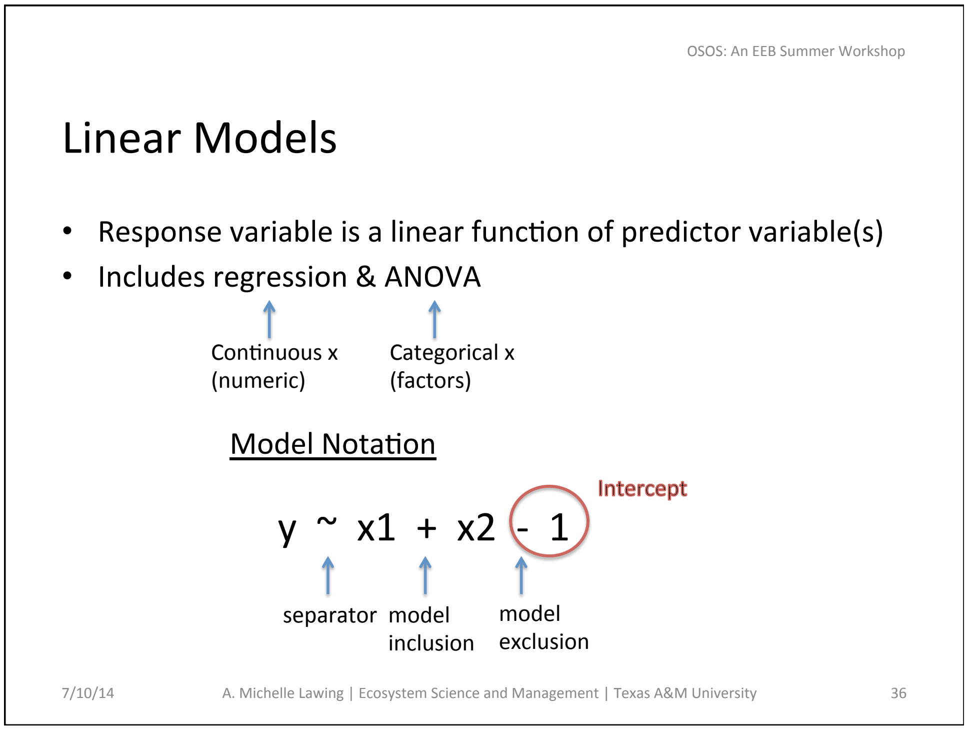


Statistical Models

- R has special syntax for statistical models, and a lot of built in capabilities
- Models represent (hypothesized) relationships among variables, usually one response (**y**) and one or more predictor (**x**) variables

example

Linear regression: $y = \beta_0 + \beta_1 x$

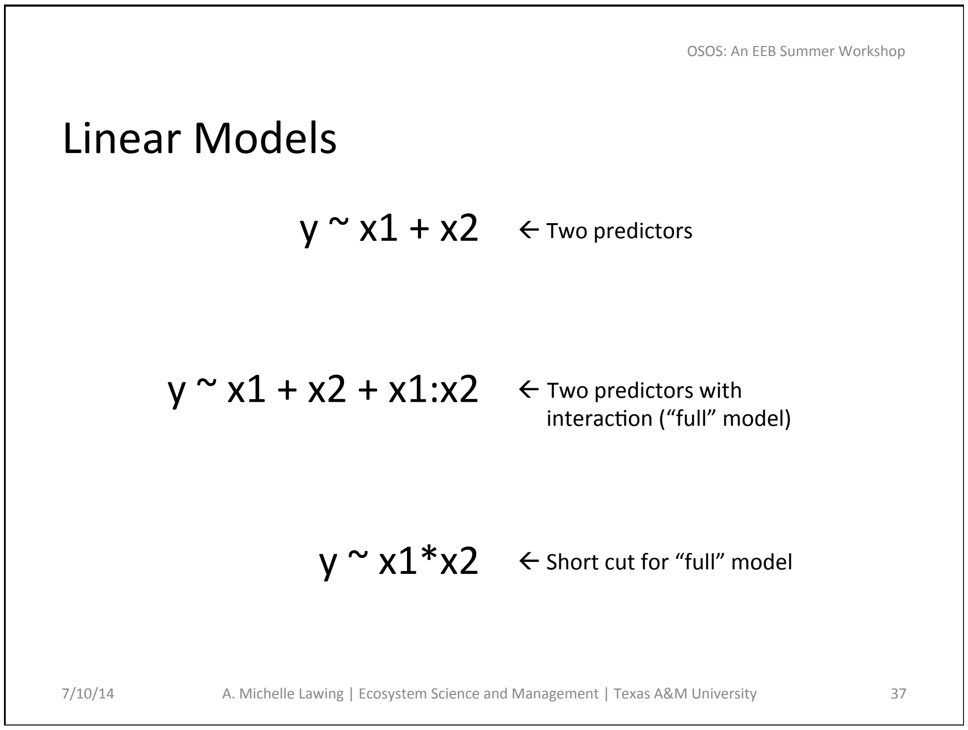


Linear Models

- Response variable is a linear function of predictor variable(s)
- Includes $\underbrace{\text{regression}}_{\substack{\text{Continuous x} \\ \text{(numeric)}}}$ & $\underbrace{\text{ANOVA}}_{\substack{\text{Categorical x} \\ \text{(factors)}}}$

Model Notation

$$y \underbrace{\sim}_{\text{separator}} x1 \underbrace{+}_{\substack{\text{model} \\ \text{inclusion}}} x2 \overset{\text{Intercept}}{\boxed{\underbrace{-}_{\substack{\text{model} \\ \text{exclusion}}}\ 1}}$$

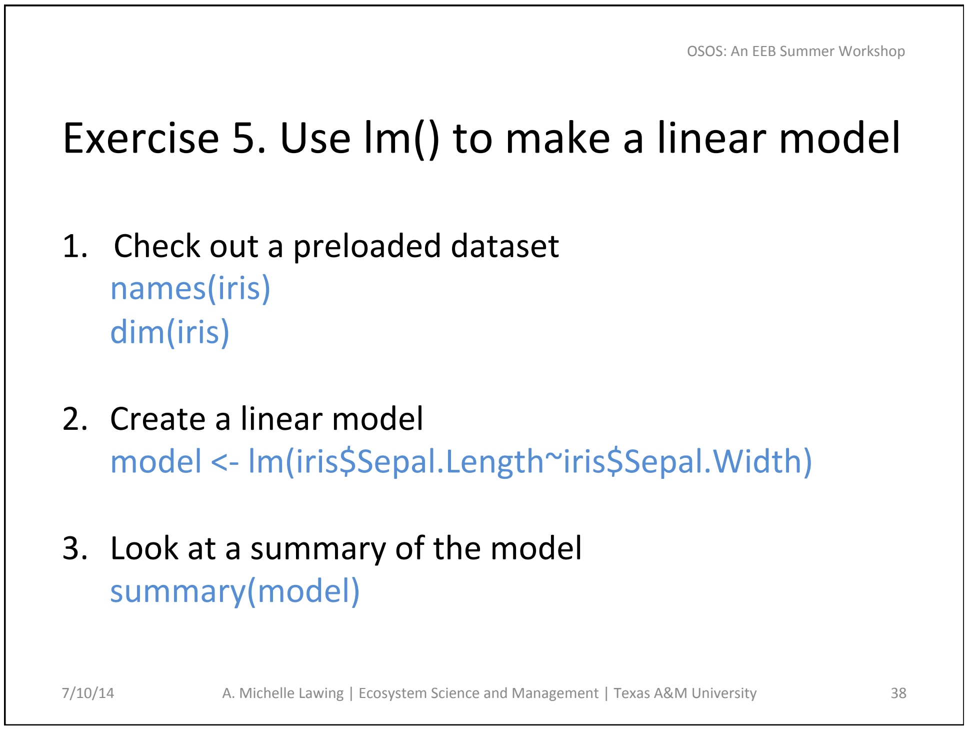


Linear Models

y ~ x1 + x2 ← Two predictors

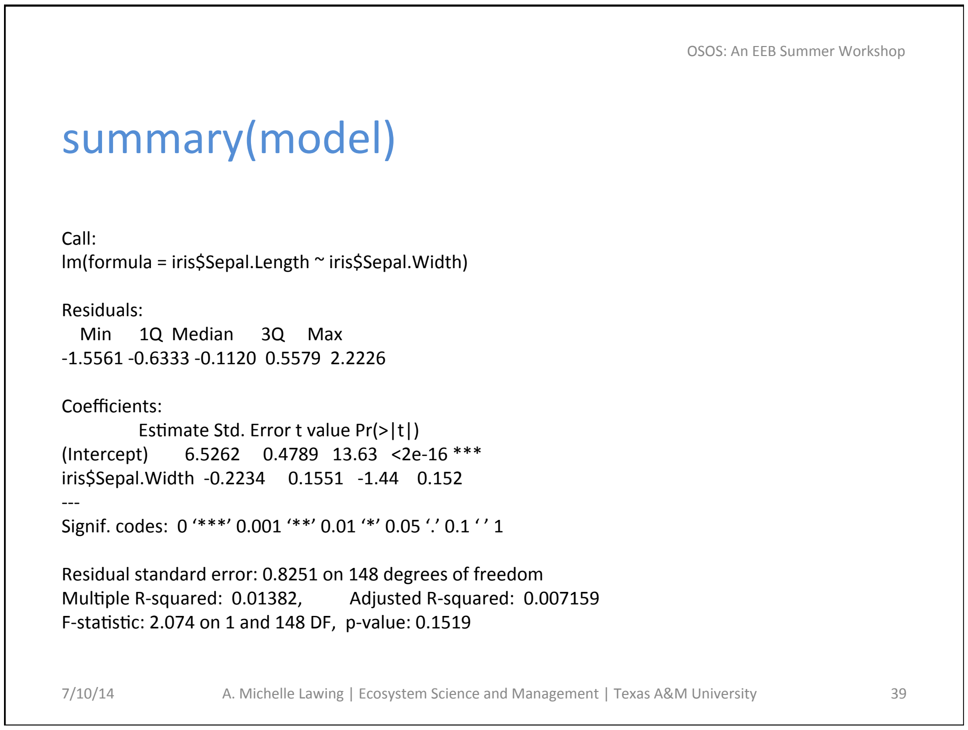
y ~ x1 + x2 + x1:x2 ← Two predictors with interaction ("full" model)

y ~ x1*x2 ← Short cut for "full" model



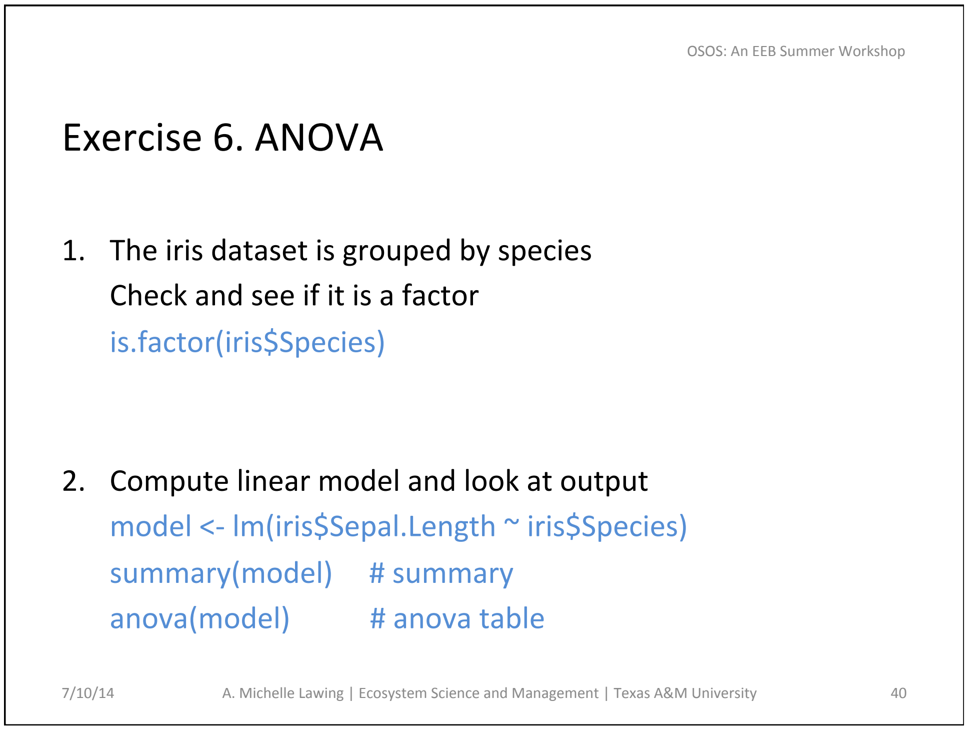
Exercise 5. Use lm() to make a linear model

1. Check out a preloaded dataset
```
names(iris)
dim(iris)
```
2. Create a linear model
```
model <- lm(iris$Sepal.Length~iris$Sepal.Width)
```
3. Look at a summary of the model
```
summary(model)
```



summary(model)

```
Call:
lm(formula = iris$Sepal.Length ~ iris$Sepal.Width)

Residuals:
    Min      1Q  Median      3Q     Max 
-1.5561 -0.6333 -0.1120  0.5579  2.2226 

Coefficients:
                 Estimate Std. Error t value Pr(>|t|)    
(Intercept)        6.5262     0.4789   13.63   <2e-16 ***
iris$Sepal.Width  -0.2234     0.1551   -1.44    0.152    
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.8251 on 148 degrees of freedom
Multiple R-squared:  0.01382,	Adjusted R-squared:  0.007159 
F-statistic: 2.074 on 1 and 148 DF,  p-value: 0.1519
```



Exercise 6. ANOVA

1. The iris dataset is grouped by species
   Check and see if it is a factor
   is.factor(iris$Species)

2. Compute linear model and look at output
   model <- lm(iris$Sepal.Length ~ iris$Species)
   summary(model) # summary
   anova(model) # anova table

ANOVA Controversy

- Sometimes R produces different results for ANOVA than other statistical programs (in default mode)
- Only happens when data are unbalanced and there is more than one factor
- The culprit is the sums of squares - R's default calculates Type I sums of squares

Type I is the sequential sums of squares
Type II assumes no interaction and calculates each ss given the other
Type III assumes significant interaction, calculates each main effect after the other and after the interaction

ANOVA Controversy

- Using the library car provides an easy way to specify the sums of squares for an anova
- function Anova()
- We will talk about libraries/packages soon

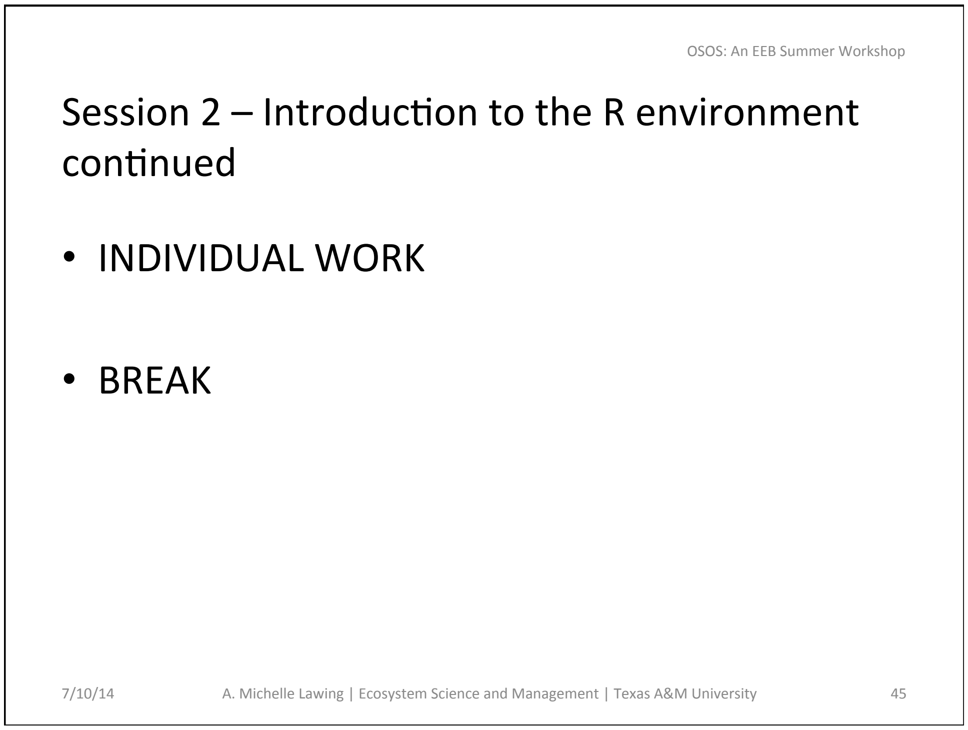
Generalizing the Linear Model

- Linear models assume: residuals are **independent**, **normal variables** with **equal variance**
- Each of these assumptions can be relaxed

Assumptions to be relaxed	Analysis
Equal Variance	Weighted Least Squares lm(model, weights=)
Normally Distributed	Generalized Linear Models glm()
Independence	Generalized least squares gls()

Regression Example

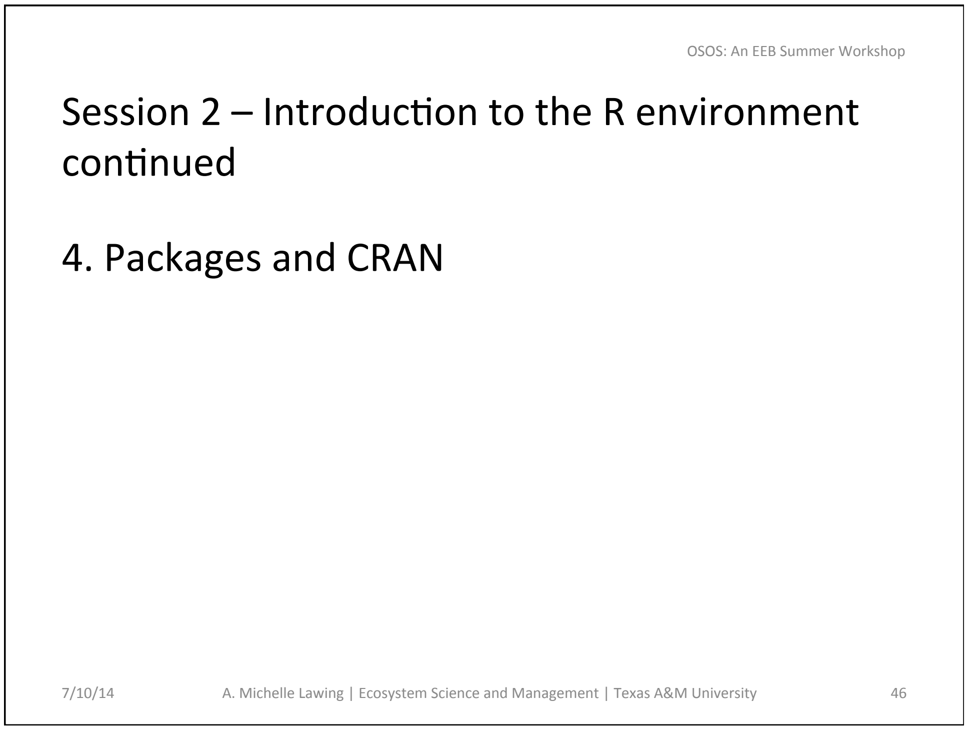
- Open Regression_Example.pdf
- Work through this document
- Copy or type commands into R (hint can also look at RegressionModel.R script)



