UVic Biology R Handbook

Geoffrey Osgood

March 6, 2021

Contents

Hey everyone! This is a little guide to R. The example data and the R markdown used to make this html (as well as a pdf version) can be found at the github for this website. Feel free to download and use as you need!

GitHub: https://github.com/gjosgood/RhandbookUVicBiology.github.io

If you are not sure how to use GitHub, you can download all the files for your own use by clicking the green "Code" Button and clicking "Download ZIP." That will give you all the code and data from the GitHub!

Basic R facts

R works mainly through vectors, data frames, functions, and variables.

Vectors are a single column (ie from a matrix) containing the same "type" of data (ie numeric, categorical)

Data frames are columns put together into a table (ie your raw data and explanatory variables). Different columns can be different data types. Most often, you will upload and save your data into a data frame in R.

Functions are the operations you perform on your data, such as statistical tests. They are denoted with "()" after the function name. You put arguments for the function in the brackets (eg the data for which you want the mean).

Variables

Variables are objects where you "put your work", such as store or name vectors or data frames in R, or save the output of functions and statistical tests.

Variables are made using either the "=" sign or "<-".

Example: lm.out<-lm(y~x, data=data) saves the output of a linear regression to a variable named lm.out.

Rules for naming variables:

1. Variable names MUST start with a letter (not underscore or number).
2. Variable names MUST contain ONLY letters, numbers, periods ("."), or underscores ("_") and not symbols like %.

In the chunk of code below, "data" is the variable I create to hold my data frame, but I could call it anything I want (like "skunk") as long as the name is not already used by something else and follows the rules.

Packages

Many functions are built into R, but some researchers want to perform tests or make plots outside of base R functionality. Since R is open source, many researchers make "packages" that contain functions for different purposes.

To first install a package, use the function: install.packages("package name"). Ensure the package name is in quotation marks. You only need to do this once.

To load a package for use: library(package) or require(package) with package name NOT in quotation marks. You need to do this everytime, or in every script, you want to use this package.

Example installing packages: vegan is an R package for community ecology analyses.

install.packages("vegan") #you will have to pick a "mirror" - choose "cloud" or one close to you.

to load the package: library(vegan)

Read in, explore, and transform data

To make life easier, from Excel save your data as a CSV or TEXT file (both can be done using 'Save As'). Ensure there are NO SPACES in column headers (use underscore "_" or periods "." or capitals to differentiate words). Keep fancy symbols (ie @) out of your names too.

“Filepath” is the path and name of your file. Right click on your saved data file (csv or text) and use Properties (PC) or Get Info (Mac) to find the filepath. Copy and paste in quotation marks in the read.csv or read.table function. However, you must change all forward slashes to “/”.

You can store data in a data frame or a vector.

Import (read) data into R

```
#Read in data
#data<-read.csv("filepath") - replace filepath with your actual filepath
#data is a variable. You could call it anything.

#example with my own filepath:
mammal_data<-read.csv("c:/Users/gjosg/Dropbox/R_Handbook_2020_UVic/Data/mammals.csv", header=TRUE, strin
#stringsAsFactors=F saves headaches later by ensuring "strings" (ie non-numbers) are read in as categor

#Make a vector
vector_example<-c(145,15,1,5,45,87,83,23) #c() is a function to bring together elements of the same typ
```

You can examine if a data frame read in correctly using head() to see the first six rows; tail to see the last; dim to see the number of rows and columns (in that order); names() for the column names; nrow() and ncol() for the number of rows and columns, respectively; and length() for the length of a vector or list. NOTE: length() will give number of elements in a vector or number of columns of a data frame (remember a data frame is a list of columns). Finally, use str() to see the structure of your data (what columns are there, what kinds of data are stored in each, ie numbers, strings)

Also use names() to rename columns

```
#Take a look to make sure data read in correctly:
head(mammal_data) #see first six rows
```

```
##                  X    body brain
## 1       Arctic fox   3.385  44.5
## 2       Owl monkey   0.480  15.5
## 3 Mountain beaver   1.350   8.1
## 4              Cow 465.000 423.0
## 5        Grey wolf  36.330 119.5
## 6             Goat  27.660 115.0
```

```
names(mammal_data) #access column names
```

```
## [1] "X"     "body"  "brain"
```

```
dim(mammal_data) #number of rows, columns in data frame
```

```
## [1] 62  3
```

```
nrow(mammal_data) #number of rows
```

```
## [1] 62
```

```
ncol(mammal_data) #number of columns
```

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

```
length(vector_example) #number of elements in my vector
```

```
## [1] 8
```

```
str(mammal_data) # how many columns, how many elements in each column, and the type of data in each col
```

```
## 'data.frame':    62 obs. of  3 variables:
##  $ X    : chr  "Arctic fox" "Owl monkey" "Mountain beaver" "Cow" ...
##  $ body : num  3.38 0.48 1.35 465 36.33 ...
##  $ brain: num  44.5 15.5 8.1 423 119.5 ...
```

```
names(mammal_data)<-c("Species", "Brain_size", "Body_size") #rename the three columns of mammal_data
```

To access or reference a single column, it is easiest to use the ""$operator : For example : mammal_{d}ata$Brain_size gives me the brain size column in the mammals data frame.

```
#Retrieve the first six rows (head) of the brain column of mammal_data
head(mammal_data$Brain_size)
```

```
## [1]   3.385   0.480   1.350 465.000  36.330  27.660
```

Data types

The basic data types in R are: factors, characters, numeric, integar, and Boolean.