Session 2 – Introduction to the R environment continued

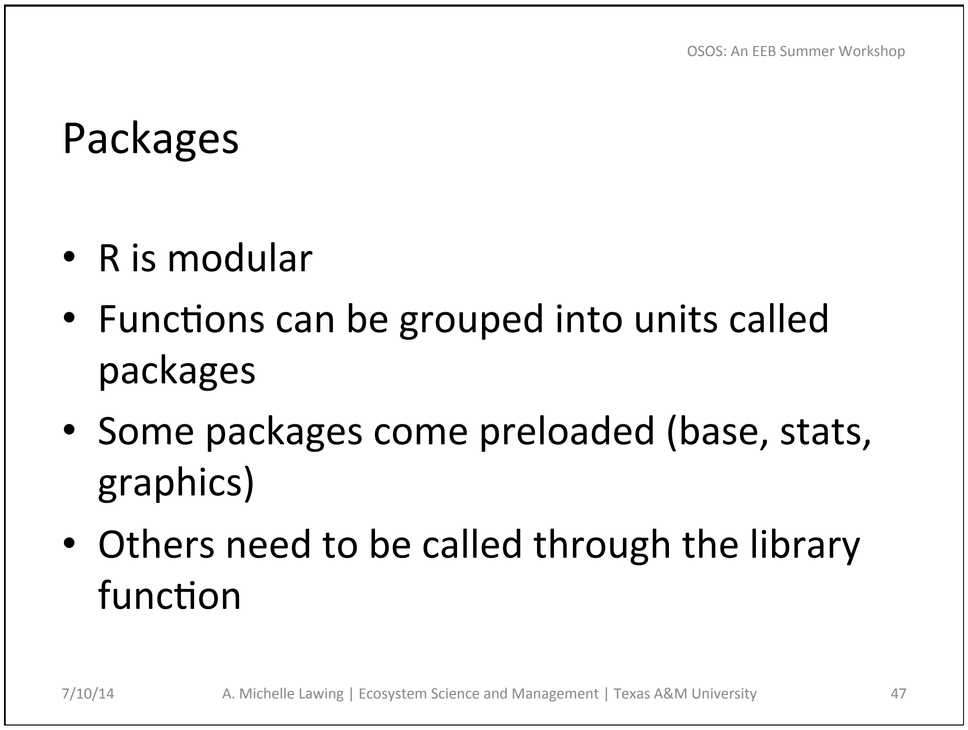
- INDIVIDUAL WORK

- BREAK



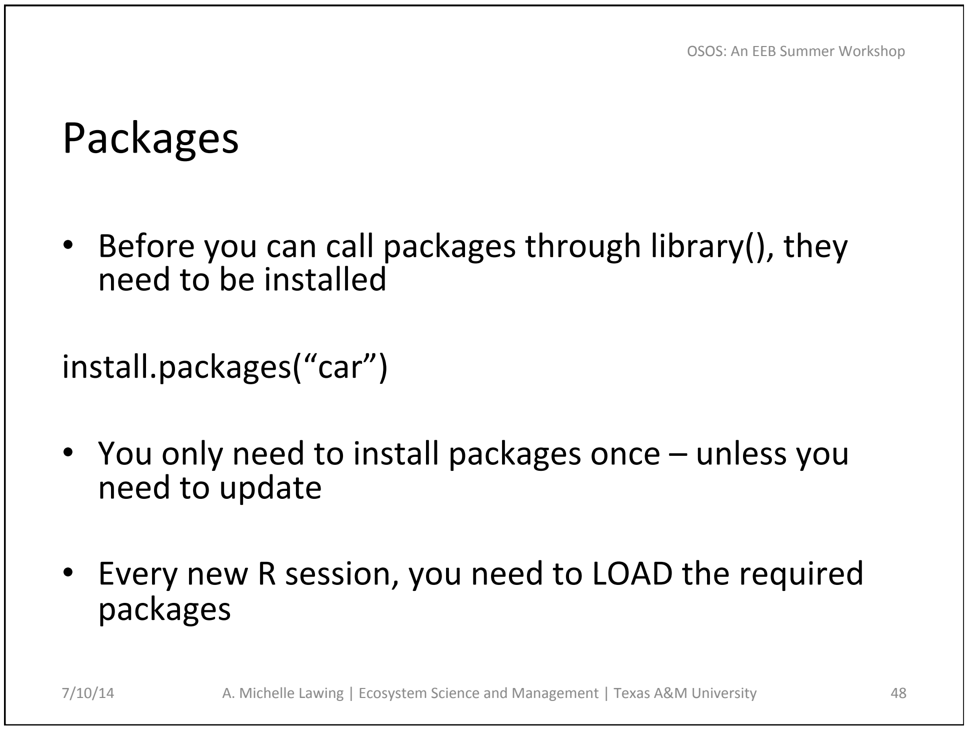
Session 2 – Introduction to the R environment continued

4. Packages and CRAN



Packages

- R is modular
- Functions can be grouped into units called packages
- Some packages come preloaded (base, stats, graphics)
- Others need to be called through the library function



Packages

- Before you can call packages through library(), they need to be installed

install.packages(“car”)

- You only need to install packages once – unless you need to update

- Every new R session, you need to LOAD the required packages

Packages

- You can list all the installed packages

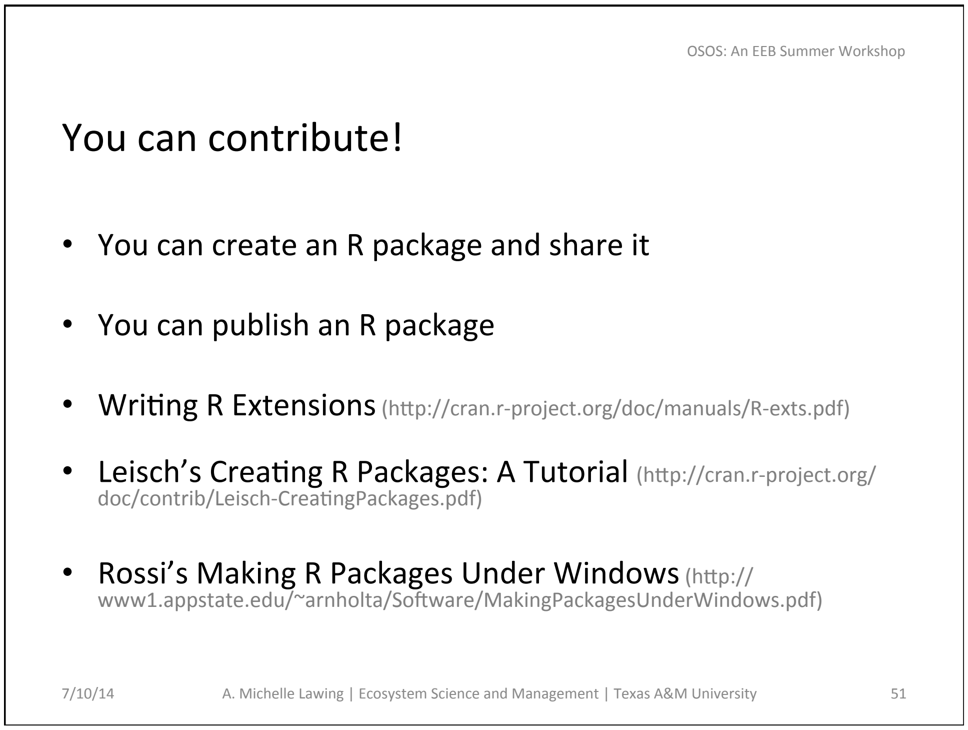
library()

- You can list all the packages currently loaded into your R session

search()

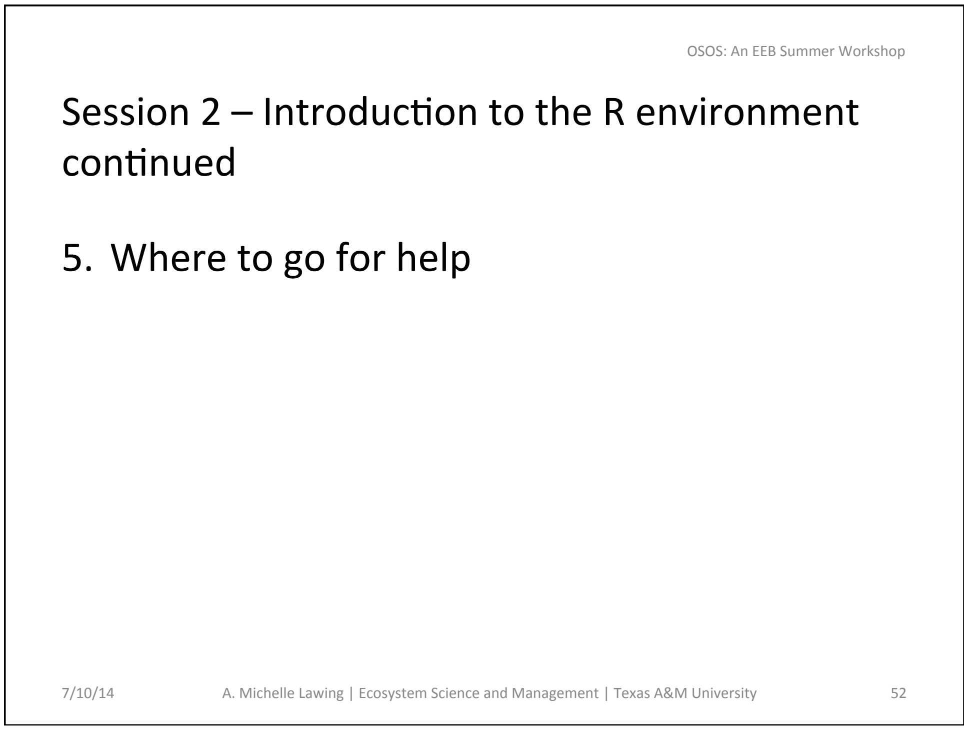
CRAN: Comprehensive R Archive Network

- CRAN is a network of ftp and web servers around the world that store identical, up-to-date, versions of code and documentation for R.
- Use the CRAN mirror nearest you to minimize network load.
- Currently there are 5,698 packages available from CRAN



You can contribute!

- You can create an R package and share it
- You can publish an R package
- Writing R Extensions (http://cran.r-project.org/doc/manuals/R-exts.pdf)
- Leisch's Creating R Packages: A Tutorial (http://cran.r-project.org/doc/contrib/Leisch-CreatingPackages.pdf)
- Rossi's Making R Packages Under Windows (http://www1.appstate.edu/~arnholta/Software/MakingPackagesUnderWindows.pdf)



Session 2 – Introduction to the R environment continued

5. Where to go for help

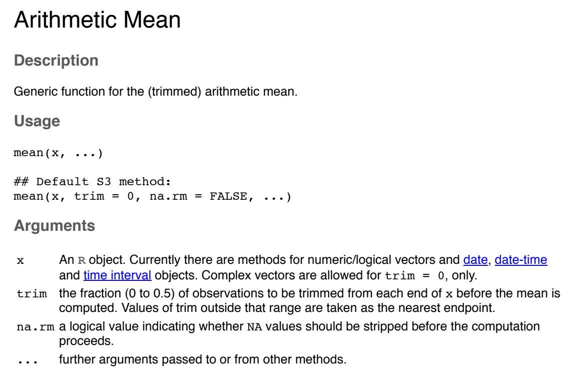
Getting Help

In R, there is a comprehensive built-in help system

help.start()	# general help
help(foo)	# help about function foo
?foo	# same thing
help.search("fooish")	# if you don't remember the function name
??fooish	# same thing

# replace foo with the name of a function

Help Documentation



Arithmetic Mean

Description

Generic function for the (trimmed) arithmetic mean.

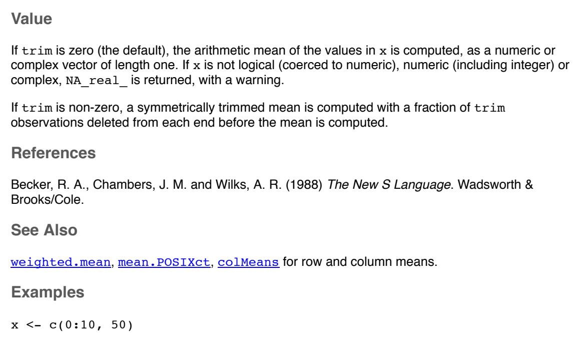
Usage

```
mean(x, ...)

## Default S3 method:
mean(x, trim = 0, na.rm = FALSE, ...)
```

Arguments

`x`	An `R` object. Currently there are methods for numeric/logical vectors and date, date-time and time interval objects. Complex vectors are allowed for `trim = 0`, only.
`trim`	the fraction (0 to 0.5) of observations to be trimmed from each end of `x` before the mean is computed. Values of trim outside that range are taken as the nearest endpoint.
`na.rm`	a logical value indicating whether `NA` values should be stripped before the computation proceeds.
`...`	further arguments passed to or from other methods.

Help Documentation



Value

If `trim` is zero (the default), the arithmetic mean of the values in `x` is computed, as a numeric or complex vector of length one. If `x` is not logical (coerced to numeric), numeric (including integer) or complex, `NA_real_` is returned, with a warning.

If `trim` is non-zero, a symmetrically trimmed mean is computed with a fraction of `trim` observations deleted from each end before the mean is computed.

References

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

See Also

`weighted.mean`, `mean.POSIXct`, `colMeans` for row and column means.

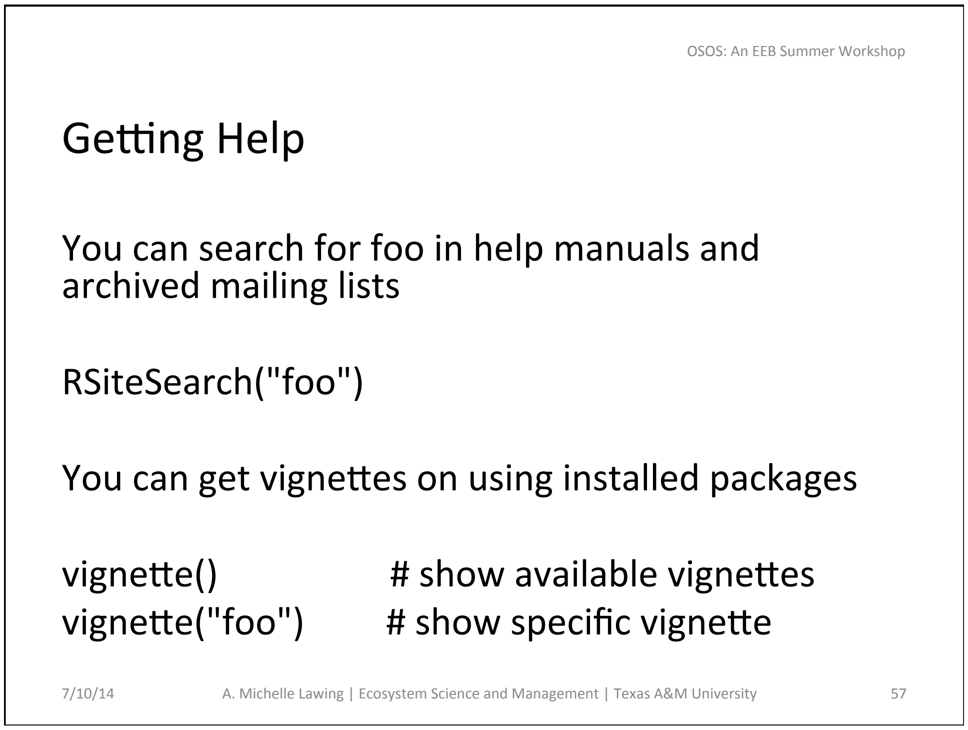
Examples

```
x <- c(0:10, 50)
```

Getting Help

apropos("foo") # list all functions containing string foo

example(foo) # show an example of function foo



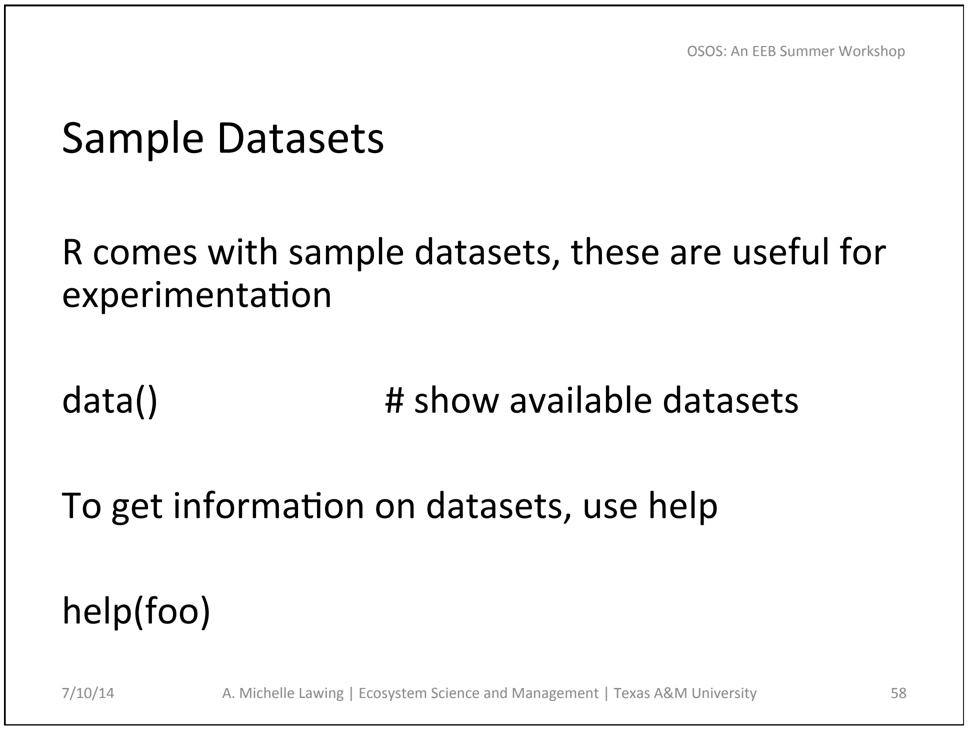
Getting Help

You can search for foo in help manuals and archived mailing lists

```
RSiteSearch("foo")
```

You can get vignettes on using installed packages

```
vignette()              # show available vignettes
vignette("foo")         # show specific vignette
```



Sample Datasets

R comes with sample datasets, these are useful for experimentation

```
data()                  # show available datasets
```

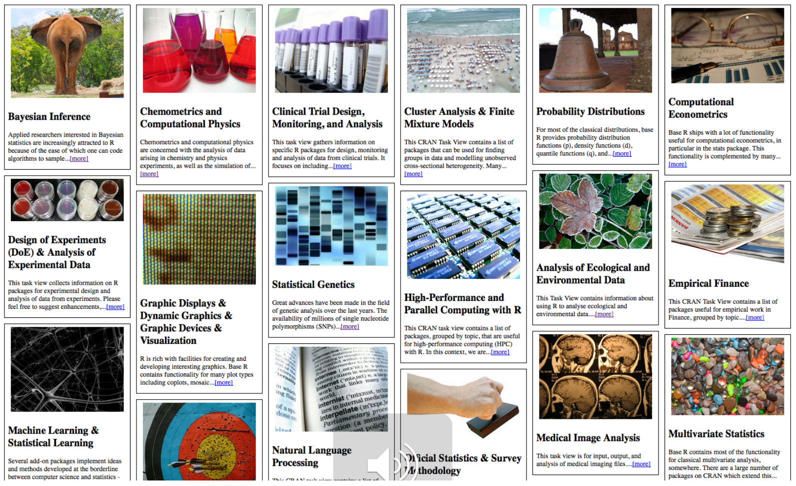
To get information on datasets, use help

```
help(foo)
```

CRAN: Task Views

- Aggregate information about packages that relate to a named task
- Useful guides to figure out what is available
- List - http://cran.r-project.org/web/views/
- Picture guide - http://www.maths.lancs.ac.uk/~rowlings/R/TaskViews/

CRAN: Task Views



Bayesian Inference

Applied researchers interested in Bayesian statistics are increasingly attracted to R because of the ease of which one can code algorithms to sample...[more]

Chemometrics and Computational Physics

Chemometrics and computational physics are concerned with the analysis of data arising in chemistry and physics experiments, as well as the simulation of...[more]

Clinical Trial Design, Monitoring, and Analysis

This task view gathers information on specific R packages for design, monitoring and analysis of data from clinical trials. It focuses on including...[more]

Cluster Analysis & Finite Mixture Models

This CRAN Task View contains a list of packages that can be used for finding groups in data and modelling unobserved cross-sectional heterogeneity. Many...[more]

Probability Distributions

For most of the classical distributions, base R provides probability distribution functions (p), density functions (d), quantile functions (q), and...[more]

Computational Econometrics

Base R ships with a lot of functionality useful for computational econometrics, in particular in the stats package. This functionality is complemented by many...[more]

Design of Experiments (DoE) & Analysis of Experimental Data

This task view collects information on R packages for experimental design and analysis of data from experiments. Please feel free to suggest enhancements,...[more]

Graphic Displays & Dynamic Graphics & Graphic Devices & Visualization

R is rich with facilities for creating and developing interesting graphics. Base R contains functionality for many plot types including coplots, mosaic...[more]

Statistical Genetics

Great advances have been made in the field of genetic analysis over the last years. The availability of millions of single nucleotide polymorphisms (SNPs)...[more]

High-Performance and Parallel Computing with R

This CRAN task view contains a list of packages, grouped by topic, that are useful for high-performance computing (HPC) with R. In this context, we are...[more]

Analysis of Ecological and Environmental Data

This Task View contains information about using R to analyse ecological and environmental data....[more]

Empirical Finance

This CRAN Task View contains a list of packages useful for empirical work in Finance, grouped by topic....[more]

Machine Learning & Statistical Learning

Several add-on packages implement ideas and methods developed at the borderline between computer science and statistics -

Natural Language Processing

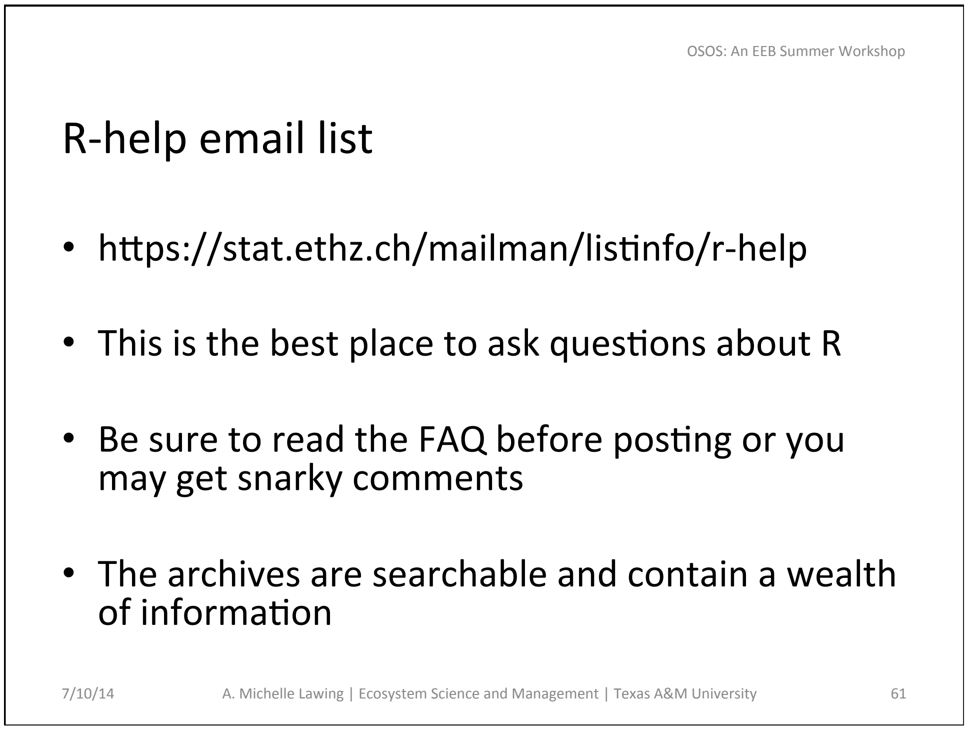
Official Statistics & Survey Methodology