Factors and characters represent all data that are NOT numbers — a type of data called strings. Strings can be identifid as being within quotation marks (" ").

A factor is used in statistics to represent strings (categorical variables) that have distinct and repeated levels of manipulation, such as a high, medium, and low food treatments in a feeding experiment. If the categorical variable is not composed of a few particular levels that are repeated many times, then the variable is called a "character." Statistical tests are run using factors (when you have categorical explanatory variables), but R will convert characters automatically to factors when needed, so it is always better to work with characters first as they are less error prone while exploring or cleaning data.

Numeric data are numbers you intend to use in analyses as continuous variables. Integers are integers.

Boolean is a special data type for the output of logical statements and takes the form TRUE or FALSE. If you see TRUE or FALSE in all caps in a data column, those data are Boolean. You can create Boolean data using logical statements ($x > y$). See an example in the code chunk below.

You can use str() on your data frame to see how the data in each of your columns are classified. You can use class() to get the class of a single vector or column.

Use as.character() to change a vector or column to a character.

Use as.factor() to change a vector or column to a factor.

Use as.numeric() to change a vector or column to numeric. NOTE: Factors are assigned numbers that represent the order of the levels (ie 1 represents the first level). When you use as.numeric on a factor, you change the vector or column to these numbers. For instance, the string “3” (notice the "“) can be turned to the number 3 using as.numeric(”3“), but only if it is classified as a character in R first. Using as.numeric() after as.factor(”3“) will give 1 (since”3" is the only element given, it is automatically assigned the first level, ie 1). See code chunk below for this in action.

Use as.numeric on Boolean variables to change TRUE to 1 and FALSE to 0 - useful if you are counting how many TRUE statements you have. BUT you can also use mathematical operators on Boolean values and R will automatically turn them into numbers first (ie sum(x>5) will count how many times x is greater than 5).

You can also treat numbers as characters if the numbers do not mean something mathematically but represent a categorical variable - month is an example.

```
#Examine structure of the data:
str(mammal_data)
```

```
## 'data.frame':    62 obs. of  3 variables:
##  $ Species   : chr  "Arctic fox" "Owl monkey" "Mountain beaver" "Cow" ...
##  $ Brain_size: num  3.38 0.48 1.35 465 36.33 ...
##  $ Body_size : num  44.5 15.5 8.1 423 119.5 ...
```

```
#tells me it is a data frame of 62 rows in 3 columns.
#Species in mammal_data is a 'character'
#body and brain are numeric

#Use as.factor() to change a variable to a factor
mammal_data$Species<-as.factor(mammal_data$Species) # $ is how I access a specific column (in this case

class(mammal_data$Species) # tells me what "class" Species is - should be a factor now.
```

```
## [1] "factor"
```

```
#Use as.character() to change a variable to a character again
mammal_data$Species<-as.character(mammal_data$Species)

class(mammal_data$Species) #Should be character now.
```

```
## [1] "character"
```

```
#Numbers to character
mammal_data$Brain_size<-as.character(mammal_data$Brain_size)
class(mammal_data$Brain_size) #Should be character now.
```

```
## [1] "character"
```

```
mammal_data$Brain_size<-as.numeric(mammal_data$Brain_size)
class(mammal_data$Brain_size) #Should be character now.
```

```
## [1] "numeric"
```

```r
#Numbers representing level order when using as.numeric on a factor. Default order is alphabetical (ie
head(as.numeric(as.factor(mammal_data$Brain_size)))
```

```
## [1] 31 14 22 56 44 42
```

```r
#Some packages include example data accessed with the data() function. The mammals data set above is an
library(MASS) #access the data
data(mammals)

#The rows are named based on the species. I can access these names with
head(rownames(mammals)) #first six row names
```

```
## [1] "Arctic fox"      "Owl monkey"      "Mountain beaver" "Cow"
## [5] "Grey wolf"       "Goat"
```

```r
#Boolean examples
#What if I wanted a column indicating if Brain size was greater than 4?
mammal_data$isBrainBig<-mammal_data$Brain_size > 4
head(mammal_data$isBrainBig)
```

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

```r
#how many species have a brain size greater than 4?
sum(mammal_data$isBrainBig)
```

```
## [1] 27
```

```r
#or
sum(mammal_data$Brain_size > 4)
```

```
## [1] 27
```

```r
#Using as.numeric on factors
numberString <- as.factor(c("5", "2", "99", "8", "15")) #use c() to put all these strings together in a

#using as.numeric will return factor levels (alphabetically first will get factor level of 1)
as.numeric(numberString)
```

```
## [1] 3 2 5 4 1
```

```r
#convert to character to ensure as.numeric will produce numbers
as.numeric(as.character(numberString))
```

```
## [1]  5  2 99  8 15
```

Exploring categorical variables, including factor levels

You can use unique() to see how many unique/distinct values or characters exist in a column or vector.

You can use table() to generate frequencies (how many observations of each level of a categorical variable).

You can use levels() to see what levels a factor has and what level order each level has. Automatically, R assigns 1 to the alphabetically first level, but you can change the order yourself using the levels argument in the factor() function. See code chunk for an example.

```r
#what unique levels exist for a grouping variable:
unique(mammal_data$Species)
```

```
##  [1] "Arctic fox"                "Owl monkey"
##  [3] "Mountain beaver"           "Cow"
##  [5] "Grey wolf"                 "Goat"
##  [7] "Roe deer"                  "Guinea pig"
##  [9] "Verbet"                    "Chinchilla"
## [11] "Ground squirrel"           "Arctic ground squirrel"
## [13] "African giant pouched rat" "Lesser short-tailed shrew"
## [15] "Star-nosed mole"           "Nine-banded armadillo"
## [17] "Tree hyrax"                "N.A. opossum"
## [19] "Asian elephant"            "Big brown bat"
## [21] "Donkey"                    "Horse"
## [23] "European hedgehog"         "Patas monkey"
## [25] "Cat"                       "Galago"
## [27] "Genet"                     "Giraffe"
## [29] "Gorilla"                   "Grey seal"
## [31] "Rock hyrax-a"              "Human"
## [33] "African elephant"          "Water opossum"
## [35] "Rhesus monkey"             "Kangaroo"
## [37] "Yellow-bellied marmot"     "Golden hamster"
## [39] "Mouse"                     "Little brown bat"
## [41] "Slow loris"                "Okapi"
## [43] "Rabbit"                    "Sheep"
## [45] "Jaguar"                    "Chimpanzee"
## [47] "Baboon"                    "Desert hedgehog"
## [49] "Giant armadillo"           "Rock hyrax-b"
## [51] "Raccoon"                   "Rat"
## [53] "E. American mole"          "Mole rat"
## [55] "Musk shrew"                "Pig"
## [57] "Echidna"                   "Brazilian tapir"
## [59] "Tenrec"                    "Phalanger"
## [61] "Tree shrew"                "Red fox"
```

```r
#how many observations are there for each level of a categorical variable:
table(mammal_data$Species)
```

```
##
##          African elephant African giant pouched rat                Arctic fox
##                         1                         1                         1
##    Arctic ground squirrel            Asian elephant                    Baboon
##                         1                         1                         1
##             Big brown bat           Brazilian tapir                       Cat
##                         1                         1                         1
##                Chimpanzee                Chinchilla                       Cow
##                         1                         1                         1
##           Desert hedgehog                    Donkey          E. American mole
##                         1                         1                         1
##                   Echidna         European hedgehog                    Galago
##                         1                         1                         1
##                     Genet           Giant armadillo                   Giraffe
##                         1                         1                         1
##                      Goat            Golden hamster                   Gorilla
##                         1                         1                         1
##                 Grey seal                 Grey wolf           Ground squirrel
##                         1                         1                         1
##                Guinea pig                     Horse                     Human
##                         1                         1                         1
##                    Jaguar                  Kangaroo Lesser short-tailed shrew
##                         1                         1                         1
##          Little brown bat                  Mole rat           Mountain beaver
##                         1                         1                         1
##                     Mouse                Musk shrew               N.A. opossum
##                         1                         1                         1
##     Nine-banded armadillo                     Okapi                Owl monkey
##                         1                         1                         1
##              Patas monkey                 Phalanger                       Pig
##                         1                         1                         1
##                    Rabbit                   Raccoon                       Rat
##                         1                         1                         1
##                   Red fox             Rhesus monkey              Rock hyrax-a
##                         1                         1                         1
##              Rock hyrax-b                  Roe deer                     Sheep
##                         1                         1                         1
##                Slow loris           Star-nosed mole                    Tenrec
##                         1                         1                         1
##                Tree hyrax                Tree shrew                    Verbet
##                         1                         1                         1
##             Water opossum     Yellow-bellied marmot
##                         1                         1
```

```
#factor levels example
colours<-as.factor(c("red", "blue", "green")) #use c() to put the strings together.

levels(colours) # the levels and their order. Notice "blue" is first
```

```
## [1] "blue"  "green" "red"
```

```
colours<-factor(colours, levels=c("red", "green", "blue"))
levels(colours) #now "red" is first and "blue" is last
```

```
## [1] "red"   "green" "blue"
```

Transform data - log, power, trigonometry

To transform data, take the function you want (ie log) and apply it to the column of data you wish to transform. You can either replace the column with the transformed data, or make a new column using the $ symbol, which is used to access columns in a data frame.