Medical Image Analysis

This task view is for input, output, and analysis of medical imaging files....[more]

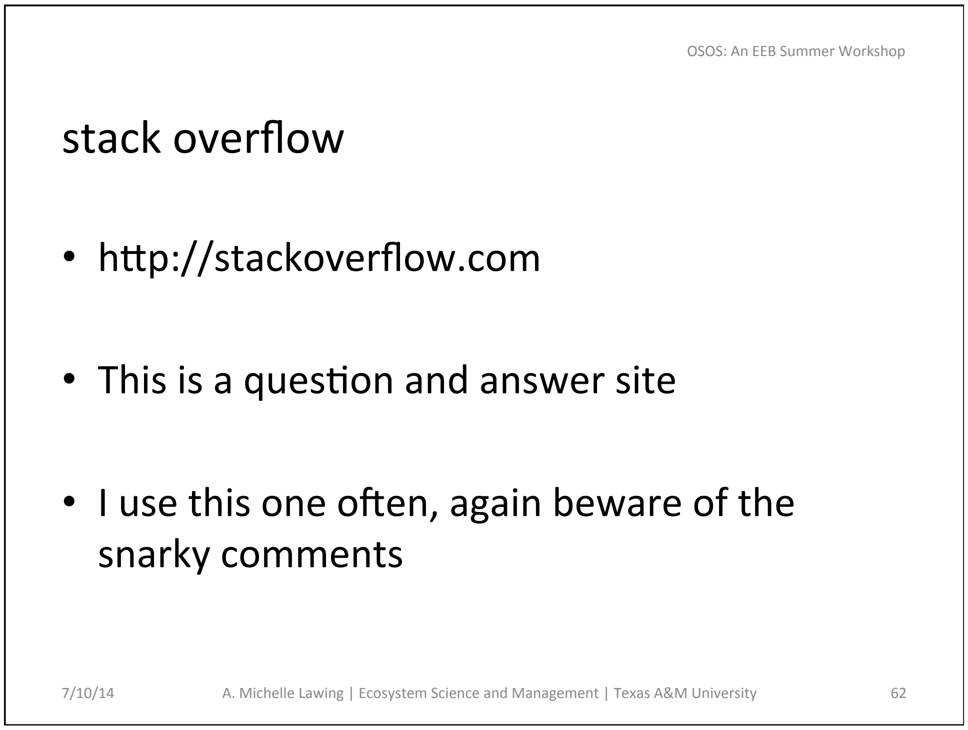
Multivariate Statistics

Base R contains most of the functionality for classical multivariate analysis, somewhere. There are a large number of packages on CRAN which extend this...



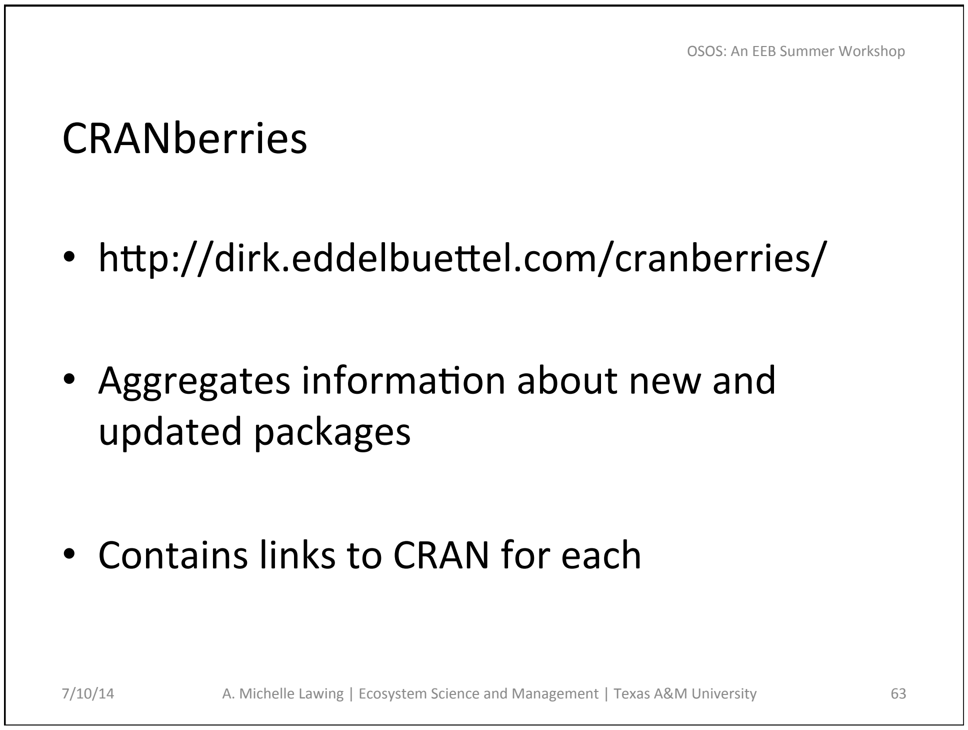
R-help email list

- https://stat.ethz.ch/mailman/listinfo/r-help
- This is the best place to ask questions about R
- Be sure to read the FAQ before posting or you may get snarky comments
- The archives are searchable and contain a wealth of information



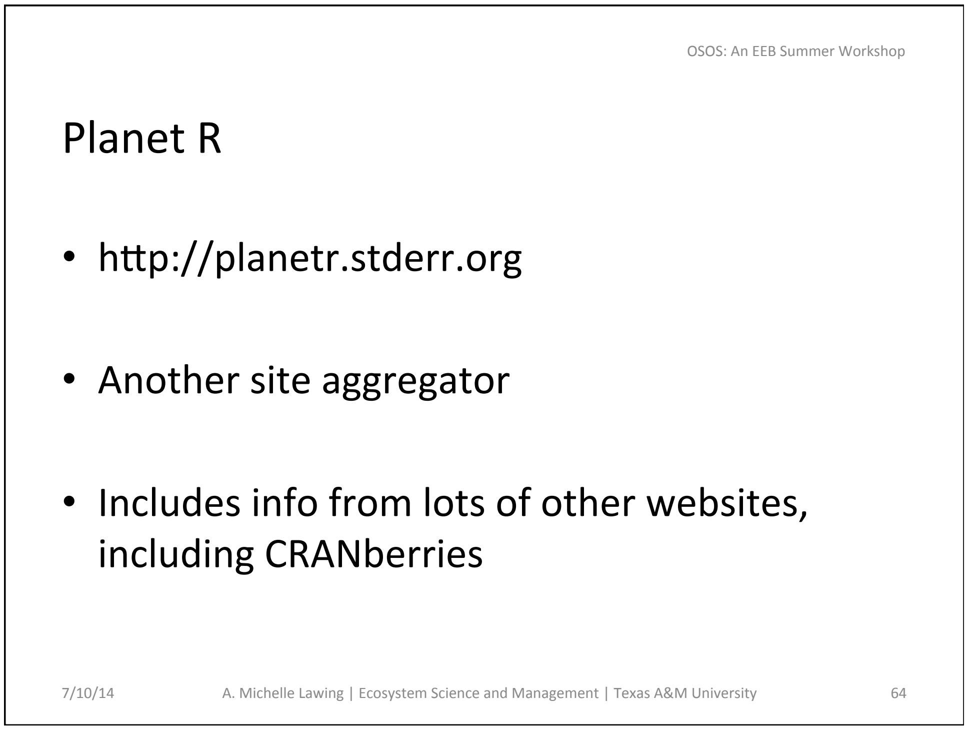
stack overflow

- http://stackoverflow.com
- This is a question and answer site
- I use this one often, again beware of the snarky comments



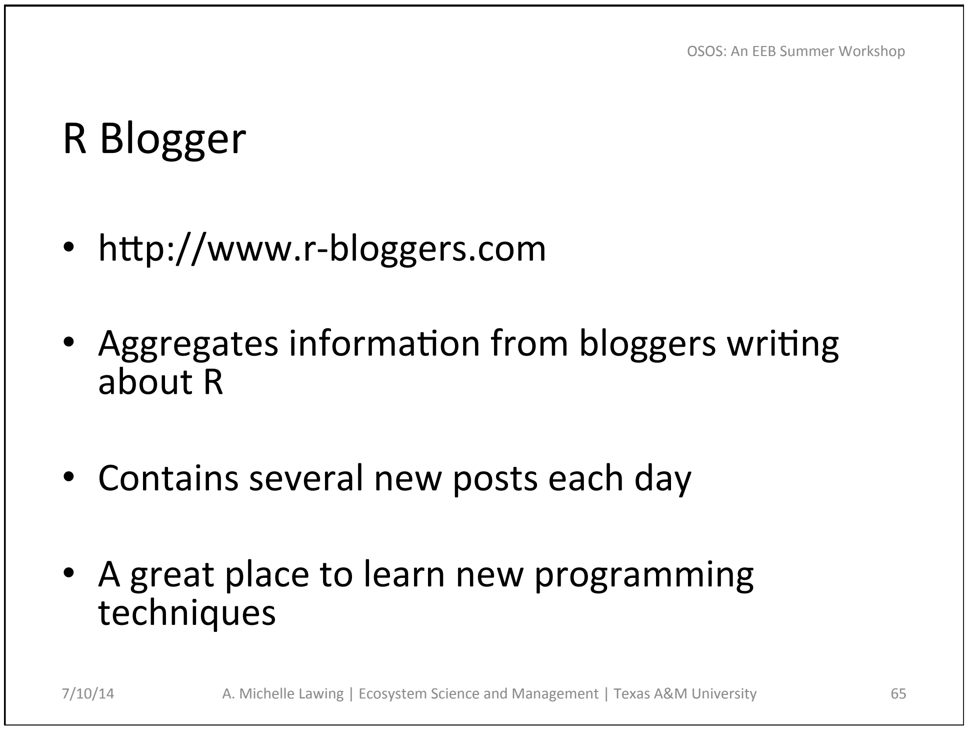
CRANberries

- http://dirk.eddelbuettel.com/cranberries/
- Aggregates information about new and updated packages
- Contains links to CRAN for each



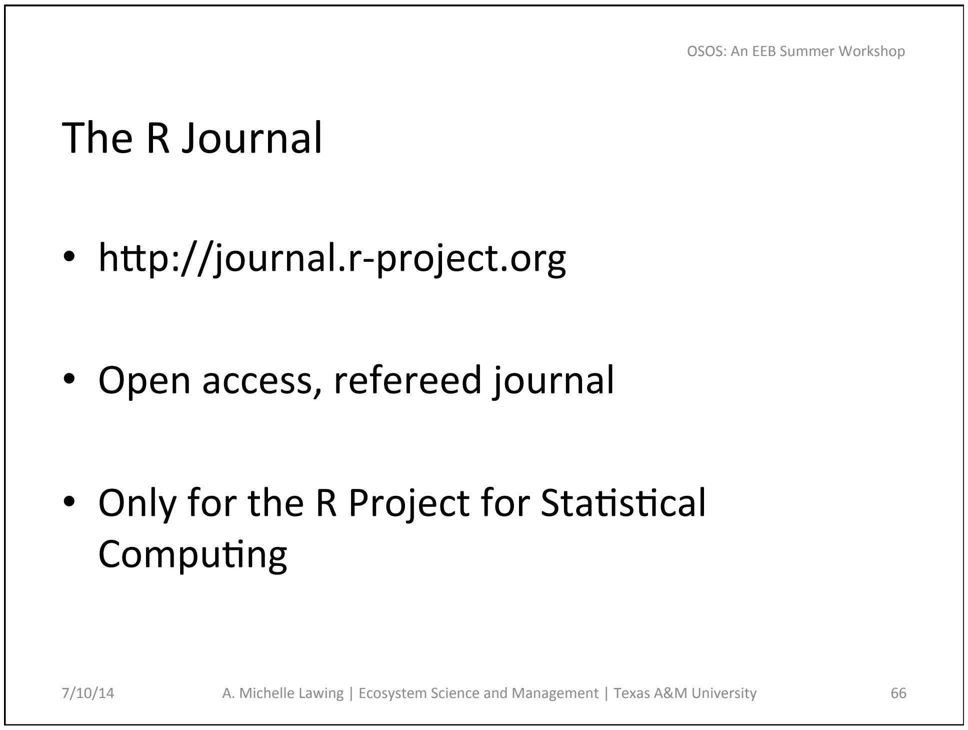
Planet R

- http://planetr.stderr.org
- Another site aggregator
- Includes info from lots of other websites, including CRANberries



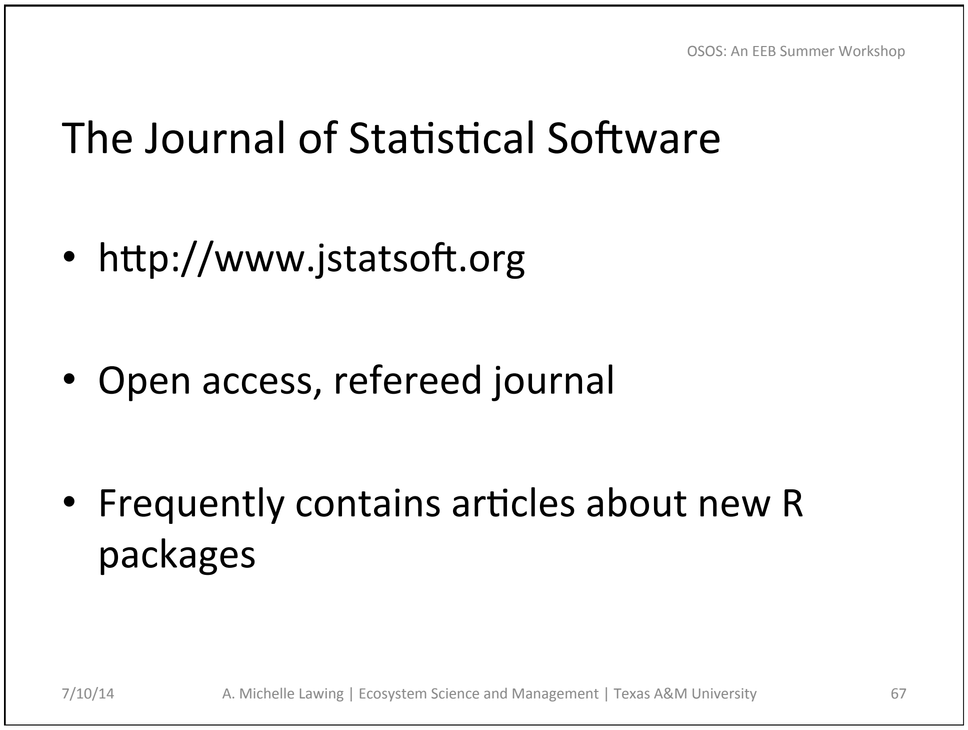
R Blogger

- http://www.r-bloggers.com
- Aggregates information from bloggers writing about R
- Contains several new posts each day
- A great place to learn new programming techniques



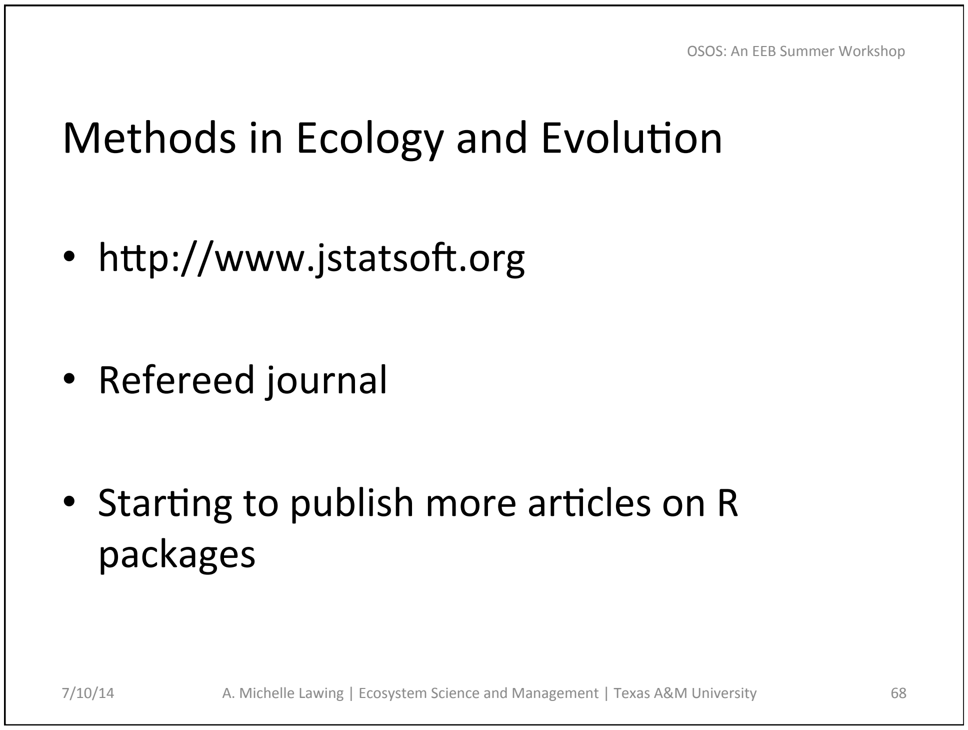
The R Journal

- http://journal.r-project.org
- Open access, refereed journal
- Only for the R Project for Statistical Computing



The Journal of Statistical Software

- http://www.jstatsoft.org
- Open access, refereed journal
- Frequently contains articles about new R packages



Methods in Ecology and Evolution

- http://www.jstatsoft.org
- Refereed journal
- Starting to publish more articles on R packages

Getting Help

Google

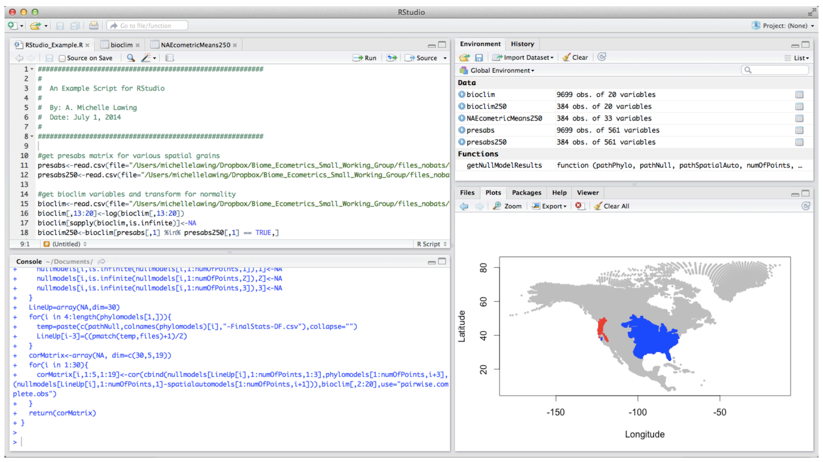
Session 2 – Introduction to the R environment continued

6. RStudio

RStudio

- Integrated Development Environment
  - Source
  - Console
  - Workspace
  - Plots
- Windows, Mac and Linux
- Desktop and server versions (can run over the web)

RStudio screen has four windows

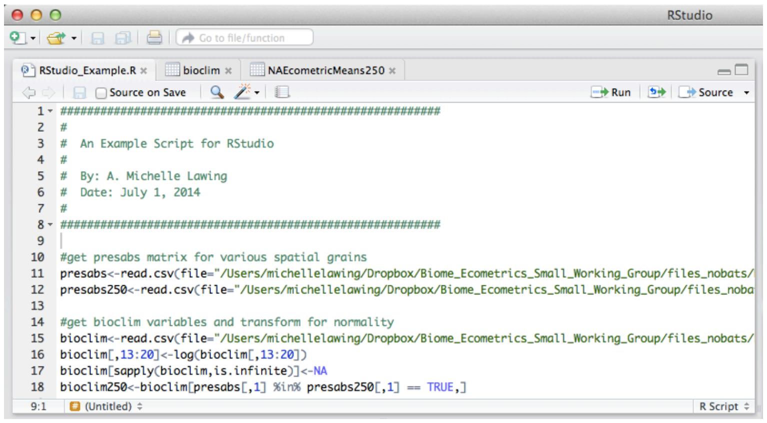


```
##########################################################
#
#  An Example Script for RStudio
#
#  By: A. Michelle Lawing
#  Date: July 1, 2014
#
##########################################################

#get presabs matrix for various spatial grains
presabs<-read.csv(file="/Users/michellelawing/Dropbox/Biome_Ecometrics_Small_Working_Group/files_nobats/
presabs250<-read.csv(file="/Users/michellelawing/Dropbox/Biome_Ecometrics_Small_Working_Group/files_noba

#get bioclim variables and transform for normality
bioclim<-read.csv(file="/Users/michellelawing/Dropbox/Biome_Ecometrics_Small_Working_Group/files_nobats/
bioclim[,13:20]<-log(bioclim[,13:20])
bioclim[sapply(bioclim,is.infinite)]<-NA
bioclim250<-bioclim[presabs[,1] %in% presabs250[,1] == TRUE,]
```

Data	
bioclim	9699 obs. of 20 variables
bioclim250	384 obs. of 20 variables
NAEcometricMeans250	384 obs. of 33 variables
presabs	9699 obs. of 561 variables
presabs250	384 obs. of 561 variables
Functions	
getNullModelResults	function (pathPhylo, pathNull, pathSpatialAuto, numOfPoints, …

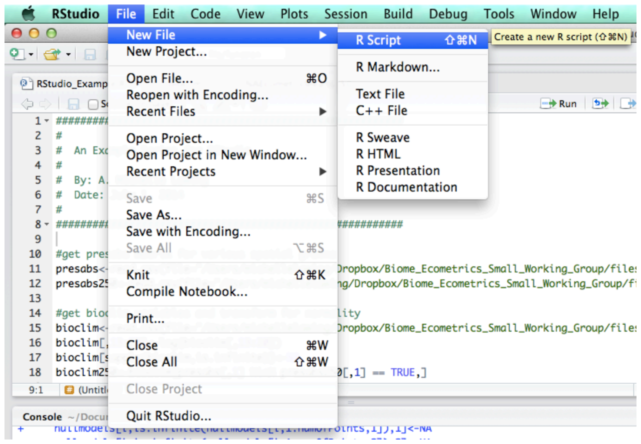
```
+     nullmodels[i,is.infinite(nullmodels[i,1:numOfPoints,1]),1]<-NA
+     nullmodels[i,is.infinite(nullmodels[i,1:numOfPoints,2]),2]<-NA
+     nullmodels[i,is.infinite(nullmodels[i,1:numOfPoints,3]),3]<-NA
+   }
+   LineUp=array(NA,dim=30)
+   for(i in 4:length(phylomodels[1,])){
+     temp=paste(c(pathNull,colnames(phylomodels)[i],"-FinalStats-DF.csv"),collapse="")
+     LineUp[i-3]=((pmatch(temp,files)+1)/2)
+   }
+   corMatrix<-array(NA, dim=c(30,5,19))
+   for(i in 1:30){
+     corMatrix[i,1:5,1:19]<-cor(cbind(nullmodels[LineUp[i],1:numOfPoints,1:3],phylomodels[1:numOfPoints,i+3],
(nullmodels[LineUp[i],1:numOfPoints,1]-spatialautomodels[1:numOfPoints,i+1])),bioclim[,2:20],use="pairwise.com
plete.obs")
+   }
+   return(corMatrix)
+ }
>
> |
```

Script Editor

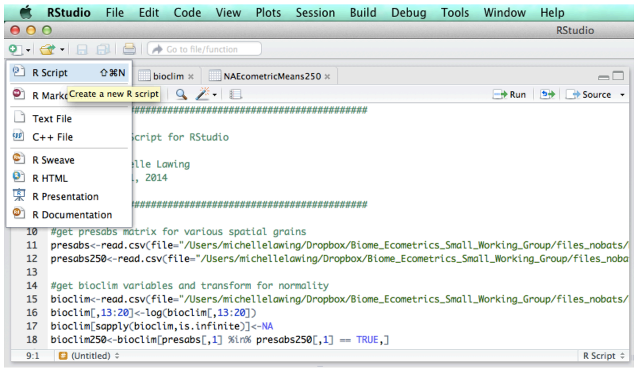


```
#########################################################
#
#   An Example Script for RStudio
#
#   By: A. Michelle Lawing
#   Date: July 1, 2014
#
#########################################################

#get presabs matrix for various spatial grains
presabs<-read.csv(file="/Users/michellelawing/Dropbox/Biome_Ecometrics_Small_Working_Group/files_nobats/
presabs250<-read.csv(file="/Users/michellelawing/Dropbox/Biome_Ecometrics_Small_Working_Group/files_noba

#get bioclim variables and transform for normality
bioclim<-read.csv(file="/Users/michellelawing/Dropbox/Biome_Ecometrics_Small_Working_Group/files_nobats/
bioclim[,13:20]<-log(bioclim[,13:20])
bioclim[sapply(bioclim,is.infinite)]<-NA
bioclim250<-bioclim[presabs[,1] %in% presabs250[,1] == TRUE,]
```

Create a New Script

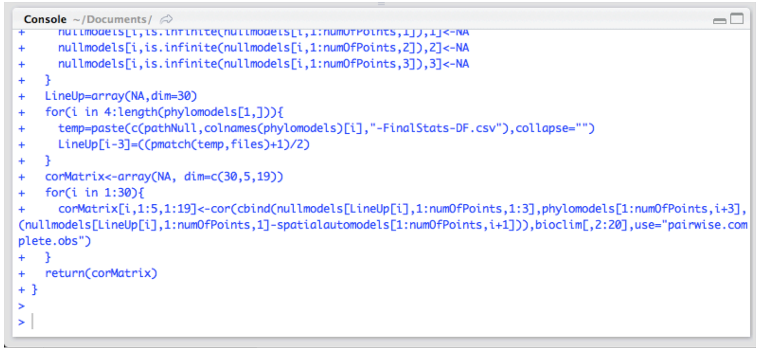


Create a New Script



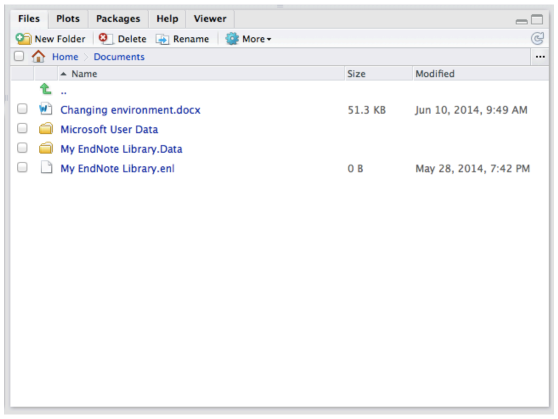
```
##########################################
cript for RStudio

lle Lawing
, 2014

##########################################
#get presabs matrix for various spatial grains
presabs<-read.csv(file="/Users/michellelawing/Dropbox/Biome_Ecometrics_Small_Working_Group/files_nobats/
presabs250<-read.csv(file="/Users/michellelawing/Dropbox/Biome_Ecometrics_Small_Working_Group/files_noba

#get bioclim variables and transform for normality
bioclim<-read.csv(file="/Users/michellelawing/Dropbox/Biome_Ecometrics_Small_Working_Group/files_nobats/
bioclim[,13:20]<-log(bioclim[,13:20])
bioclim[sapply(bioclim,is.infinite)]<-NA
bioclim250<-bioclim[presabs[,1] %in% presabs250[,1] == TRUE,]
```

Console



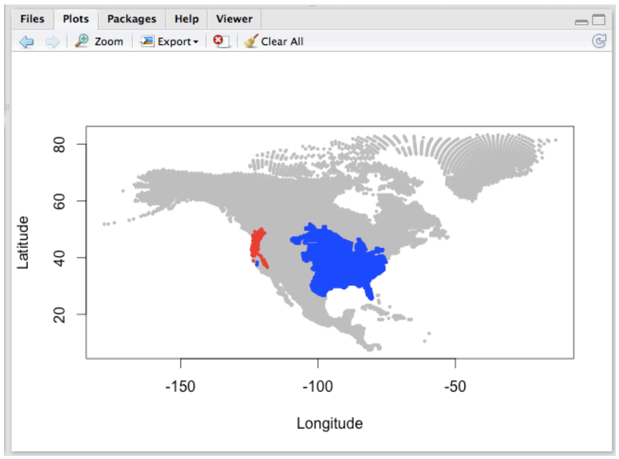
```
+     nullmodels[i,is.infinite(nullmodels[i,1:numOfPoints,1]),1]<-NA
+     nullmodels[i,is.infinite(nullmodels[i,1:numOfPoints,2]),2]<-NA
+     nullmodels[i,is.infinite(nullmodels[i,1:numOfPoints,3]),3]<-NA
+   }
+   LineUp=array(NA,dim=30)
+   for(i in 4:length(phylomodels[1,])){
+     temp=paste(c(pathNull,colnames(phylomodels)[i],"-FinalStats-DF.csv"),collapse="")
+     LineUp[i-3]=((pmatch(temp,files)+1)/2)
+   }
+   corMatrix<-array(NA, dim=c(30,5,19))
+   for(i in 1:30){
+     corMatrix[i,1:5,1:19]<-cor(cbind(nullmodels[LineUp[i],1:numOfPoints,1:3],phylomodels[1:numOfPoints,i+3],
(nullmodels[LineUp[i],1:numOfPoints,1]-spatialautomodels[1:numOfPoints,i+1])),bioclim[,2:20],use="pairwise.com
plete.obs")
+   }
+   return(corMatrix)
+ }
>
> |
```

Files Tab

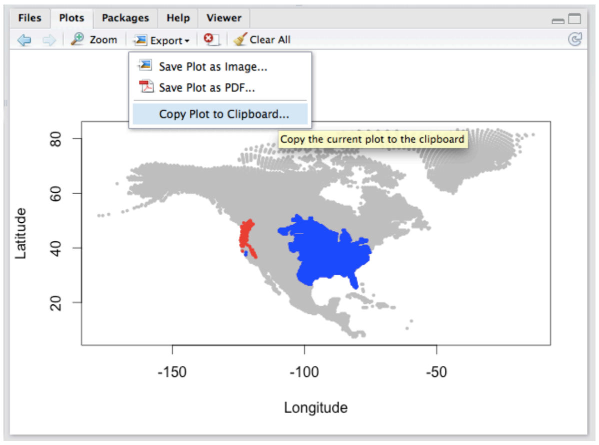


Name	Size	Modified
..		
Changing environment.docx	51.3 KB	Jun 10, 2014, 9:49 AM
Microsoft User Data		
My EndNote Library.Data		
My EndNote Library.enl	0 B	May 28, 2014, 7:42 PM

Plots Tab



Export Graphics



Export Graphics

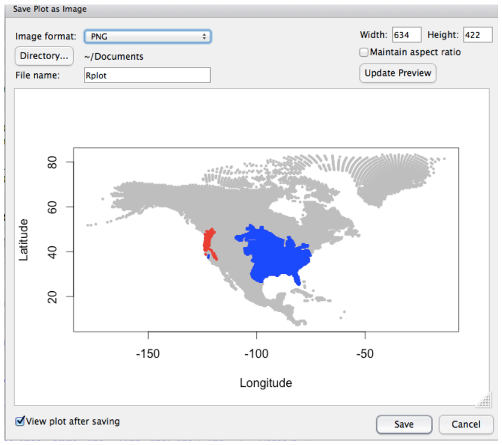

Save Plot as Image

Image format: PNG
Directory... ~/Documents
File name: Rplot
Width: 634 Height: 422
Maintain aspect ratio
View plot after saving

Export Graphics

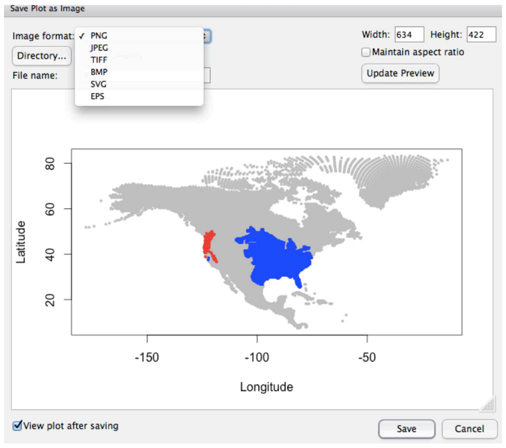


Image format: ✓ PNG
JPEG
TIFF
BMP
SVG
EPS

File name:

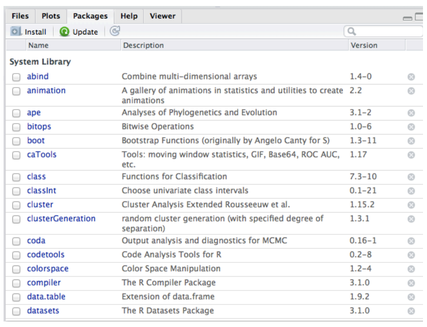
Width: 634 Height: 422

☐ Maintain aspect ratio

☑ View plot after saving

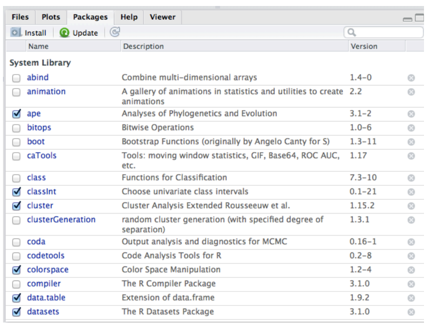
Packages

Check the box to load library. Click on the install button to download libraries not on the list.



System Library

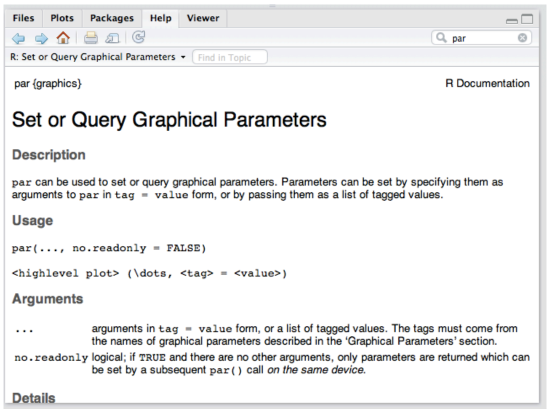
	Name	Description	Version
☐	abind	Combine multi-dimensional arrays	1.4-0
☐	animation	A gallery of animations in statistics and utilities to create animations	2.2
☐	ape	Analyses of Phylogenetics and Evolution	3.1-2
☐	bitops	Bitwise Operations	1.0-6
☐	boot	Bootstrap Functions (originally by Angelo Canty for S)	1.3-11
☐	caTools	Tools: moving window statistics, GIF, Base64, ROC AUC, etc.	1.17
☐	class	Functions for Classification	7.3-10
☐	classInt	Choose univariate class intervals	0.1-21
☐	cluster	Cluster Analysis Extended Rousseeuw et al.	1.15.2
☐	clusterGeneration	random cluster generation (with specified degree of separation)	1.3.1
☐	coda	Output analysis and diagnostics for MCMC	0.16-1
☐	codetools	Code Analysis Tools for R	0.2-8
☐	colorspace	Color Space Manipulation	1.2-4
☐	compiler	The R Compiler Package	3.1.0
☐	data.table	Extension of data.frame	1.9.2
☐	datasets	The R Datasets Package	3.1.0



System Library

	Name	Description	Version
☐	abind	Combine multi-dimensional arrays	1.4-0
☐	animation	A gallery of animations in statistics and utilities to create animations	2.2
☑	ape	Analyses of Phylogenetics and Evolution	3.1-2
☐	bitops	Bitwise Operations	1.0-6
☐	boot	Bootstrap Functions (originally by Angelo Canty for S)	1.3-11
☐	caTools	Tools: moving window statistics, GIF, Base64, ROC AUC, etc.	1.17
☐	class	Functions for Classification	7.3-10
☑	classInt	Choose univariate class intervals	0.1-21
☑	cluster	Cluster Analysis Extended Rousseeuw et al.	1.15.2
☐	clusterGeneration	random cluster generation (with specified degree of separation)	1.3.1
☐	coda	Output analysis and diagnostics for MCMC	0.16-1
☐	codetools	Code Analysis Tools for R	0.2-8
☑	colorspace	Color Space Manipulation	1.2-4
☐	compiler	The R Compiler Package	3.1.0
☑	data.table	Extension of data.frame	1.9.2
☑	datasets	The R Datasets Package	3.1.0

Help



par {graphics} R Documentation

Set or Query Graphical Parameters

Description

`par` can be used to set or query graphical parameters. Parameters can be set by specifying them as arguments to `par` in `tag = value` form, or by passing them as a list of tagged values.

Usage

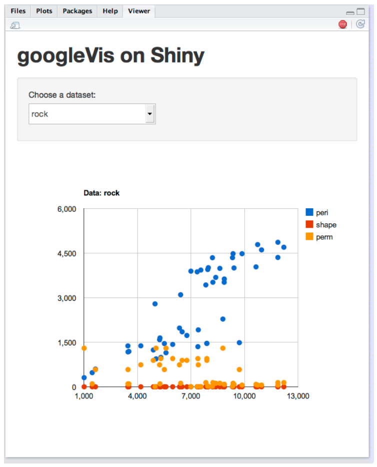
```
par(..., no.readonly = FALSE)

<highlevel plot> (\dots, <tag> = <value>)
```

Arguments

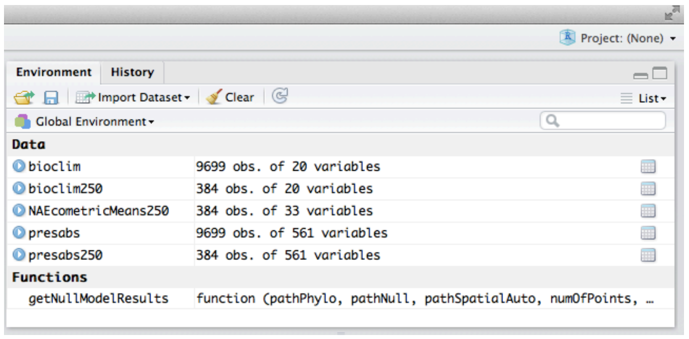
`...`	arguments in `tag = value` form, or a list of tagged values. The tags must come from the names of graphical parameters described in the 'Graphical Parameters' section.
`no.readonly`	logical; if `TRUE` and there are no other arguments, only parameters are returned which can be set by a subsequent `par()` call *on the same device*.