You can save transformed data into in the same column, but I recommend making a new column.

```r
library(MASS) #access the data
data(mammals)

ncol(mammals) # 2 columns
```

```
## [1] 2
```

```r
#Transform data

###################
#Log-transform#####
###################
mammals$log_body <- log(mammals$body) #make a new column and put log-transformed body values into it. T

ncol(mammals) # a new column has been added
```

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

```r
mammals$log10_body <- log(mammals$body, base=10) #make a new column and put log-transformed body values


#Exponential function to reverse ln-transformation
mammals$undo_log <- exp(mammals$log_body) #make a new column and put exponential transformed log-body v

#Exponential function to reverse log10-transformation
mammals$undo_log10 <- 10^(mammals$log10_body) #make a new column and put exponential transformed log-bo

#####################
#Powers:#############
#####################

#Square root:
mammals$sqrt_brain <- sqrt(mammals$brain) #make a new column and put square-root transformed brain valu

mammals$brain_squared <- mammals$brain^2 #make a new column and put squared transformed brain values in
mammals$brain_cubed <- mammals$brain^3 #make a new column and put cubed transformed brain values into i

mammals$brain_cubic_root <- mammals$brain^(1/3) #make a new column and put cubic root of brain values i

mammals$arcsine_brain <- asin(mammals$brain) #make a new column and put cubic root
```

```
## Warning in asin(mammals$brain): NaNs produced
```

```
############################
#Trigonometry functions:####
############################
#Making example data:##
invasive_data<-data.frame(Treatment=rep(c("Pesticide", "Control"), rep(10,2)), Number_of_Quadrats=c(15,
##################

#Transformations:
invasive_data$ProportionInvasive<-invasive_data$InvasiveSpecies/invasive_data$Number_of_Quadrats #calcu

#Sine, cosine, and tangent
invasive_data$sine_prop<-sin(invasive_data$ProportionInvasive)
invasive_data$cosine_prop<-cos(invasive_data$ProportionInvasive)
invasive_data$tangent_prop<-tan(invasive_data$ProportionInvasive)

#Arcsine, arc-cosine, arc-tangent
invasive_data$arcsine_prop<-asin(invasive_data$ProportionInvasive)
invasive_data$arc_cosine_prop<-acos(invasive_data$ProportionInvasive)
invasive_data$arc_tangent_prop<-atan(invasive_data$ProportionInvasive)
```

Subset and order data frames

Use "[]" to access specific rows, columns, or cells.

Use subset() to subset data based on levels of a categorical variable (eg. get only the data in a "low" treatment).

You can use order() to re-order data frames based on alphabetical or numerical order.

```
#Example data set:
#Make a data frame holding Shannon diversity indices for sites within two MPAs, one column to hold the

#the shannon diversity held in separate objects called vectors
#A vector is a single column of data of one type
WhaleSanctuary=c(1.8816523, 1.8389828, 1.3239195, 0.6931472,  0.6615632,1.2299186, 1.9963156, 1.0045784

KelpMPA=c(1.3835889, 1.2442544, 1.0567522, 0.9973048, 1.9783942,1.1156818, 1.3214234, 1.9923252, 1.2984

length(WhaleSanctuary) #length() tells me how many entries are in a list or vector
```

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

```
#You can also store them together in a data frame:
MPA_Data<-data.frame(MPA=c(rep("WhaleSanctuary", length(WhaleSanctuary)), rep("KelpMPA", length(KelpMPA
#makes a data frame with each row as an observation. One column identifies the MPA and the other column
#rep() replicates any text the specified number of times.
#so the MPA column repeats WhaleSanctuary for all the WhaleSanctuary numbers and then repeats KelpMPA f

#Subsetting data frames:
#getting only the Kelp MPA data:
KelP_MPA<-subset(MPA_Data, MPA=="KelpMPA") # == sign means find something exactly equal to this - not t

#Retrieve cells from a data frame:
```

```
#[] is used to access data frames and matrices. In R: rows go before columns, so
MPA_Data[3,2] #will give me the value in the third row, second column of MPA_Data
```

```
## [1] 1.323919
```

```
#If I want all of one row: leave the columns blank:
MPA_Data[2,] #gives me entire second row over all columns
```

```
##             MPA ShannonDiversity
## 2 WhaleSanctuary         1.838983
```

```
#If I want an entire column:
MPA_Data[,2] #gives me entire second column
```

```
##  [1] 1.8816523 1.8389828 1.3239195 0.6931472 0.6615632 1.2299186 1.9963156
##  [8] 1.0045784 0.3767702 2.0468186 1.3835889 1.2442544 1.0567522 0.9973048
## [15] 1.9783942 1.1156818 1.3214234 1.9923252 1.2984907 1.3025272
```

```
# $ is also used to access specific columns of a data frame (or parts of a list - a data frame is reall
MPA_Data$MPA #gives me the MPA column
```

```
##  [1] "WhaleSanctuary" "WhaleSanctuary" "WhaleSanctuary" "WhaleSanctuary"
##  [5] "WhaleSanctuary" "WhaleSanctuary" "WhaleSanctuary" "WhaleSanctuary"
##  [9] "WhaleSanctuary" "WhaleSanctuary" "KelpMPA"        "KelpMPA"
## [13] "KelpMPA"        "KelpMPA"        "KelpMPA"        "KelpMPA"
## [17] "KelpMPA"        "KelpMPA"        "KelpMPA"        "KelpMPA"
```