Details

Viewer Tab

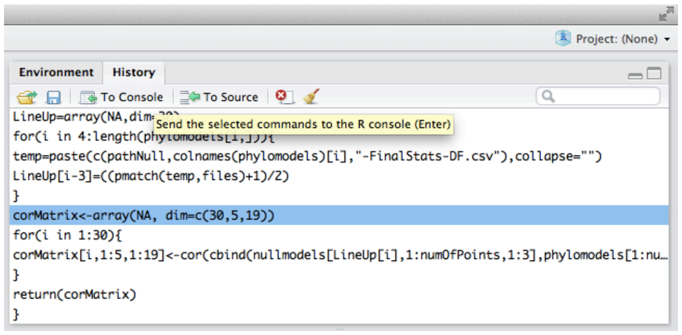


Environment Tab



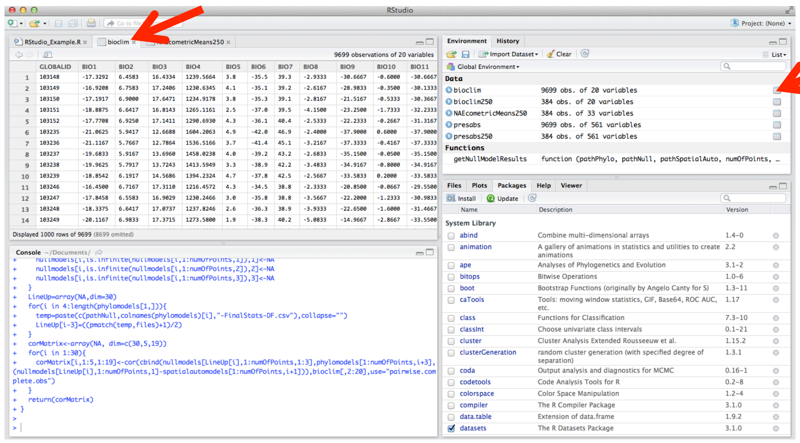
```
Data
bioclim               9699 obs. of 20 variables
bioclim250            384 obs. of 20 variables
NAEcometricMeans250   384 obs. of 33 variables
presabs               9699 obs. of 561 variables
presabs250            384 obs. of 561 variables
Functions
getNullModelResults   function (pathPhylo, pathNull, pathSpatialAuto, numOfPoints, …
```

History Tab



```
LineUp=array(NA,dim=30)
for(i in 4:length(phylomodels[1,])){
temp=paste(c(pathNull,colnames(phylomodels)[i],"-FinalStats-DF.csv"),collapse="")
LineUp[i-3]=((pmatch(temp,files)+1)/2)
}
corMatrix<-array(NA, dim=c(30,5,19))
for(i in 1:30){
corMatrix[i,1:5,1:19]<-cor(cbind(nullmodels[LineUp[i],1:numOfPoints,1:3],phylomodels[1:nu…
}
return(corMatrix)
}
```

Tabular Data with Script Editor



9699 observations of 20 variables

	GLOBALID	BIO1	BIO2	BIO3	BIO4	BIO5	BIO6	BIO7	BIO8	BIO9	BIO10	BIO11
1	103148	-17.3292	6.4583	16.4334	1239.5664	3.8	-35.5	39.3	-2.9333	-30.6667	-0.6000	-30.6667
2	103149	-16.9208	6.7583	17.2406	1230.6345	4.1	-35.1	39.2	-2.6167	-28.9833	-0.3500	-30.1333
3	103150	-17.1917	6.9000	17.6471	1234.9178	3.8	-35.3	39.1	-2.8167	-21.5167	-0.5333	-30.3667
4	103151	-18.8875	6.6417	16.8143	1265.1161	2.5	-37.0	39.5	-4.1500	-23.2500	-1.7333	-32.2333
5	103152	-17.7708	6.9250	17.1411	1290.6930	4.3	-36.1	40.4	-2.5333	-22.2333	-0.2667	-31.3167
6	103235	-21.0625	5.9417	12.6688	1604.2063	4.9	-42.0	46.9	-2.4000	-37.9000	0.6000	-37.9000
7	103236	-21.1167	5.7667	12.7864	1536.5166	3.7	-41.4	45.1	-3.2167	-37.3333	-0.4167	-37.3333
8	103237	-19.6833	5.9167	13.6960	1458.0238	4.0	-39.2	43.2	-2.6833	-35.1500	-0.0500	-35.1500
9	103238	-19.9625	5.7917	13.7243	1413.5949	3.3	-38.9	42.2	-3.4833	-34.9167	-0.8000	-34.9167
10	103239	-18.8542	6.1917	14.5686	1394.2324	4.7	-37.8	42.5	-2.5667	-33.5833	0.2000	-33.5833
11	103246	-16.4500	6.7167	17.3110	1216.4572	4.3	-34.5	38.8	-2.3333	-20.8500	-0.0667	-29.5500
12	103247	-17.8458	6.5583	16.9029	1230.2466	3.0	-35.8	38.8	-3.5667	-22.2000	-1.2333	-30.9833
13	103248	-18.3375	6.6417	17.0737	1237.8246	2.6	-36.3	38.9	-3.9333	-22.6500	-1.6000	-31.4667
14	103249	-20.1167	6.9833	17.3715	1273.5800	1.9	-38.3	40.2	-5.0833	-14.9667	-2.8667	-33.5500

Displayed 1000 rows of 9699 (8699 omitted)

Console ~/Documents/

```
+     nullmodels[i,is.infinite(nullmodels[i,1:numOfPoints,1]),1]<-NA
+     nullmodels[i,is.infinite(nullmodels[i,1:numOfPoints,2]),2]<-NA
+     nullmodels[i,is.infinite(nullmodels[i,1:numOfPoints,3]),3]<-NA
+   }
+   LineUp=array(NA,dim=30)
+   for(i in 4:length(phylomodels[1,])){
+     temp=paste(c(pathNull,colnames(phylomodels)[i],"-FinalStats-DF.csv"),collapse="")
+     LineUp[i-3]=((pmatch(temp,files)+1)/2)
+   }
+   corMatrix<-array(NA, dim=c(30,5,19))
+   for(i in 1:30){
+     corMatrix[i,1:5,1:19]<-cor(cbind(nullmodels[LineUp[i],1:numOfPoints,1:3],phylomodels[1:numOfPoints,i+3],
(nullmodels[LineUp[i],1:numOfPoints,1]-spatialautomodels[1:numOfPoints,i+1])),bioclim[,2:20],use="pairwise.com
plete.obs")
+   }
+   return(corMatrix)
+ }
>
> |
```

Data

bioclim	9699 obs. of 20 variables
bioclim250	384 obs. of 20 variables
NAEcometricMeans250	384 obs. of 33 variables
presabs	9699 obs. of 561 variables
presabs250	384 obs. of 561 variables

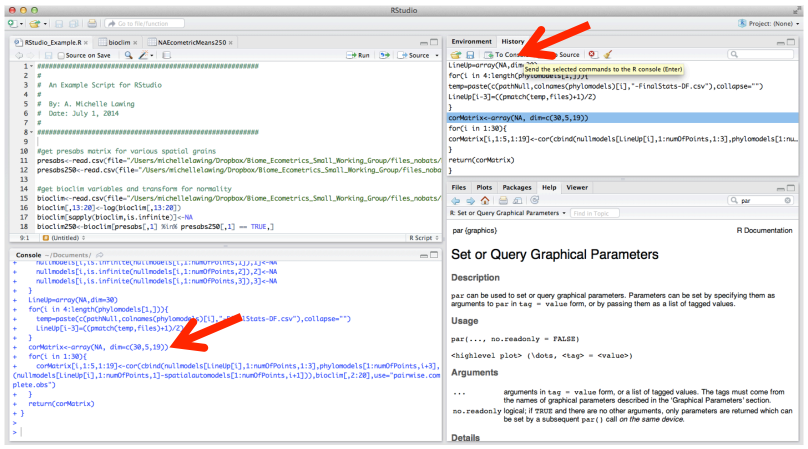
Functions

getNullModelResults	function (pathPhylo, pathNull, pathSpatialAuto, numOfPoints, …

System Library

	Name	Description	Version
☐	abind	Combine multi-dimensional arrays	1.4-0
☐	animation	A gallery of animations in statistics and utilities to create animations	2.2
☐	ape	Analyses of Phylogenetics and Evolution	3.1-2
☐	bitops	Bitwise Operations	1.0-6
☐	boot	Bootstrap Functions (originally by Angelo Canty for S)	1.3-11
☐	caTools	Tools: moving window statistics, GIF, Base64, ROC AUC, etc.	1.17
☐	class	Functions for Classification	7.3-10
☐	classInt	Choose univariate class intervals	0.1-21
☐	cluster	Cluster Analysis Extended Rousseeuw et al.	1.15.2
☐	clusterGeneration	random cluster generation (with specified degree of separation)	1.3.1
☐	coda	Output analysis and diagnostics for MCMC	0.16-1
☐	codetools	Code Analysis Tools for R	0.2-8
☐	colorspace	Color Space Manipulation	1.2-4
☐	compiler	The R Compiler Package	3.1.0
☐	data.table	Extension of data.frame	1.9.2
✓	datasets	The R Datasets Package	3.1.0

History to Console



```
##########################################################
#
#  An Example Script for RStudio
#
#  By: A. Michelle Lawing
#  Date: July 1, 2014
#
##########################################################

#get presabs matrix for various spatial grains
presabs<-read.csv(file="/Users/michellelawing/Dropbox/Biome_Ecometrics_Small_Working_Group/files_nobats/
presabs250<-read.csv(file="/Users/michellelawing/Dropbox/Biome_Ecometrics_Small_Working_Group/files_noba

#get bioclim variables and transform for normality
bioclim<-read.csv(file="/Users/michellelawing/Dropbox/Biome_Ecometrics_Small_Working_Group/files_nobats/
bioclim[,13:20]<-log(bioclim[,13:20])
bioclim[sapply(bioclim,is.infinite)]<-NA
bioclim250<-bioclim[presabs[,1] %in% presabs250[,1] == TRUE,]
```

Console ~/Documents/

```
+     nullmodels[i,is.infinite(nullmodels[i,1:numOfPoints,1]),1]<-NA
+     nullmodels[i,is.infinite(nullmodels[i,1:numOfPoints,2]),2]<-NA
+     nullmodels[i,is.infinite(nullmodels[i,1:numOfPoints,3]),3]<-NA
+   }
+   LineUp=array(NA,dim=30)
+   for(i in 4:length(phylomodels[1,])){
+     temp=paste(c(pathNull,colnames(phylomodels)[i],"-FinalStats-DF.csv"),collapse="")
+     LineUp[i-3]=((pmatch(temp,files)+1)/2)
+   }
+   corMatrix<-array(NA, dim=c(30,5,19))
+   for(i in 1:30){
+     corMatrix[i,1:5,1:19]<-cor(cbind(nullmodels[LineUp[i],1:numOfPoints,1:3],phylomodels[1:numOfPoints,i+3],
(nullmodels[LineUp[i],1:numOfPoints,1]-spatialautomodels[1:numOfPoints,i+1])),bioclim[,2:20],use="pairwise.com
plete.obs")
+   }
+   return(corMatrix)
+ }
>
> |
```

History:

```
LineUp=array(NA,dim=30)
for(i in 4:length(phylomodels[1,])){
temp=paste(c(pathNull,colnames(phylomodels)[i],"-FinalStats-DF.csv"),collapse="")
LineUp[i-3]=((pmatch(temp,files)+1)/2)
}
corMatrix<-array(NA, dim=c(30,5,19))
for(i in 1:30){
corMatrix[i,1:5,1:19]<-cor(cbind(nullmodels[LineUp[i],1:numOfPoints,1:3],phylomodels[1:nu…
}
return(corMatrix)
}
```

par {graphics} R Documentation

Set or Query Graphical Parameters

Description

`par` can be used to set or query graphical parameters. Parameters can be set by specifying them as arguments to `par` in `tag = value` form, or by passing them as a list of tagged values.

Usage

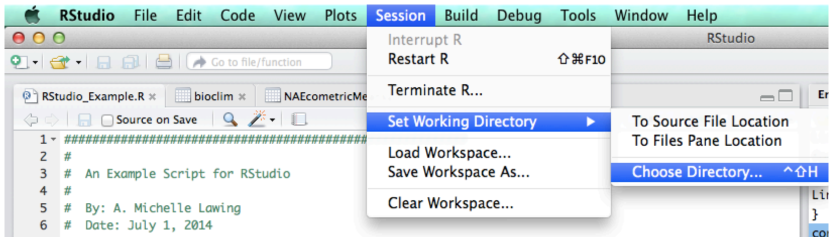
```
par(..., no.readonly = FALSE)

<highlevel plot> (\dots, <tag> = <value>)
```

Arguments

`...`	arguments in `tag = value` form, or a list of tagged values. The tags must come from the names of graphical parameters described in the 'Graphical Parameters' section.
`no.readonly`	logical; if `TRUE` and there are no other arguments, only parameters are returned which can be set by a subsequent `par()` call *on the same device.*

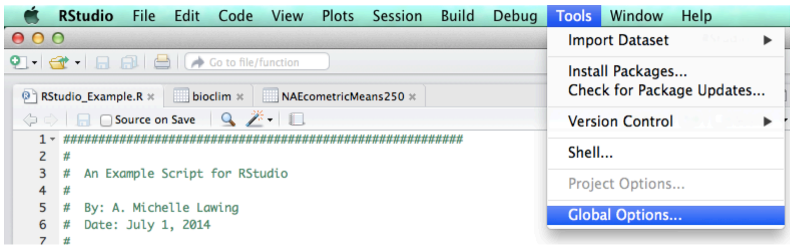
Details

Changing the Working Directory



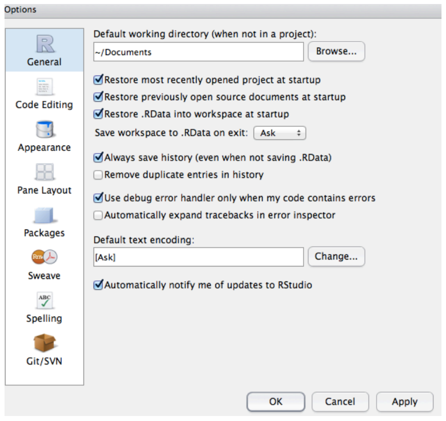
```
##############################################
#
#  An Example Script for RStudio
#
#  By: A. Michelle Lawing
#  Date: July 1, 2014
```

Set Default Directory


RStudio File Edit Code View Plots Session Build Debug Tools Window Help
Import Dataset
Install Packages...
Check for Package Updates...
Version Control
Shell...
Project Options...
Global Options...

```
##############################################################
#
#  An Example Script for RStudio
#
#  By: A. Michelle Lawing
#  Date: July 1, 2014
#
```

Set Default Directory



Default working directory (when not in a project):
~/Documents

✓ Restore most recently opened project at startup
✓ Restore previously open source documents at startup
✓ Restore .RData into workspace at startup
Save workspace to .RData on exit: Ask

✓ Always save history (even when not saving .RData)
☐ Remove duplicate entries in history

✓ Use debug error handler only when my code contains errors
☐ Automatically expand tracebacks in error inspector

Default text encoding:
[Ask]

✓ Automatically notify me of updates to RStudio

RStudio PCA Example

- Open RStudio_PCA.pdf
- Work through this document
- Copy or type commands into R (hint can also look at PCA.R script)

Session 2 – Introduction to the R environment continued

7. Wrap-up

Problems with Downloads?

- Git bash
- QGIS

Regression Example

OSOS EEB Summer Workshop

In the following exercise, the instructions and observations appear as normal text, the gray boxes show the r script RegressionModel.R, and the output appears directly after the script. Figures that are output from the script are also impeded in this document.

It is good to follow a consistent style when starting new scripts. The top of this script includes a header so you can keep track of your code.

```
##########################################################################
# RegressionModel.R
#
# This script runs through an example regression analysis
#
# Depends on: ipomopsis.txt
#
# created 11/1/2010 by M. Frazier
#
# last modified: 7/2/2014
#             by: A. Michelle Lawing
#
# More detailed description if necessary....
##########################################################################
```

First thing, set the working directory. This is the computer location where R looks for things. In our case, we want the working directory to be the "RegressionExample" folder. Replace the path in the code below with your path to the same folder. Use dir() to see what is inside.

```
setwd("/Users/michellelawing/Documents/RegressionExample") # set working directory
dir() # list files inside directory
```

```
## [1] "ipomopsis.txt"
```

Load data. We can read data from a website! But if this fails, the data are also in the directory folder. You can access the folder using the alternative to read in data below. After you read in the data, use some functions to check out data properties.

```
ipomopsis<- read.table(
  'http://www.unc.edu/courses/2007spring/enst/562/001/data/lab5/ipomopsis.txt',
  header=T,sep='\t')
#ipomopsis<- read.table("ipomopsis.txt", header=T,sep='\t') # Alternative to read in data

head(ipomopsis) # first 6 rows of dataframe
```

```
##    Root Fruit  Grazing
## 1 6.225 59.77 Ungrazed
## 2 6.487 60.98 Ungrazed
## 3 4.919 14.73 Ungrazed
## 4 5.130 19.28 Ungrazed
## 5 5.417 34.25 Ungrazed
## 6 5.359 35.53 Ungrazed
```

```r
dim(ipomopsis)  # dimensions: number of rows, columns
```

```
## [1] 40  3
```

```r
summary(ipomopsis) # summary
```

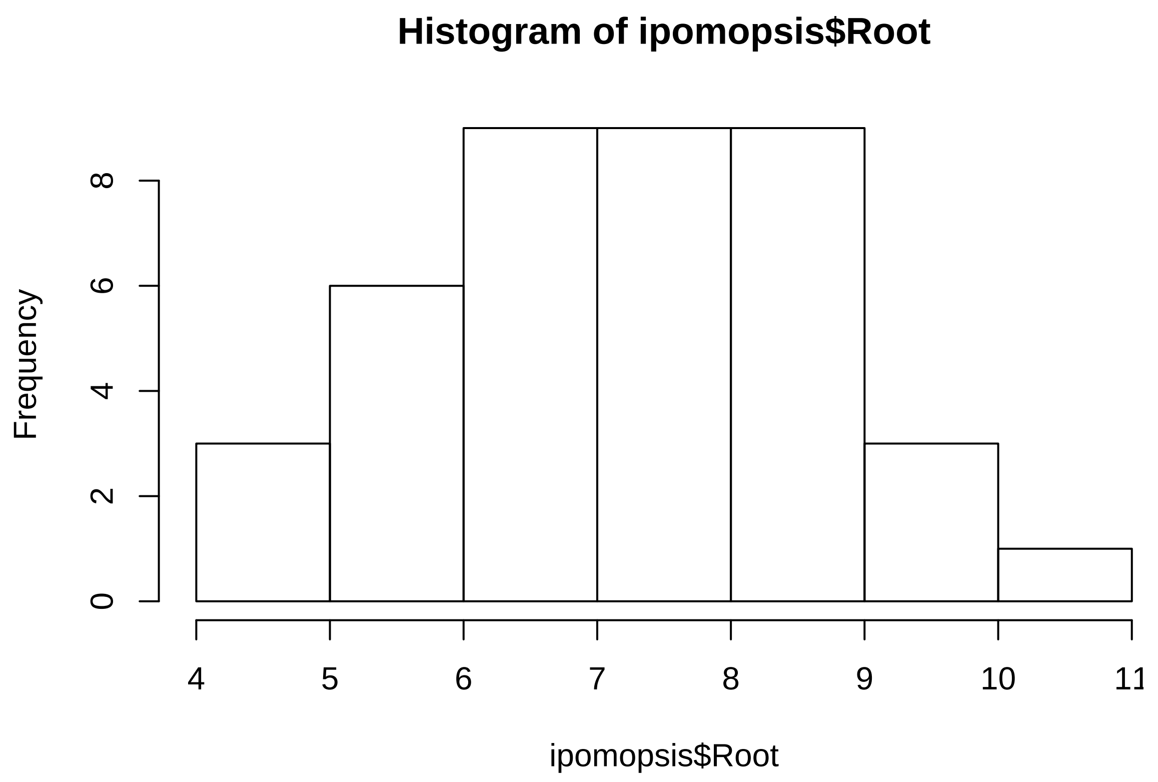
```
##       Root            Fruit           Grazing
##  Min.   : 4.43   Min.   : 14.7   Grazed  :20
##  1st Qu.: 6.08   1st Qu.: 41.1   Ungrazed:20
##  Median : 7.12   Median : 60.9
##  Mean   : 7.18   Mean   : 59.4
##  3rd Qu.: 8.51   3rd Qu.: 76.2
##  Max.   :10.25   Max.   :116.0
```

```r
lapply(ipomopsis, class) # variable classes
```

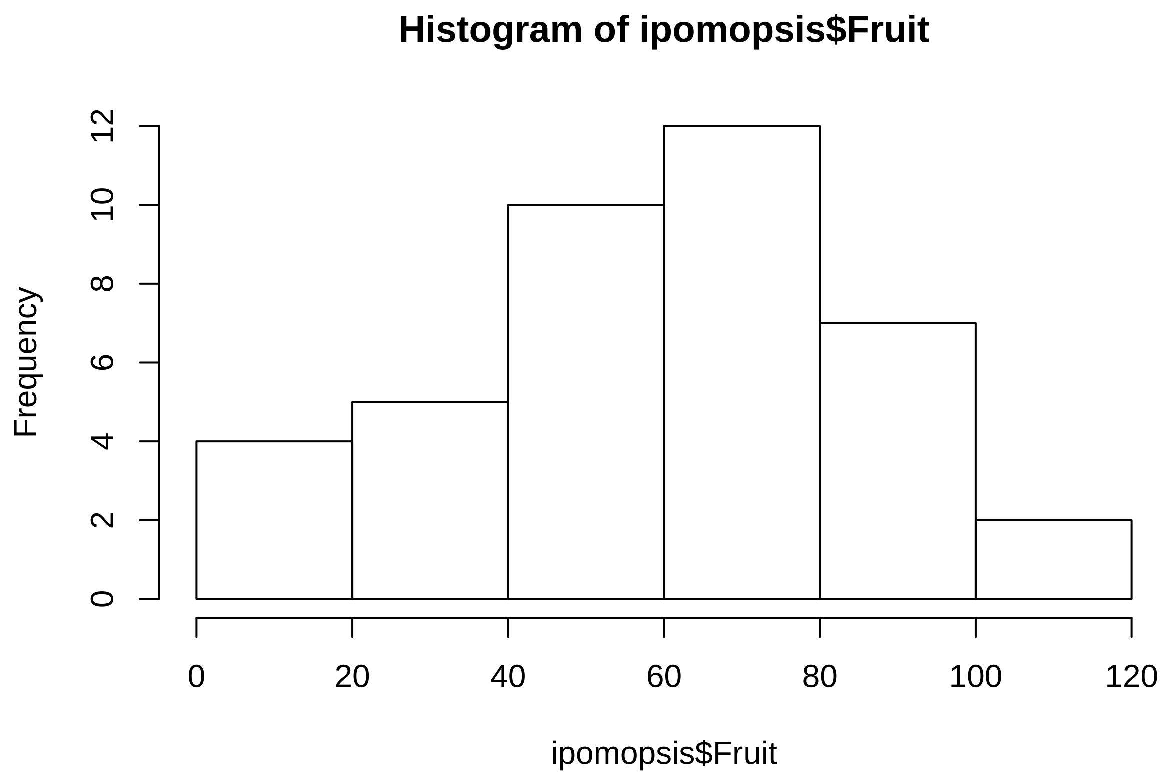
```
## $Root
## [1] "numeric"
##
## $Fruit
## [1] "numeric"
##
## $Grazing
## [1] "factor"
```

Perform some quick visual checks of the data.

```r
hist(ipomopsis$Root) #histogram
```

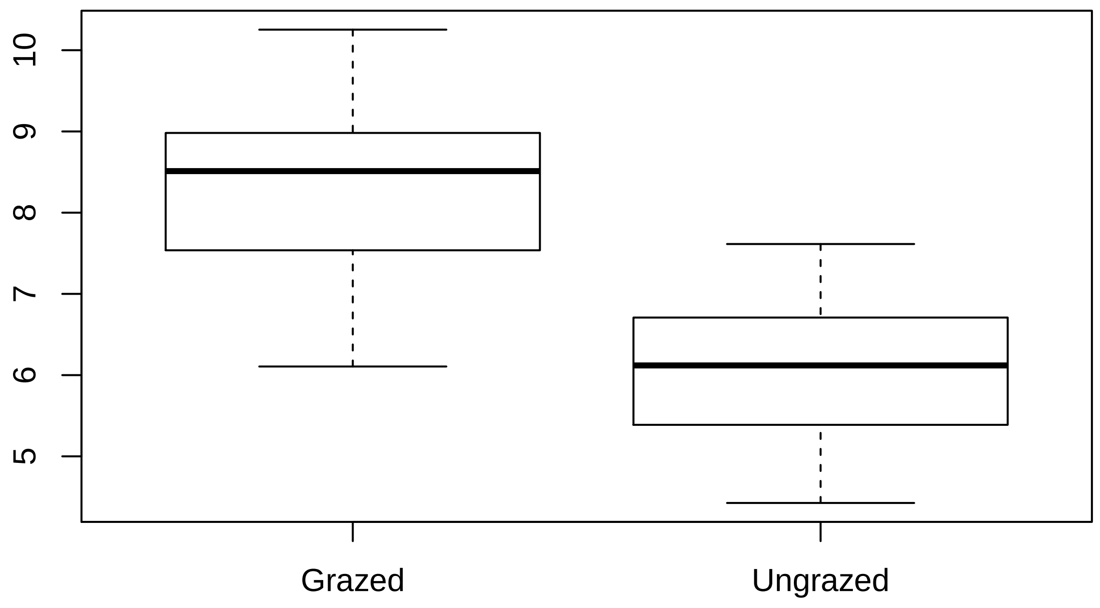


```r
hist(ipomopsis$Fruit) #histogram
```

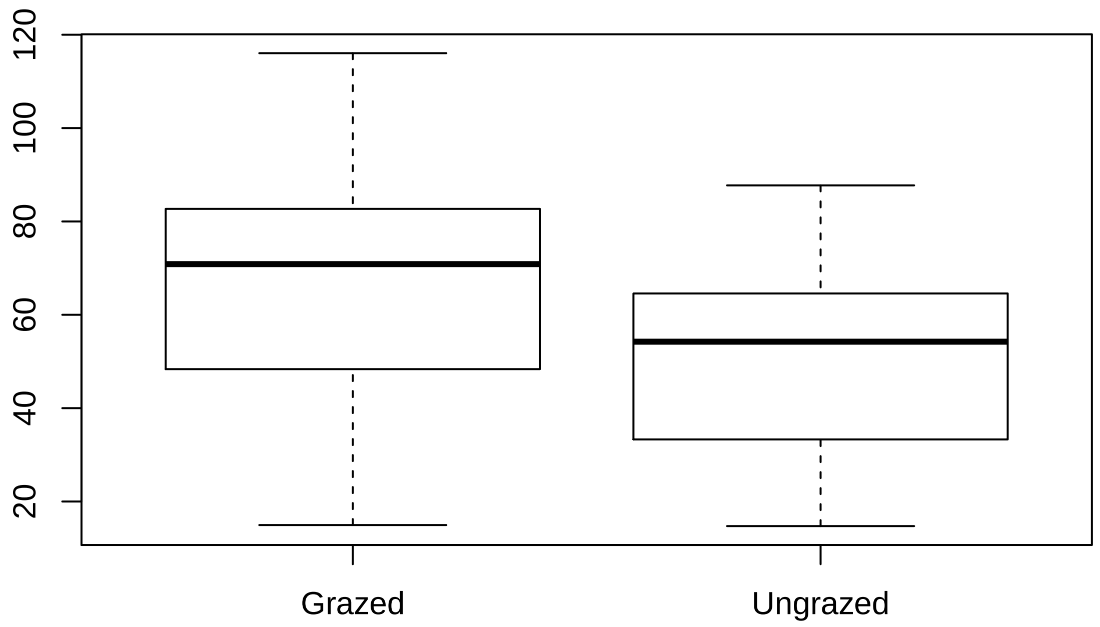


These variables look normal. Now we look at their relationship.

```r
plot(ipomopsis$Grazing, ipomopsis$Root)
```

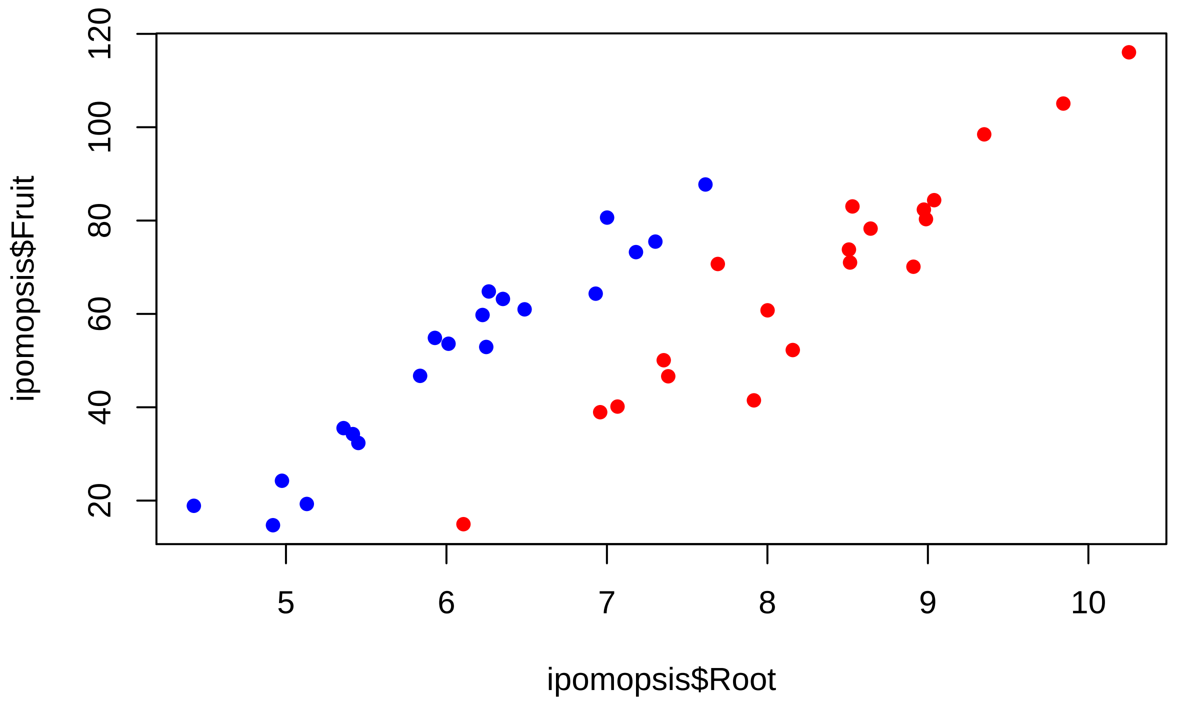


```r
plot(ipomopsis$Grazing, ipomopsis$Fruit)
```



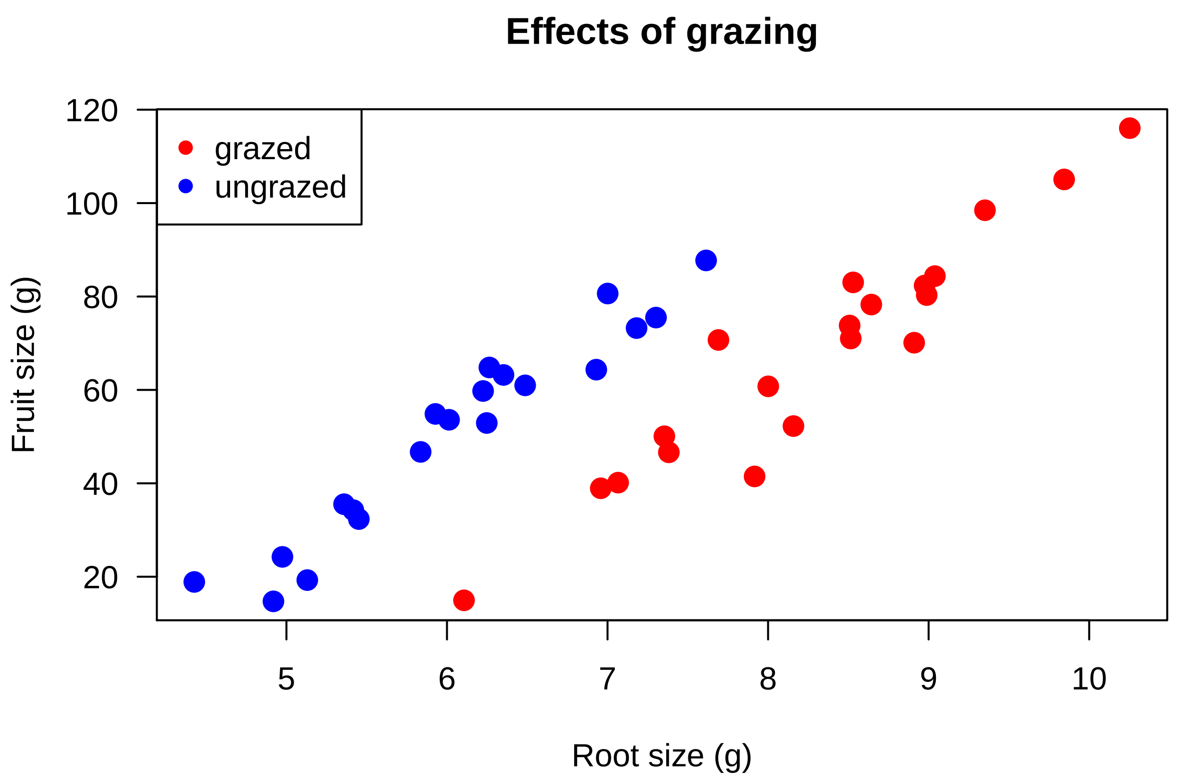
There doesn't appear to be much of an effect of grazing on Fruit size, but we will continue with the full analysis.

```
################################
## Plotting the point data
################################

plot(ipomopsis$Fruit ~ ipomopsis$Root,
     col=c("red", "blue")[ipomopsis$Grazing], #a little color coding
     pch=16) #pch changes the type of point
```



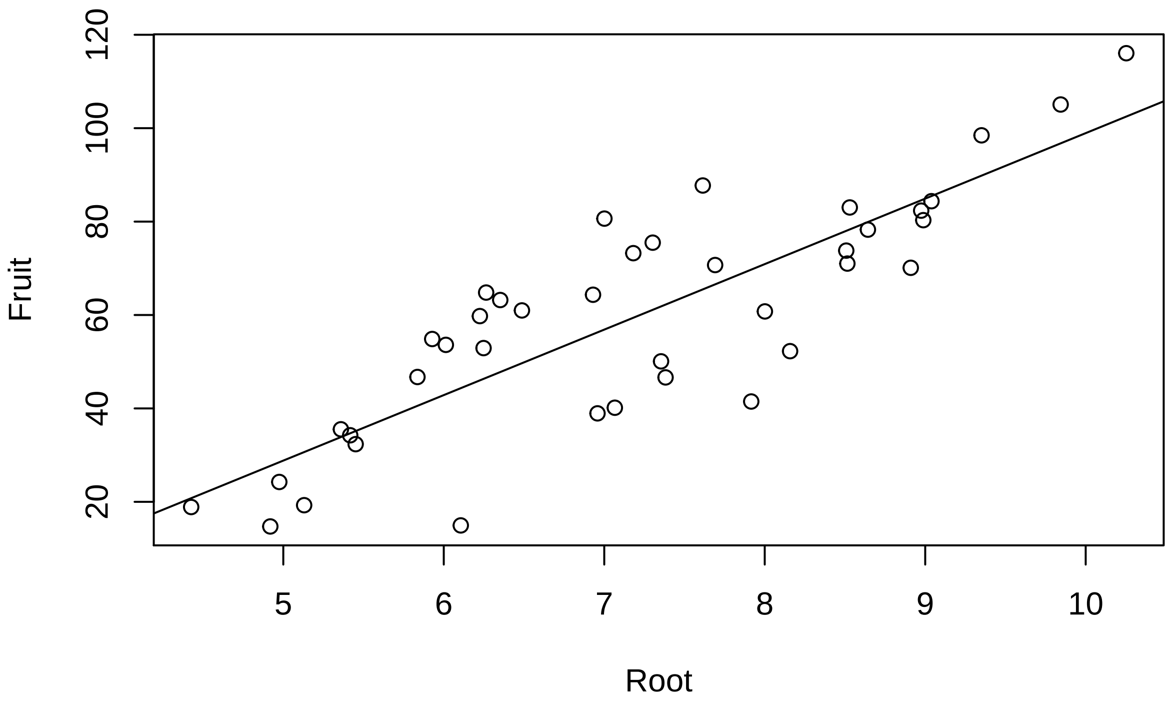
It now looks like grazing does have an effect. This looks like an important plot, so I will take the time to make it look good. We'll spend more time on making plots look good later on in the workshop. Add a legend to the plot.

```r
plot(ipomopsis$Fruit ~ ipomopsis$Root,
     col = c("red", "blue")[ipomopsis$Grazing], #a little color coding
     pch = 16,
     main = "Effects of grazing",
     ylab = "Fruit size (g)",
     xlab = "Root size (g)",
     las = 1, #las rotates the labels on the y axis
     cex = 1.5 #cex scales the point size up or down
     )
legend("topleft", legend=c("grazed", "ungrazed"), col=c("red", "blue"), pch=16)
```



Now we will us lm() to make linear models of the relationship.

```r
##################################################################
## Linear models using ordinary least squares regression models
##################################################################

mod1 <- lm(Fruit ~ Root, data=ipomopsis) #### Here is the simplest possible model
plot(Fruit ~ Root, data=ipomopsis)
abline(mod1) #this plots the regression model on the figure
```



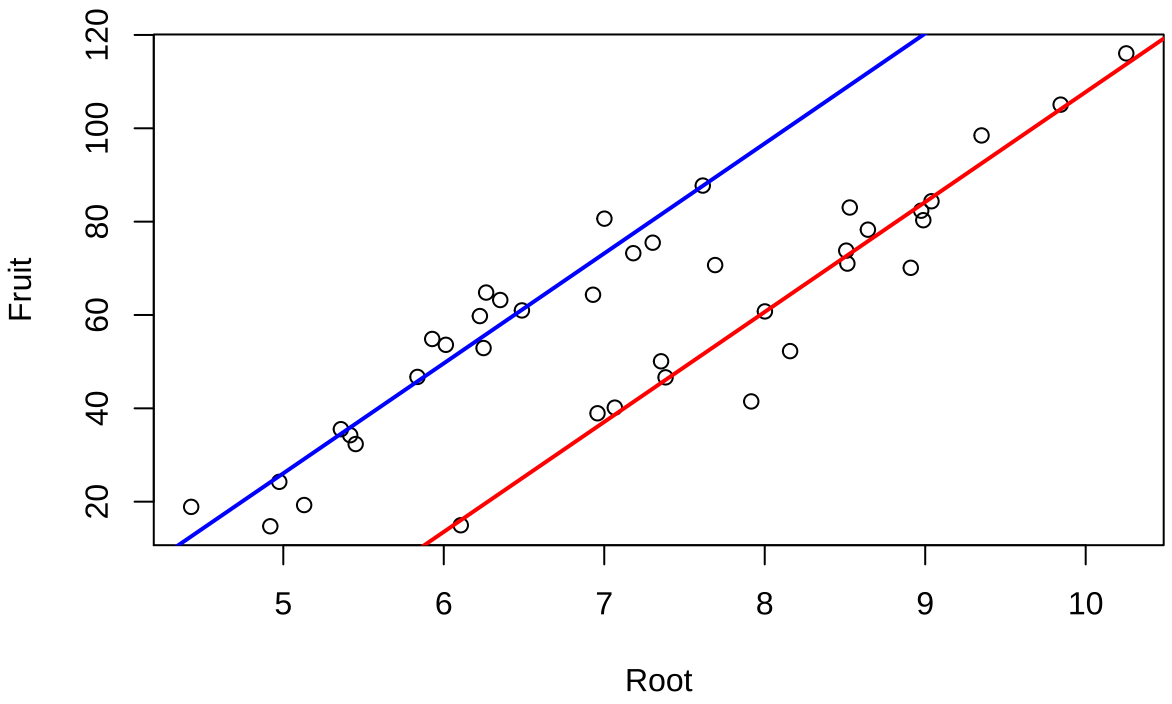
```r
summary(mod1)
```

```
##
## Call:
## lm(formula = Fruit ~ Root, data = ipomopsis)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -29.384 -10.445  -0.757  10.761  23.756
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   -41.29      10.72   -3.85  0.00044 ***
## Root           14.02       1.46    9.58  1.1e-11 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 13.5 on 38 degrees of freedom
## Multiple R-squared:  0.707,  Adjusted R-squared:   0.7
## F-statistic: 91.8 on 1 and 38 DF,  p-value: 1.1e-11
```

Make a model with varying intercept. There is a strong positive correlation between root size and fruit size. We can also use abline(intercept, slope) to plot the model fit. In this case, the "grazed" category is the default category that the model fits (i.e. dummy variable is 1 for this category). The "ungrazed" category is compared to the "grazed" category. We have to calculate the intercept for the "ungrazed" category by adding the intercept and Grazing Estimate. Notice we use the same slope for the grazed and ungrazed categories.

```r
mod2 <- lm(Fruit ~ Root + Grazing, data=ipomopsis) #varying intercepts
plot(Fruit ~ Root, data=ipomopsis)
# add line for the "grazed" category
```

```
abline(mod2$coefficients[1], mod2$coefficients[2], col="red", lwd=2)
# add line for the "ungrazed" category
abline((mod2$coefficients[1] + mod2$coefficients[3]), mod2$coefficients[2],
       col="blue", lwd=2)
```



```
summary(mod2)
```

```
##
## Call:
## lm(formula = Fruit ~ Root + Grazing, data = ipomopsis)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -17.192  -2.822   0.322   3.914  17.329
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)       -127.83       9.66   -13.2  1.3e-15 ***
## Root                23.56       1.15    20.5  < 2e-16 ***
## GrazingUngrazed     36.10       3.36    10.8  6.1e-13 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.75 on 37 degrees of freedom
## Multiple R-squared:  0.929,  Adjusted R-squared:  0.925
## F-statistic:  242 on 2 and 37 DF,  p-value: <2e-16
```

Now run the full model with interaction effect.

```
mod3 <- lm(Fruit ~ Root*Grazing, data=ipomopsis) ### Interaction effect
summary(mod3)
```

```
##
## Call:
## lm(formula = Fruit ~ Root * Grazing, data = ipomopsis)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -17.318  -2.832   0.125   3.851  17.131
##
## Coefficients:
##                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)           -125.173     12.811   -9.77  1.2e-11 ***
## Root                    23.240      1.531   15.18  < 2e-16 ***
## GrazingUngrazed         30.806     16.842    1.83    0.076 .
## Root:GrazingUngrazed     0.756      2.354    0.32    0.750
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.83 on 36 degrees of freedom
## Multiple R-squared:  0.929,  Adjusted R-squared:  0.923
## F-statistic:  158 on 3 and 36 DF,  p-value: <2e-16
```

The fact that the p-value for the Root:Grazing interaction is not significant suggests that this more complex model is not necessary. Rather than using p-values to determine which model is better, we can perform model selection by using AIC(). The model with the lowest AIC value is considered the best fit. According to this, mod2 is the best one.

```
###############################
#### Model selection
###############################

AIC(mod1, mod2, mod3) #model selection
```

```
##      df   AIC
## mod1  3 325.8
## mod2  4 271.1
## mod3  5 273.0
```

Another way to compare models is to use the anova function.

```
anova(mod1, mod2, mod3) #model selection
```

```
## Analysis of Variance Table
##
## Model 1: Fruit ~ Root
## Model 2: Fruit ~ Root + Grazing
## Model 3: Fruit ~ Root * Grazing
##   Res.Df  RSS Df Sum of Sq     F  Pr(>F)
## 1     38 6949
## 2     37 1684  1      5264 112.8 1.2e-12 ***
## 3     36 1680  1         5   0.1    0.75
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Mod2 is significantly better than mod1, mod 3 is NOT significantly better than mod1 or mod2. There is more information in the regression model that we can access.

```
################################################################
## Peaking into the linear model
################################################################

names(mod2) # here is a list of some of them
```

```
##  [1] "coefficients"  "residuals"     "effects"       "rank"
##  [5] "fitted.values" "assign"        "qr"            "df.residual"
##  [9] "contrasts"     "xlevels"       "call"          "terms"
## [13] "model"
```

```
mod2$coefficients # they can be accessed like this
```

```
##     (Intercept)            Root GrazingUngrazed
##         -127.83           23.56           36.10
```

One thing that can be interesting is to look at what the model predicts for each sample. For example, the first sample has a Root size of 6.225 and is Ungrazed. How big should the fruit be based on the model?