The function order() orders rows from A to Z or smallest to largest. Put a negative size in the brackets to do the reverse or use the decreasing=TRUE argument.

```
#Order rows with small diversity first
MPA_Data[order(MPA_Data$ShannonDiversity),]
```

```
##              MPA ShannonDiversity
## 9  WhaleSanctuary        0.3767702
## 5  WhaleSanctuary        0.6615632
## 4  WhaleSanctuary        0.6931472
## 14        KelpMPA        0.9973048
## 8  WhaleSanctuary        1.0045784
## 13        KelpMPA        1.0567522
## 16        KelpMPA        1.1156818
## 6  WhaleSanctuary        1.2299186
## 12        KelpMPA        1.2442544
## 19        KelpMPA        1.2984907
## 20        KelpMPA        1.3025272
## 17        KelpMPA        1.3214234
## 3  WhaleSanctuary        1.3239195
## 11        KelpMPA        1.3835889
## 2  WhaleSanctuary        1.8389828
## 1  WhaleSanctuary        1.8816523
## 15        KelpMPA        1.9783942
## 18        KelpMPA        1.9923252
## 7  WhaleSanctuary        1.9963156
## 10 WhaleSanctuary        2.0468186
```

```
#Order sites with the largest diversity first
MPA_Data[order(-MPA_Data$ShannonDiversity),]
```

```
##               MPA ShannonDiversity
## 10 WhaleSanctuary        2.0468186
## 7  WhaleSanctuary        1.9963156
## 18        KelpMPA        1.9923252
## 15        KelpMPA        1.9783942
## 1  WhaleSanctuary        1.8816523
## 2  WhaleSanctuary        1.8389828
## 11        KelpMPA        1.3835889
## 3  WhaleSanctuary        1.3239195
## 17        KelpMPA        1.3214234
## 20        KelpMPA        1.3025272
## 19        KelpMPA        1.2984907
## 12        KelpMPA        1.2442544
## 6  WhaleSanctuary        1.2299186
## 16        KelpMPA        1.1156818
## 13        KelpMPA        1.0567522
## 8  WhaleSanctuary        1.0045784
## 14        KelpMPA        0.9973048
## 4  WhaleSanctuary        0.6931472
## 5  WhaleSanctuary        0.6615632
## 9  WhaleSanctuary        0.3767702
```

```
#Order brain size based on species - alphabetical order
mammal_data[order(mammal_data$Species, decreasing=FALSE),]
```

```
##                     Species Brain_size Body_size isBrainBig
## 33         African elephant   6654.000   5712.00       TRUE
## 13 African giant pouched rat      1.000      6.60      FALSE
## 1                Arctic fox      3.385     44.50      FALSE
## 12    Arctic ground squirrel      0.920      5.70      FALSE
## 19           Asian elephant   2547.000   4603.00       TRUE
## 47                   Baboon     10.550    179.50       TRUE
## 20            Big brown bat      0.023      0.30      FALSE
## 58          Brazilian tapir    160.000    169.00       TRUE
## 25                      Cat      3.300     25.60      FALSE
## 46               Chimpanzee     52.160    440.00       TRUE
## 10               Chinchilla      0.425      6.40      FALSE
## 4                       Cow    465.000    423.00       TRUE
## 48          Desert hedgehog      0.550      2.40      FALSE
## 21                   Donkey    187.100    419.00       TRUE
## 53        E. American mole      0.075      1.20      FALSE
## 57                  Echidna      3.000     25.00      FALSE
## 23       European hedgehog      0.785      3.50      FALSE
## 26                   Galago      0.200      5.00      FALSE
## 27                    Genet      1.410     17.50      FALSE
## 49          Giant armadillo     60.000     81.00        TRUE
## 28                  Giraffe    529.000    680.00        TRUE
## 6                      Goat     27.660    115.00        TRUE
## 38           Golden hamster      0.120      1.00       FALSE
## 29                  Gorilla    207.000    406.00        TRUE
## 30                Grey seal     85.000    325.00        TRUE
## 5                 Grey wolf     36.330    119.50        TRUE
## 11          Ground squirrel      0.101      4.00       FALSE
## 8                Guinea pig      1.040      5.50       FALSE
## 22                    Horse    521.000    655.00        TRUE
## 32                    Human     62.000   1320.00        TRUE
## 45                   Jaguar    100.000    157.00        TRUE
## 36                 Kangaroo     35.000     56.00        TRUE
## 14 Lesser short-tailed shrew      0.005      0.14       FALSE
## 40          Little brown bat      0.010      0.25       FALSE
## 54                 Mole rat      0.122      3.00       FALSE
## 3           Mountain beaver      1.350      8.10       FALSE
## 39                    Mouse      0.023      0.40       FALSE
## 55               Musk shrew      0.048      0.33       FALSE
## 18              N.A. opossum      1.700      6.30       FALSE
## 16      Nine-banded armadillo      3.500     10.80       FALSE
## 42                    Okapi    250.000    490.00        TRUE
## 2                Owl monkey      0.480     15.50       FALSE
## 24             Patas monkey     10.000    115.00        TRUE
## 60                Phalanger      1.620     11.40       FALSE
## 56                      Pig    192.000    180.00        TRUE
## 43                   Rabbit      2.500     12.10       FALSE
## 51                  Raccoon      4.288     39.20        