```
mod2$fitted.values # provides a predicted fruit size for each sample
```

```
##      1      2      3      4      5      6      7      8      9     10
##  54.94  61.11  24.17  29.14  35.90  34.53  87.66  57.93  25.49  71.55
##     11     12     13     14     15     16     17     18     19     20
##  55.48  36.70  49.94  47.94  55.85  77.46  73.22  12.55  80.31  45.77
##     21     22     23     24     25     26     27     28     29     30
## 113.73  36.10  60.67  85.13  82.09  16.03  53.37  83.93  83.62 104.10
##     31     32     33     34     35     36     37     38     39     40
##  72.62  45.43  75.80  58.67  92.48  38.65  64.37  46.09  72.78  73.14
```

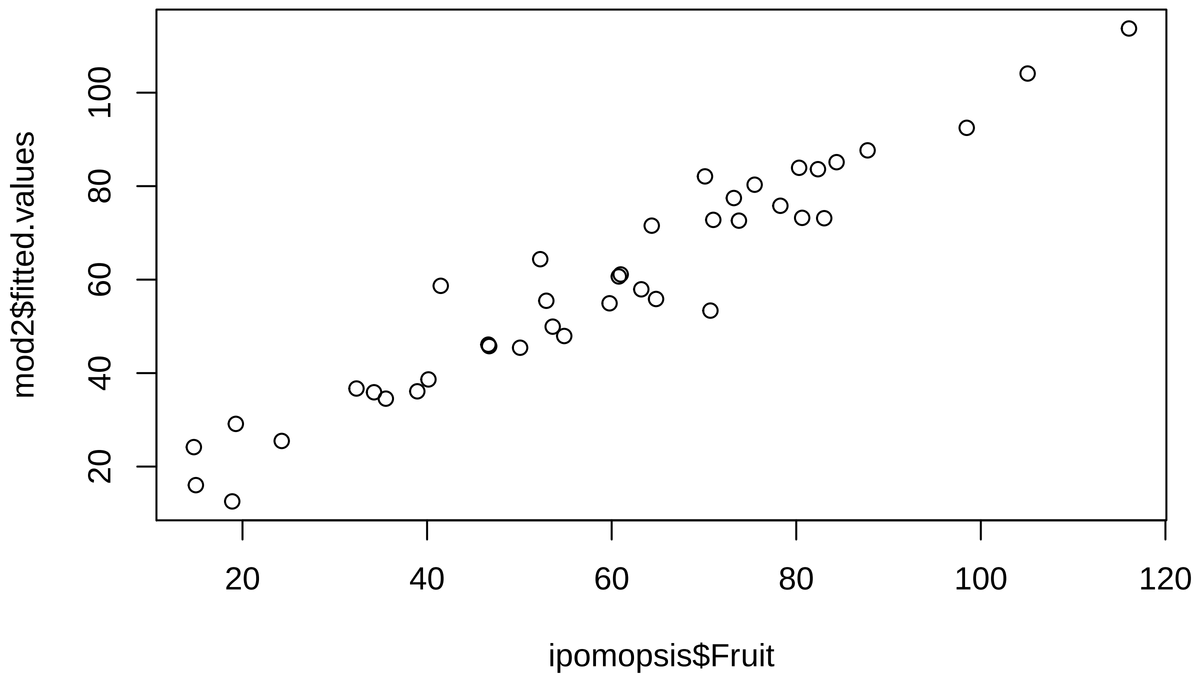
According to the model, the predicted size of fruit for that sample was 54.935. The observed weight was 59.77, so the model is a little low in that case. The difference between the observed and predicted value is the "residual".

```
mod2$residuals
```

```
##        1        2        3        4        5        6        7
##  4.83482 -0.12792 -9.43576 -9.85693 -1.64867  0.99782  0.06991
##        8        9       10       11       12       13       14
##  5.28269 -1.23512 -7.20502 -2.55707 -4.34971  3.66955  6.92215
##       15       16       17       18       19       20       21
##  8.95597 -4.21859  7.42222  6.33934 -4.81936  0.95967  2.31819
##       22       23       24       25       26       27       28
##   2.83855   0.09542  -0.75991 -11.98066  -1.07829  17.32903  -3.61835
##        29        30        31        32        33        34        35
##  -1.27207   0.97425   1.17047   4.64877   2.47987 -17.19198   5.98935
##        36        37        38        39        40
##   1.50406 -12.11351   0.54909  -1.77445   9.89215
```

It can be useful to compare the predicted and observed values for the data.

```r
plot(ipomopsis$Fruit, mod2$fitted.values) # compare the predicted and observed values
```



It looks like the model does a good job. Here are some plots to evaluate the model assumptions.

```r
plot(mod2,which=1) # 1 is residuals vs. fitted values
```

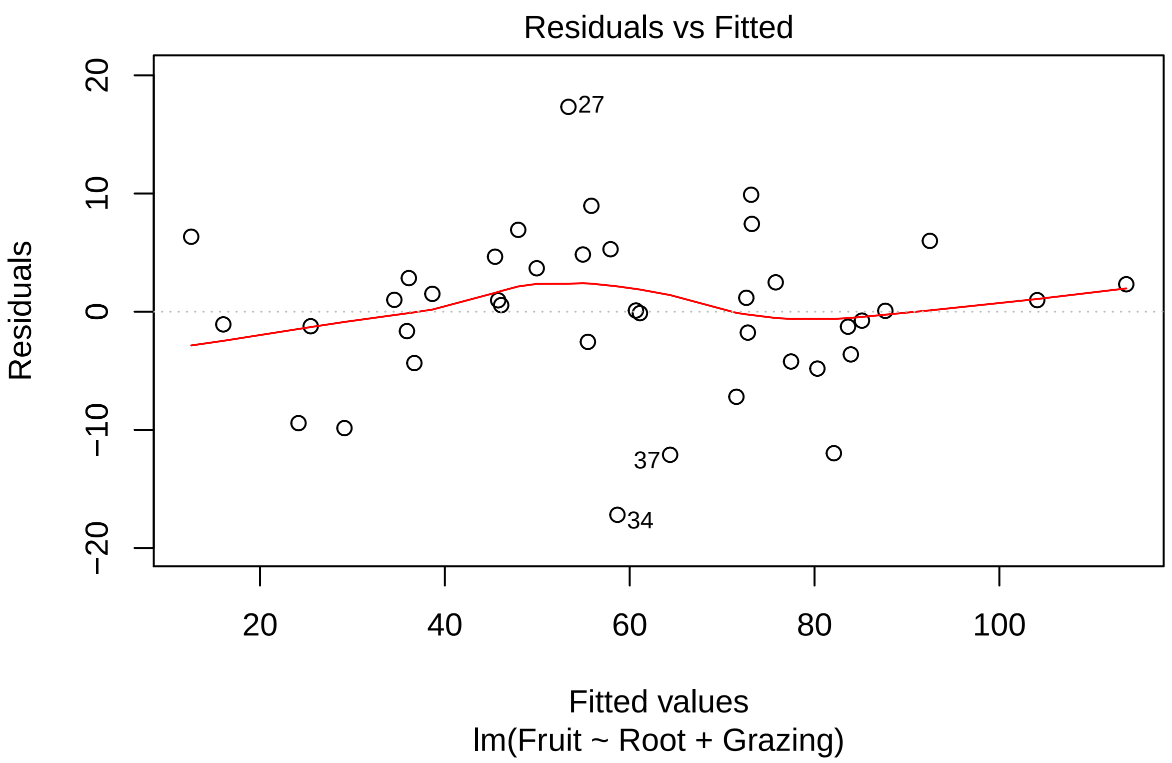


Fig. 1: A plot of residuals vs. fitted values (predicted values). Ideally, these points will be symmetric around the zero line. There may be a pattern that suggests we should be fitting a curved line.

```
plot(mod2,which=2) # 2 is Normal Q-Q plot
```

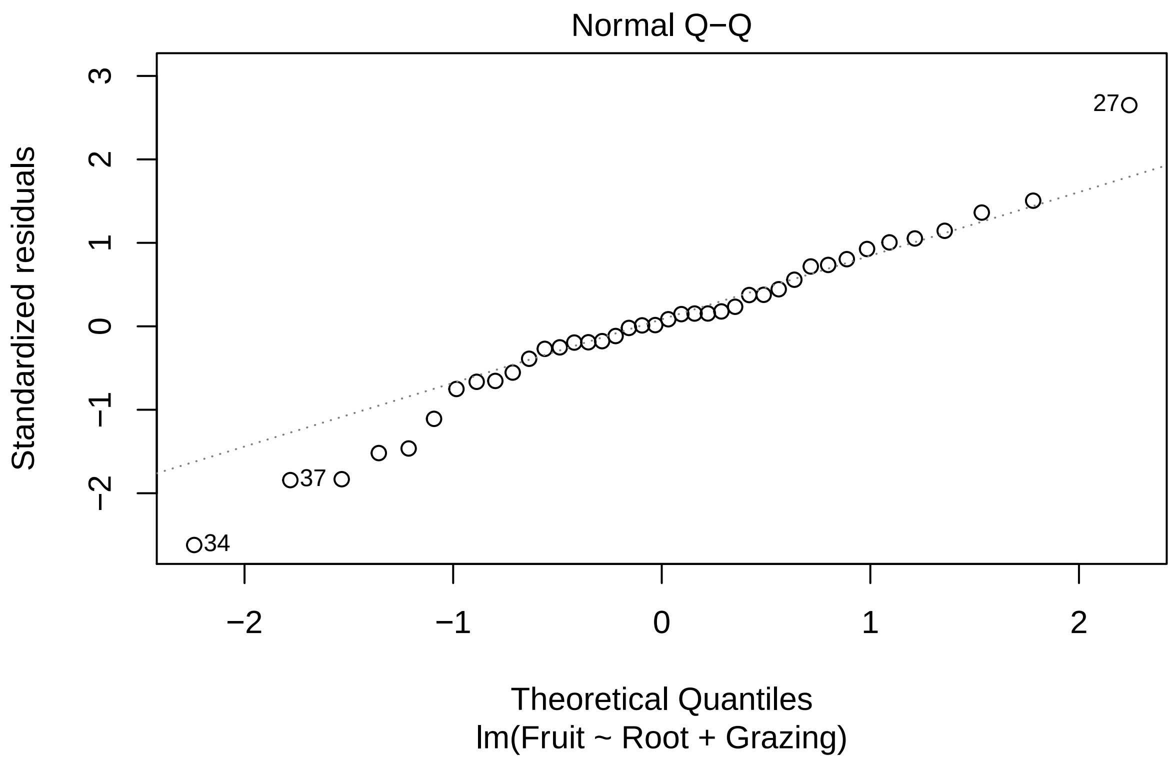


Fig. 2: Normal Q-Q plot. If residuals are from a normal distribution the points should lie close to the line.

```r
plot(mod2,which=3) # 3 is square root of standardized residuals
```

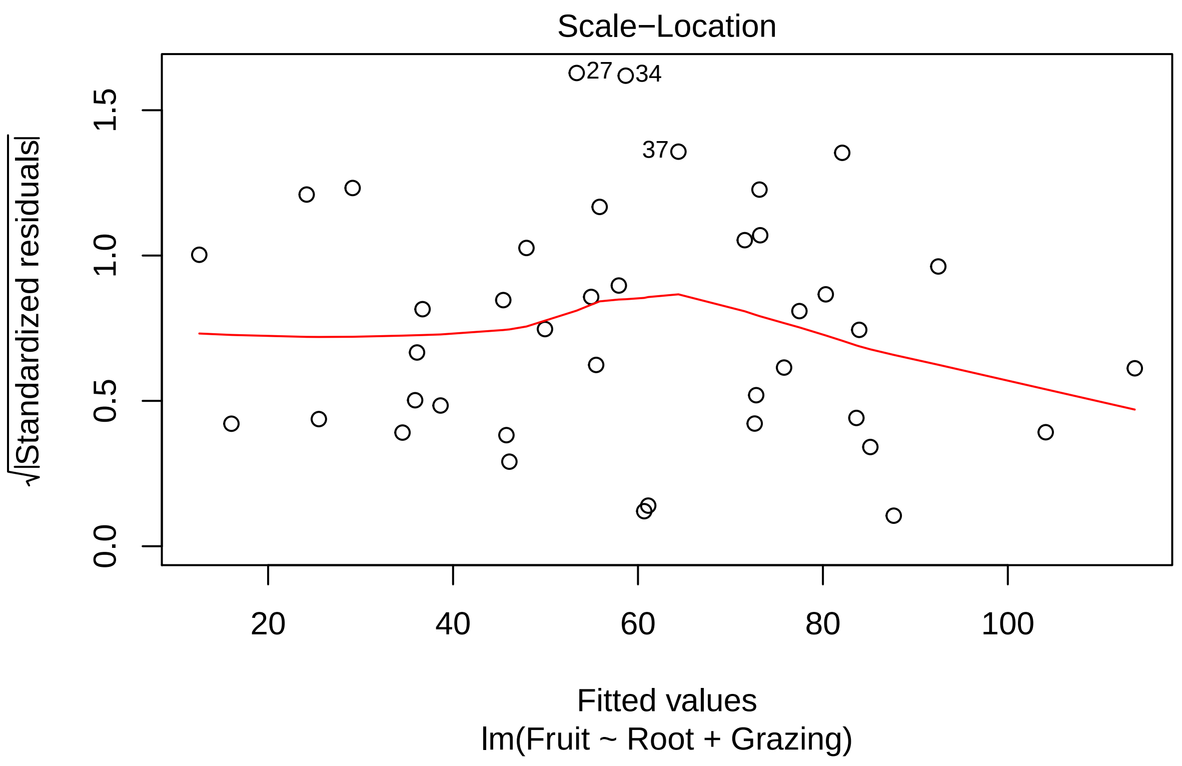


Fig. 3: Similar to Fig. 1, but the square root of standardized residuals is used. There are theoretical reasons why this is better for evaluating whether the variance is constant. A funnel shape to the points might indicate heteroscedasticity (unequal variance).

```r
plot(mod2,which=4) # 4 is Cook's distance
```

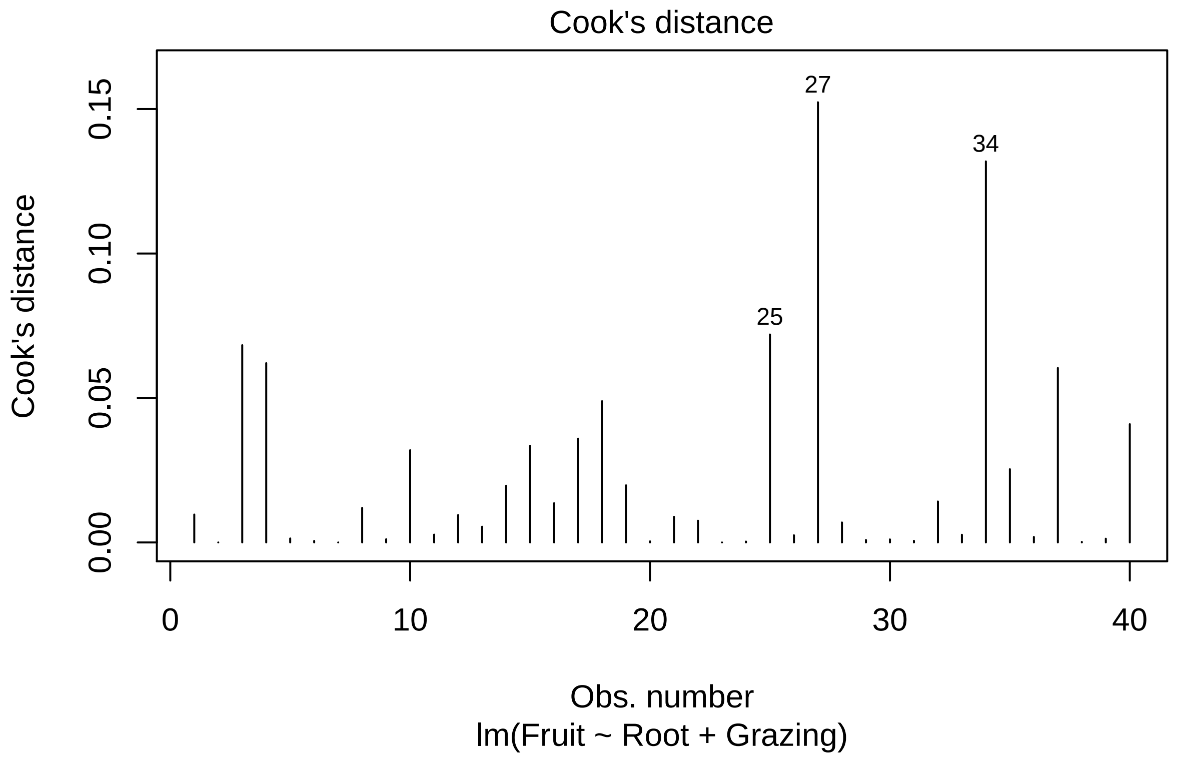


Fig. 4: Identifies residuals that are influential in driving the regression line. Cook's distance measures the extent to which the relationship would change if the point were omitted. Values >1 are considered influential. In this case, everything is less than 0.16. Note: outliers may or may not be influential. Influential outliers can be problematic.

To get confidence intervals around predictions, use confint().

```r
confint(mod2)
```

```
##                     2.5 %  97.5 %
## (Intercept)      -147.41 -108.25
## Root               21.23   25.89
## GrazingUngrazed    29.30   42.91
```

RStudio PCA Example

OSOS EEB Summer Workshop

In the following exercise, the instructions and observations appear as normal text, the gray boxes show the r script in PCA.R, and the output appears directly after the script. Figures that are output from the script are also impeded in this document.

```
###########################################################################
# PCA.R
#
# This script runs through an example ordination using PCA and points out some basics for
# first time users in RStudio.
#
# Depends on: devtools
#             ggbiplot
#
# created 11/28/2013 by Thiago G. Martins
#
# last modified: 7/2/2014
#            by: A. Michelle Lawing
#
# This code demonstrates how to apply and visualize PCA in R, specifically using RStudio.
# There are many packages and functions that can apply PCA in R. In this exercise we will
# use function prcomp from the stats package. We will also visualize PCA in R using Base R
# graphics. However, the recommended visualization function for PCA is ggbiplot, which is
# implemented by Vince Q. Vu and available on github.
#
###########################################################################
```

Set the working directory using the "Session" tab at the top of RStudio. Click on "set working directory"" and then "choose directory". Scroll to the folder RStudioPCA.

```
#set working dir
```

We will use the classic iris dataset. The data contain four continuous variables which correspond to physical measures of flowers and a categorical variable describing the flowers' species.

```
data(iris) # load data
head(iris) # have a look
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

Notice in the environment window the iris dataset is now loaded. Click on it in that window and see the data tab pop up in the script window. This tabular form is similar to the visualization of the data you see in excel.

Since skewness and the magnitude of the variables influence PCs, it is good practice to apply skewness transformation, center and scale the variables prior to the application of PCA. Therefore, we apply a log transformation to the variables before ordination. We also assign the categorical variable to its own factor.

```r
log.ir <- log(iris[, 1:4]) # log transform
ir.species <- iris[, 5] # put species in their own variable
```

In the environment window you will now see the new data.frame called log.ir and the new list (ir.pca) and factor (ir.species) under the Values heading. Those are not interactive like the data.frames under data (i.e., you cannot click on them).

Next we will apply PCA to the four continuous variables and use the categorical variable to visualize the PCs later. Set center and scale. equal to TRUE in the call to prcomp to standardize the variables prior to the application of PCA. Click on the Help tab and type in prcomp in the search bar at the top. Look at the help page that pops up. You can see that there are many usages of the prcomp function. We will not enter a formula, we will enter in a data matix in the form of a data.frame. Scroll down and look at all the arguments that you can pass to prcomp. You will see that two of those are the center and scale. arguments that we will change from their default values.

```r
# apply PCA - scale. = TRUE is highly
# advisable, but default is FALSE.
ir.pca <- prcomp(log.ir, center = TRUE, scale. = TRUE)
```

Analyzing the results

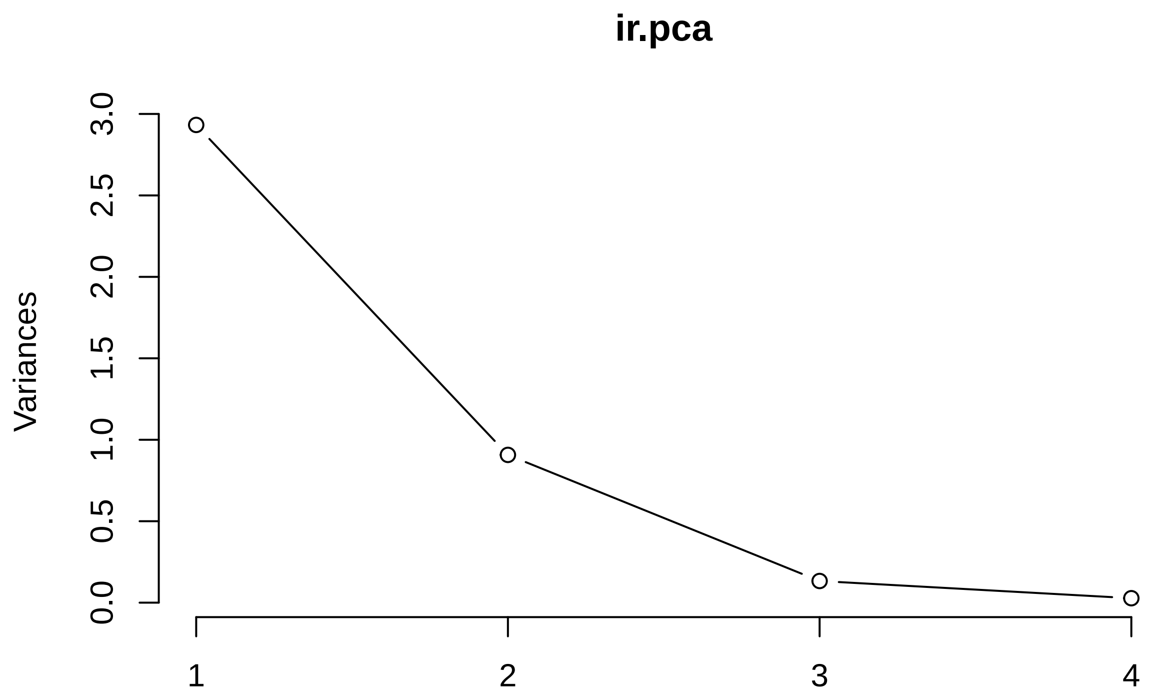
The prcomp function returns an object of class prcomp, which have some methods available. The print method returns the standard deviation of each of the four PCs, and their rotation (or loadings), which are the coefficients of the linear combinations of the continuous variables.

```r
# print method
print(ir.pca)
```

```
## Standard deviations:
## [1] 1.7125 0.9524 0.3647 0.1657
##
## Rotation:
##                   PC1      PC2     PC3      PC4
## Sepal.Length  0.5038 -0.45500  0.7089  0.19148
## Sepal.Width  -0.3024 -0.88914 -0.3312 -0.09125
## Petal.Length  0.5768 -0.03379 -0.2193 -0.78619
## Petal.Width   0.5675 -0.03546 -0.5829  0.58045
```

The plot method returns a plot of the variances (y-axis) associated with the PCs (x-axis). The Figure below is useful to decide how many PCs to retain for further analysis. In this simple case with only 4 PCs this is not a hard task and we can see that the first two PCs explain most of the variability in the data.

```r
# plot method
plot(ir.pca, type = "l")
```



Notice the plot that pops up in the plots window. If you scroll over the background space between the console and plot windows you will see the cursor turn to arrows indicating that you can change the size of the windows. Click on that space and drag the plots window to lengthen from its current size. You will see that the plot resizes depending on the size of the window. Keep this in mind, because when you save out your graphics, you will need to check their actual layout and not necessarily what you see in the plot tab. Despite this one drawback, there are many other features that make viewing plots easier in RStudio.

The summary method describes the importance of the PCs. The first row describes again the standard deviation associated with each PC. The second row shows the proportion of the variance in the data explained by each component while the third row describe the cumulative proportion of explained variance. We can see that the first two PCs account for more than 95% of the variance of the data.

```
# summary method
summary(ir.pca)
```

```
## Importance of components:
##                           PC1   PC2    PC3     PC4
## Standard deviation     1.712 0.952 0.3647 0.16568
## Proportion of Variance 0.733 0.227 0.0333 0.00686
## Cumulative Proportion  0.733 0.960 0.9931 1.00000
```

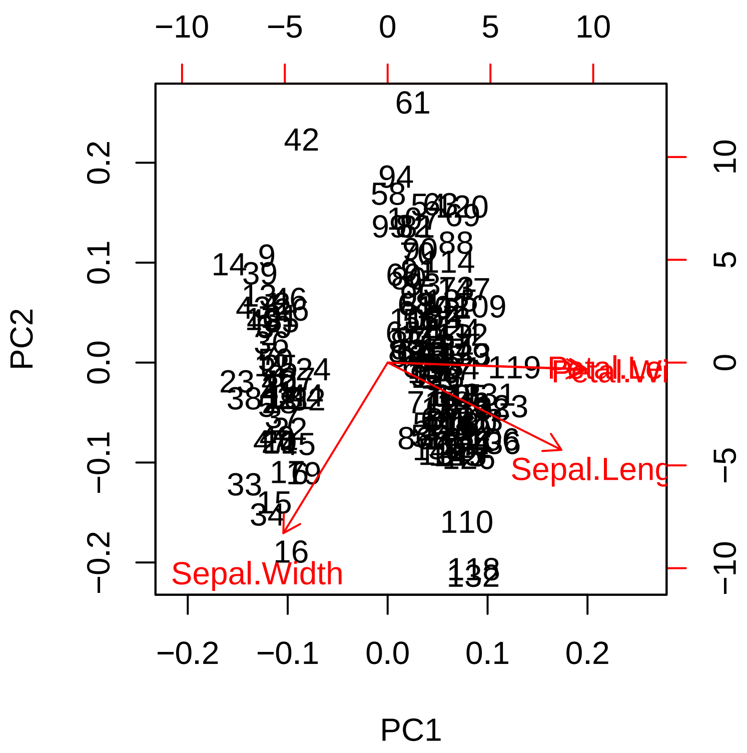
We can use the predict function if we observe new data and want to predict their PCs values. Just for illustration pretend the last two rows of the iris data has just arrived and we want to see what are their PC values. We will use the tail function to extract the last two rows of the iris data.

```
# Predict PCs
predict(ir.pca, newdata=tail(log.ir, 2))
```

```
##        PC1      PC2     PC3      PC4
## 149 1.0810 -1.01156 -0.7082 -0.06811
## 150 0.9712 -0.06159 -0.5009 -0.12412
```

Create a biplot of the results. The biplot function is used to make a plot of the first two PCs and the loadings of each variable on the PC axes. Type into the console ?biplot to see the help pages for the biplot function.

```r
# plot results
biplot(ir.pca)
```

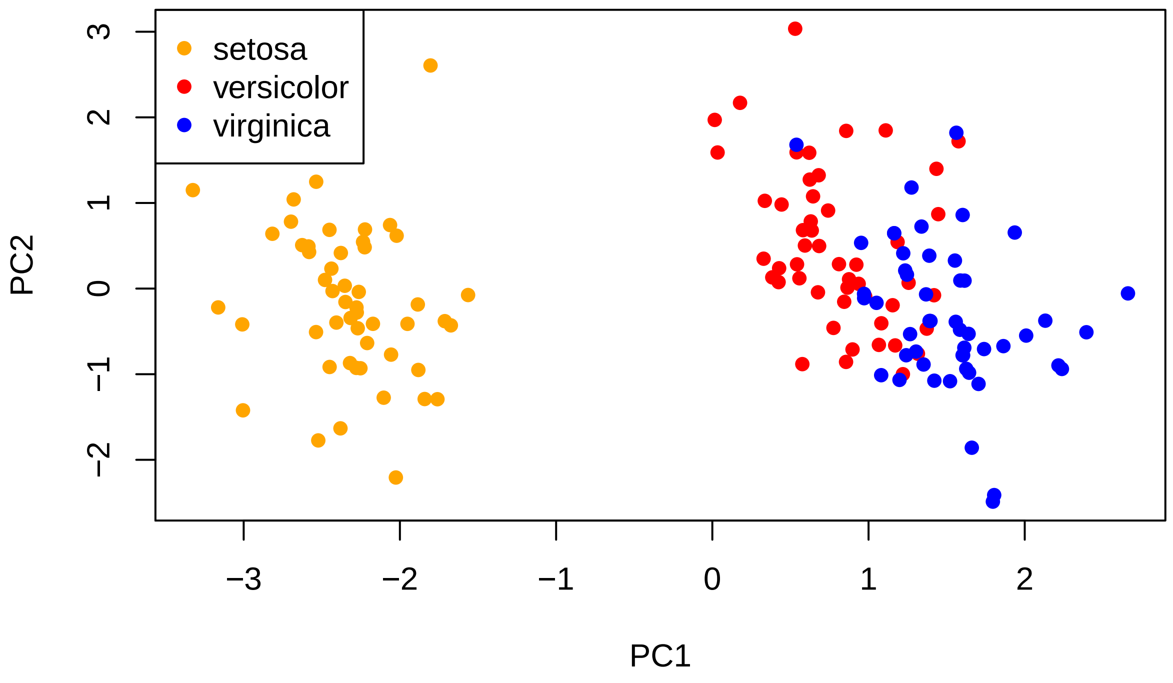


The numbers in the plotted above refer to the row entry in the data.frame. This is a very useful quick plot to looking at the properties of your data. However, you can make a prettier plot on your own. The PC scores are loaded in the result list. Check out the head of the PC scores. Plot PC1 and PC2, but first make a vector of colors that correspond to the species identity in each row. To select the row that corresponds to each species name we subset the new vector ir.color

```r
# see the first six rows of PC scores
head(ir.pca$x)
```

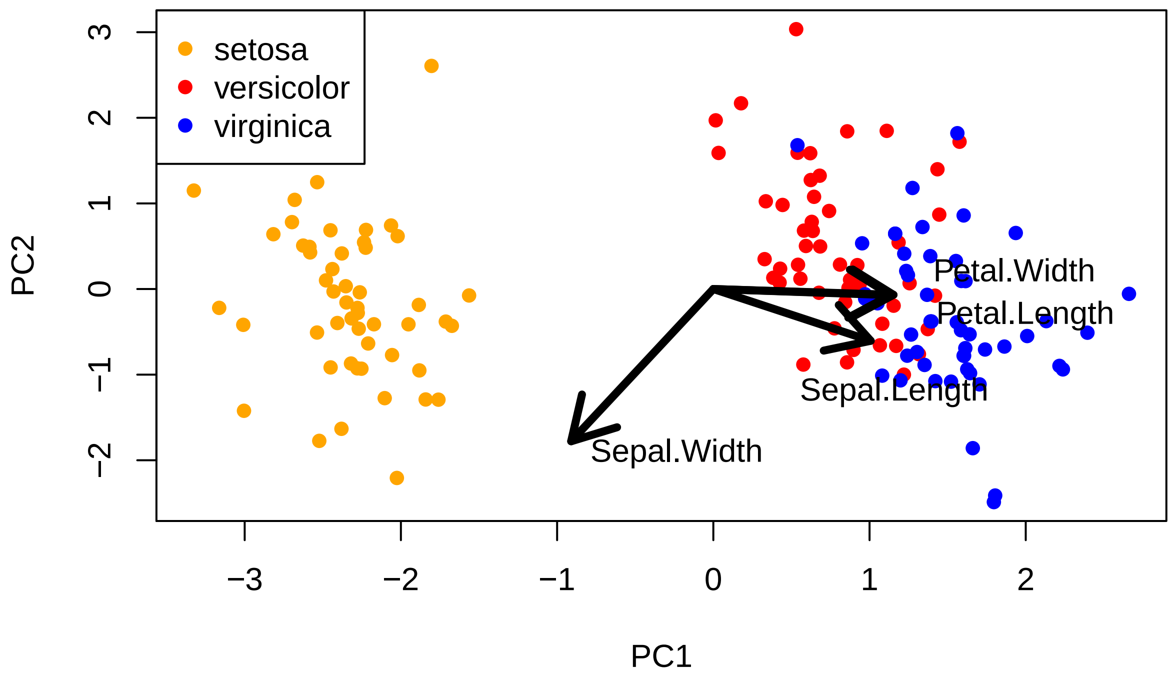
```
##          PC1     PC2      PC3       PC4
## [1,] -2.407 -0.3970  0.19396  0.004779
## [2,] -2.224  0.6902  0.35000  0.048868
## [3,] -2.581  0.4275  0.01890  0.049910
## [4,] -2.451  0.6860 -0.06875 -0.149646
## [5,] -2.537 -0.5083  0.02932 -0.040048
## [6,] -1.841 -1.2899 -0.25277  0.163891
```

```r
# vector of colors
# make a new variable with the same length as ir.species
ir.color<-array(NA, dim = length(ir.species))
# set all of the same species values equal to a color
ir.color[ir.species==unique(ir.species)[1]]<-"orange"
ir.color[ir.species==unique(ir.species)[2]]<-"red"
ir.color[ir.species==unique(ir.species)[3]]<-"blue"

# plot PC1 and PC2
plot(ir.pca$x[, 1:2], pch = 16, col = ir.color)
# add a legend
legend("topleft", legend = unique(ir.species), col = unique(ir.color),
       pch = 16)
```



We can see with the color coding, that setosa differentiates from the other two species on PC1. Plot the rotated variable axes on the PC plot and add the text for the variable names.

```r
# plot PC1 and PC2
plot(ir.pca$x[, 1:2], pch = 16, col = ir.color)

# add a legend
legend("topleft", legend = unique(ir.species), col = unique(ir.color), pch = 16)

# rotate data
rot <- ir.pca$rotation
arrows(0, 0, 2 * rot[1, 1], 2 * rot[2, 1], lwd = 4)
arrows(0, 0, 2 * rot[1, 2], 2 * rot[2, 2], lwd = 4)
arrows(0, 0, 2 * rot[3, 1], 2 * rot[3, 2], lwd = 4)
arrows(0, 0, 2 * rot[4, 1], 2 * rot[4, 2], lwd = 4)

# add text
text(2 * rot[1, 1] + 0.15, 2 * rot[2, 1] - 0.25, rownames(rot)[1], pos = 1, cex = 1)
text(2 * rot[1, 2], 2 * rot[2, 2] - 0.15, rownames(rot)[2], pos = 4, cex = 1)
text(2 * rot[3, 1] + 0.15, 2 * rot[3, 2] - 0.25, rownames(rot)[3], pos = 4, cex = 1)
text(2 * rot[4, 1] + 0.15, 2 * rot[4, 2] + 0.25, rownames(rot)[4], pos = 4, cex = 1)
```



This plot is okay, but we can still do better by installing a package. For the final example below, we need to install devtools. Click on the Packages tab and then click install. Type in devtools, make sure the box is checked for install dependencies, and then click install. Now scroll down to devtools in the packages tab and check the box next to it. Notice in the console that the library function was invoked by checking the box next to the package that you want to use.

```
# install devtools and load library
```

The package that we will use to make very nice biplots in R is in github, so we use the install_github function, then scroll down to the ggbiplot name under the packages tab and check the box. If you do not see the library installed under packages, hit the refresh packages button under the packages tab.

```
#install ggbiplot from github
install_github("ggbiplot", "vqv")
```

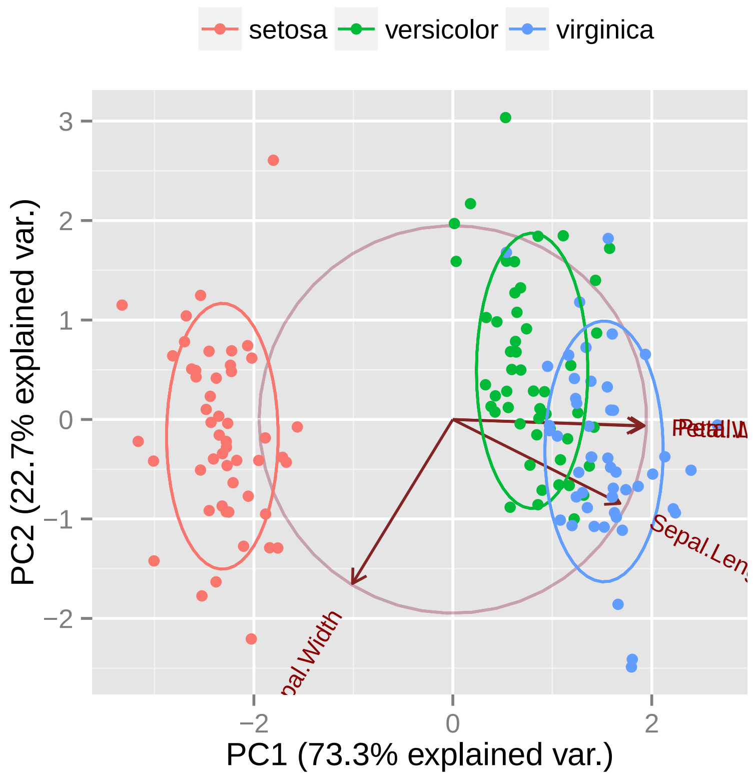
```
## Installing github repo ggbiplot/master from vqv
## Downloading master.zip from https://github.com/vqv/ggbiplot/archive/master.zip
## Installing package from /var/folders/p6/fjbczz_x3y97zvbbzccx4ddh0000gn/T//Rtmp9xolpQ/master.zip
## arguments 'minimized' and 'invisible' are for Windows only
## Installing ggbiplot
## '/Library/Frameworks/R.framework/Resources/bin/R' --vanilla CMD INSTALL  \
##   '/private/var/folders/p6/fjbczz_x3y97zvbbzccx4ddh0000gn/T/Rtmp9xolpQ/devtools10917b583a74/ggbiplot
##   --library='/Library/Frameworks/R.framework/Versions/3.1/Resources/library'  \
##   --install-tests
```

```
# load library ggbiplot
```

The code below produces a very nice biplot. It projects the data on the first two PCs. Other PCs can be chosen through the argument choices of the function. It colors each point according to the flowers' species and draws a Normal contour line with ellipse.prob probability (default to 68%) for each group. More info about ggbiplot can be obtained by the usual ?ggbiplot

Do not worry about the syntax for this plot today. You will learn the syntax for gg style plots tomorrow with Rebecca. In the meantime, enjoy!

```
g <- ggbiplot(ir.pca, obs.scale = 1, var.scale = 1,
              groups = ir.species, ellipse = TRUE,
              circle = TRUE)
g <- g + scale_color_discrete(name = '')
g <- g + theme(legend.direction = 'horizontal',
               legend.position = 'top')
print(g)
```



One last thing, click on your plots tab. Use the back arrow to scroll through the various plots you made within this example. All the plots are available for export. If you would like to export a plot, click on the export button and decide what format you would like to save the plot.

R – Beginners Reference Card

From OSOS: An EEB Workshop
College Station, TX 7.10.2014

Text and numbers in blue can/should be modified

Help

help.start()	starts an html version of the help menu
help(help.start)	documentation on function help.start, replace help.start with any function for which you need help
?help.start	same as above

Orientation

ls()	show objects in the current environment
dir()	show files in the current directory
getwd()	get working directory
setwd(“path”)	set working directory to path

Assign, remove, and load

x <- 1	assign x to 1
save(x, file=”x”)	save r object to a file named x in the current working directory
rm(x)	removes the assignment to x
load(“x”)	loads x from file named x
data(precip)	loads preloaded dataset precip
library(MASS)	loads installed package MASS

I/O

read.table(“file”)	open a table from file
read.csv(“file”)	open a csv from file
write.table(x, file=” file”)	saves a variable to a file

Data creation

c(1,2,3)	concatenates 1, 2, and 3 into variable
1:3	generates a sequence from 1 to 3
rep(1:3,2)	repeats 1 to 3 twice

Extracting Data

x <- c(1,2,3)	assign data to x for following extractions
x[1]	1st element
x[-1]	all but the 1st element
x[1:2]	elements 1 and 2
x[c(1,3)]	specific elements 1 and 3
x[x > 1]	elements GREATER THAN 1
x [x == 1]	elements EQUALS to 1
x [x > 1 & x < 3]	elements greater than 1 AND less than 3
x [x < 1 | x > 2]	elements less than 1 OR greater than 2

Converting Data

as.array(x)	converts x to class array
as.data.frame(x)	converts x to class data.frame
as.logical(x)	converts x to class logical
as.numeric(x)	converts x to class numeric
as.character(x)	converts x to class character
as.complex(x)	converts x to class complex

Variable Information

is.array(x)	checks class x, returns logical
is.data.frame(x)	checks class x, returns logical
is.logical(x)	checks class x, returns logical
is.numeric(x)	checks class x, returns logical
is.character(x)	checks class x, returns logical
is.complex(x)	checks class x, returns logical
is.na(x)	checks elements for na, returns logical
is.null(x)	checks elements for null, returns logical
length(x)	returns the length of x
dim(x)	returns the dimensions of x
dimnames(x)	returns the dimension names of x
nrow(x)	returns the number of rows of x
class(x)	get or set class x
unclass(x)	remove class x

Data Manipulation

which.max(x)	returns the index of the max element in x
which.min(x)	returns the index of the min element in x
rev(x)	reverses the order of x
sort(x)	sorts x
which(x == 2)	returns the indices where x equals 2
na.omit(x)	suppresses the elements with missing data
unique(x)	suppresses the duplicated elements
sample(x,2)	samples x twice

Basic Math

max(x)	returns the max value of x
min(x)	returns the min value of x
range(x)	returns the range of x
sum(x)	returns the sum of all elements of x
diff(x)	returns the lagged and iterated differences of x
prod(x)	returns the product of all elements of x
mean(x)	returns the mean of x
median(x)	returns the median of x
var(x)	returns the variance of x
sd(x)	returns the standard deviation of x
cor(x,y)	returns the correlation of x and y
round(x,2)	returns each element of x with 2 digits after the decimal
log(x)	returns the natural logarithm of each element of x

Matrices

x <-as.matrix(x)	converts x to class matrix
t(x)	transpose x
diag(x)	returns the diagnol of x
%*%	matrix multiplication
rowSums(x)	sum of each row of x
colSums(x)	sum of each column of x
rowMeans(x)	mean of each row of x
colMeans(x)	mean of each column of x

Plotting

plot(x,y)	bivariate plot of x and y
hist(x)	histogram of x
pie(x)	pie chart of x
boxplot(x)	boxplot of x

R – Beginners Reference Card