TRUE
## 52                      Rat      0.280      1.90       FALSE
## 62                  Red fox      4.235     50.40        TRUE
## 35            Rhesus monkey      6.800    179.00        TRUE
## 31             Rock hyrax-a      0.750     12.30       FALSE
## 50             Rock hyrax-b      3.600     21.00       FALSE
## 7                  Roe deer     14.830     98.20        TRUE
## 44                    Sheep     55.500    175.00        TRUE
## 41               Slow loris      1.400     12.50       FALSE
## 15          Star-nosed mole      0.060      1.00       FALSE
## 59                   Tenrec      0.900      2.60       FALSE
## 17               Tree hyrax      2.000     12.30       FALSE
## 61               Tree shrew      0.104      2.50       FALSE
## 9                    Verbet      4.190     58.00        TRUE
## 34            Water opossum      3.500      3.90       FALSE
## 37      Yellow-bellied marmot      4.050     17.00        TRUE
```

```
#Order brain size based on species - reverse alphabetical order
mammal_data[order(mammal_data$Species, decreasing=TRUE),]
```

```
##                     Species Brain_size Body_size isBrainBig
## 37      Yellow-bellied marmot      4.050     17.00        TRUE
## 34            Water opossum      3.500      3.90       FALSE
## 9                    Verbet      4.190     58.00        TRUE
## 61               Tree shrew      0.104      2.50       FALSE
## 17               Tree hyrax      2.000     12.30       FALSE
## 59                   Tenrec      0.900      2.60       FALSE
## 15           Star-nosed mole      0.060     1.00       FALSE
## 41                Slow loris      1.400    12.50       FALSE
## 44                     Sheep     55.500   175.00        TRUE
## 7                   Roe deer     14.830    98.20        TRUE
## 50              Rock hyrax-b      3.600    21.00       FALSE
## 31              Rock hyrax-a      0.750    12.30       FALSE
## 35             Rhesus monkey      6.800   179.00        TRUE
## 62                   Red fox      4.235    50.40        TRUE
## 52                       Rat      0.280     1.90       FALSE
## 51                   Raccoon      4.288    39.20        TRUE
## 43                    Rabbit      2.500    12.10       FALSE
## 56                       Pig    192.000   180.00        TRUE
## 60                 Phalanger      1.620    11.40       FALSE
## 24              Patas monkey     10.000   115.00        TRUE
## 2                 Owl monkey      0.480    15.50       FALSE
## 42                     Okapi    250.000   490.00        TRUE
## 16     Nine-banded armadillo      3.500    10.80       FALSE
## 18              N.A. opossum      1.700     6.30       FALSE
## 55                Musk shrew      0.048     0.33       FALSE
## 39                     Mouse      0.023     0.40       FALSE
## 3            Mountain beaver      1.350     8.10       FALSE
## 54                  Mole rat      0.122     3.00       FALSE
## 40          Little brown bat      0.010     0.25       FALSE
## 14 Lesser short-tailed shrew      0.005     0.14       FALSE
## 36                  Kangaroo     35.000    56.00        TRUE
## 45                    Jaguar    100.000   157.00        TRUE
## 32                     Human     62.000  1320.00        TRUE
## 22                     Horse    521.000   655.00        TRUE
## 8                 Guinea pig      1.040     5.50       FALSE
## 11           Ground squirrel      0.101     4.00       FALSE
## 5                  Grey wolf     36.330   119.50        TRUE
## 30                 Grey seal     85.000   325.00        TRUE
## 29                   Gorilla    207.000   406.00        TRUE
## 38            Golden hamster      0.120     1.00       FALSE
## 6                       Goat     27.660   115.00        TRUE
## 28                   Giraffe    529.000   680.00        TRUE
## 49           Giant armadillo     60.000    81.00        TRUE
## 27                     Genet      1.410    17.50       FALSE
## 26                    Galago      0.200     5.00       FALSE
## 23         European hedgehog      0.785     3.50       FALSE
## 57                   Echidna      3.000    25.00       FALSE
## 53          E. American mole      0.075     1.20       FALSE
## 21                    Donkey    187.100   419.00        TRUE
## 48           Desert hedgehog      0.550     2.40       FALSE
## 4                        Cow    465.000   423.00        TRUE
## 10                Chinchilla      0.425     6.40       FALSE
## 46                Chimpanzee     52.160   440.00        TRUE
## 25                       Cat      3.300    25.60       FALSE
## 58           Brazilian tapir    160.000   169.00        TRUE
## 20              Big brown bat      0.023     0.30       FALSE
## 47                    Baboon     10.550   179.50        TRUE
## 19            Asian elephant   2547.000  4603.00        TRUE
## 12    Arctic ground squirrel      0.920     5.70       FALSE
## 1                 Arctic fox      3.385    44.50       FALSE
## 13 African giant pouched rat      1.000      6.60      FALSE
## 33          African elephant   6654.000   5712.00       TRUE
```

Reshaping data - when you have multiple response variables or levels, there are two ways to organize your data frame:

1. "wide" format with each variable as its own column.
2. "long" with the values for the variable in one column and an ID column indicating which variable or level the value represents.

Use the function melt() to change wide format to long and the function dcast() to turn long into wide format - in the reshape2 package.

Example: Ozone levels in different months, days, etc.

```
#Example data set - ozone levels
#install.packages("mlbench")
library(mlbench)
data(Ozone)
Ozone_example<-Ozone[,c(1,2,4,5,6,8)] # I only want some columns for the example
names(Ozone_example)<-c("Month", "Day", "Ozone", "Pressure", "Wind", "Temperature")
head(Ozone_example) # it is currently in wide format, with ozone, pressure, wind, temperature each in t
```

```
##   Month Day Ozone Pressure Wind Temperature
## 1     1   1     3     5480    8          NA
## 2     1   2     3     5660    6          38
## 3     1   3     3     5710    4          40
## 4     1   4     5     5700    3          45
## 5     1   5     5     5760    3          54
## 6     1   6     6     5720    4          35
```

```
dim(Ozone_example)
```

```
## [1] 366   6
```

```
#Reshaping data frames
#install.packages("reshape2")
library(reshape2)

#Turn wide format into long
Ozone_long<-melt(Ozone_example, id.vars=c("Month", "Day")) # I want to take the climatic variables into
names(Ozone_long)<-c("Month", "Day", "Climatic_variable", "Climatic_value")
head(Ozone_long) #the four columns of climate variables have been collapased into 2, and the data frame
```

```
##   Month Day Climatic_variable Climatic_value
## 1     1   1             Ozone              3
## 2     1   2             Ozone              3
## 3     1   3             Ozone              3
## 4     1   4             Ozone              5
## 5     1   5             Ozone              5
## 6     1   6             Ozone              6
```

```
dim(Ozone_long)
```

```
## [1] 1464    4
```

```
#Turn wide into long format - use dcast
Ozone_wide<-dcast(Ozone_long, Month+Day~Climatic_variable, value.vars=c("Climatic_value"))#I want to ke
```

```
## Using Climatic_value as value column: use value.var to override.
```

```
head(Ozone_long)
```

```
##   Month Day Climatic_variable Climatic_value
## 1     1   1             Ozone              3
## 2     1   2             Ozone              3
## 3     1   3             Ozone              3
## 4     1   4             Ozone              5
## 5     1   5             Ozone              5
## 6     1   6             Ozone              6
```

```
dim(Ozone_long)
```

```
## [1] 1464    4
```

Summary statistics

Read in example data

```
#Example data used in ANOVA section
library(carData) #remember to install.packages(carData) before first use
data(Soils) #some packages have data sets saved in them. The data() function retrieves them - in this c
head(Soils)
```

```
##   Group Contour Depth Gp Block   pH     N Dens   P    Ca   Mg    K   Na Conduc
## 1     1     Top  0-10 T0     1 5.40 0.188 0.92 215 16.35 7.65 0.72 1.14   1.09
## 2     1     Top  0-10 T0     2 5.65 0.165 1.04 208 12.25 5.15 0.71 0.94   1.35
## 3     1     Top  0-10 T0     3 5.14 0.260 0.95 300 13.02 5.68 0.68 0.60   1.41
## 4     1     Top  0-10 T0     4 5.14 0.169 1.10 248 11.92 7.88 1.09 1.01   1.64
## 5     2     Top 10-30 T1     1 5.14 0.164 1.12 174 14.17 8.12 0.70 2.17   1.85
## 6     2     Top 10-30 T1     2 5.10 0.094 1.22 129  8.55 6.92 0.81 2.67   3.18
```

Mean, range, standard deviation, standard error

```
#Mean pH across all samples:
mean(Soils$pH)
```

```
## [1] 4.669375
```

```r
#Mean pH by Depth class:
aggregate(pH~Depth, data=Soils, FUN=mean) #aggregate runs a function for each level of a specified grou
```

```
##   Depth       pH
## 1  0-10 5.397500
## 2 10-30 5.004167
## 3 30-60 4.278333
## 4 60-90 3.997500
```

```r
#Standard deviation of pH across all samples:
sd(Soils$pH)
```

```
## [1] 0.6718549
```

```r
#Standard deviation of pH by depth:
aggregate(pH~Depth, data=Soils, FUN=sd)
```

```
##   Depth        pH
## 1  0-10 0.2487103
## 2 10-30 0.5968624
## 3 30-60 0.3306422
## 4 60-90 0.2032967
```

```r
#Variance of pH across all samples:
var(Soils$pH)
```

```
## [1] 0.451389
```

```r
#Variance of pH by depth:
aggregate(pH~Depth, data=Soils, FUN=var)
```

```
##   Depth         pH
## 1  0-10 0.06185682
## 2 10-30 0.35624470
## 3 30-60 0.10932424
## 4 60-90 0.04132955
```

```r
#Standard error of the mean
#There is no function for standard error in R, so we have to make our own using the standard error form
se<-function(x) {sd(x)/sqrt(length(x))}
#Standard error of the mean of pH across all samples:
se(Soils$pH)
```

```
## [1] 0.0969739
```

```r
#Standard error of the mean of pH by depth:
aggregate(pH~Depth, data=Soils, FUN=se)
```

```
##   Depth         pH
## 1  0-10 0.07179648
## 2 10-30 0.17229933
## 3 30-60 0.09544817
## 4 60-90 0.05868670
```

```
#Can also get minimum, maximum, mean, and quartiles:
summary(Soils$pH)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
##   3.740   4.058   4.545   4.669   5.140   6.670
```

```
summary(Soils) #get summaries of all variables at once
```

```
##      Group           Contour     Depth          Gp     Block        pH       
##  1      : 4   Depression:16   0-10 :12   D0     : 4   1:12   Min.   :3.740  
##  2      : 4   Slope     :16   10-30:12   D1     : 4   2:12   1st Qu.:4.058  
##  3      : 4   Top       :16   30-60:12   D3     : 4   3:12   Median :4.545  
##  4      : 4                   60-90:12   D6     : 4   4:12   Mean   :4.669  
##  5      : 4                              S0     : 4          3rd Qu.:5.140  
##  6      : 4                              S1     : 4          Max.   :6.670  
##  (Other):24                              (Other):24                         
##        N                 Dens              P               Ca        
##  Min.   :0.03000   Min.   :0.780   Min.   : 79.0   Min.   : 3.820  
##  1st Qu.:0.05075   1st Qu.:1.127   1st Qu.:108.8   1st Qu.: 5.040  
##  Median :0.08450   Median :1.400   Median :131.0   Median : 7.305  
##  Mean   :0.10194   Mean   :1.316   Mean   :166.2   Mean   : 8.029  
##  3rd Qu.:0.12925   3rd Qu.:1.502   3rd Qu.:214.2   3rd Qu.: 9.735  
##  Max.   :0.29800   Max.   :1.600   Max.   :445.0   Max.   :16.350  
##                                                                   
##        Mg               K                Na             Conduc      
##  Min.   : 5.150   Min.   :0.1400   Min.   : 0.600   Min.   : 0.670  
##  1st Qu.: 7.537   1st Qu.:0.2750   1st Qu.: 2.545   1st Qu.: 2.790  
##  Median : 8.515   Median :0.4250   Median : 5.520   Median : 6.635  
##  Mean   : 8.465   Mean   :0.4662   Mean   : 5.600   Mean   : 6.589  
##  3rd Qu.: 9.648   3rd Qu.:0.6425   3rd Qu.: 8.355   3rd Qu.: 9.852  
##  Max.   :10.960   Max.   :1.0900   Max.   :11.040   Max.   :13.320  
## 
```

```
#or can use these functions (with aggregate as needed):
max(Soils$pH) # for maximum
```

```
## [1] 6.67
```

```
min(Soils$pH) # for minimum
```

```
## [1] 3.74
```

```
range(Soils$pH) # for range
```

```
## [1] 3.74 6.67
```

```r
median(Soils$pH) # for median
```

```
## [1] 4.545
```

Confidence intervals:

Here are the steps for calculating confidence intervals in R:

Calculate confidence intervals per depth interval example.

1. Get the mean and standard error.

```r
#I need the mean and se to find the confidence intervals, so I store it in an object:

#in case you need to run the custom se function (R does not have one of its own):
se<-function(x) {sd(x)/sqrt(length(x))}

se_pH<-aggregate(pH~Depth, data=Soils, FUN=se) #se by depth
mean_pH<-aggregate(pH~Depth, data=Soils, FUN=mean) #mean by depth
names(se_pH) <- c("Depth", "SE_pH") #give me names representing what is in this data frame created by a
names(mean_pH) <- c("Depth", "Mean_pH")
```

2. Multiply the standard error by the correct quantile (for the normal distribution) to calculate the confidence limits.

The function qnorm() gives me critical values for normal distribution for a given % of the area under the curve.

For 95% confidence intervals, I look for the quantiles at 97.5%, which are the same since the normal distribution is symmetrical. So, I only need to use 0.975 in my qnorm function below. However, if my hypothesis is one-tailed, then I'll need either 0.05 or 0.95 in the function instead, depending on whether it is lower or upper tailed.

```r
#This line gives me the plus/minus value by Depth interval, which I add to the mean to get the limits:
limits<-se_pH$SE_pH*qnorm(0.975, mean=mean_pH$Mean_pH) #for 95% confidence intervals two-sided of a nor
```

3. Subtract and add the confidence limit to the mean to calculate the full confidence interval.

```r
#Upper limit
upper_CI<-mean_pH$Mean_pH+limits
#Lower limit
lower_CI<-mean_pH$Mean_pH-limits

#Put together into one data frame of means and confidence intervals by depth:
pH_mean_CI<-data.frame(Depth=mean_pH$Depth, Mean_pH=mean_pH$Mean_pH, Upper_CI=upper_CI, Lower_CI=lower_
```

Plots

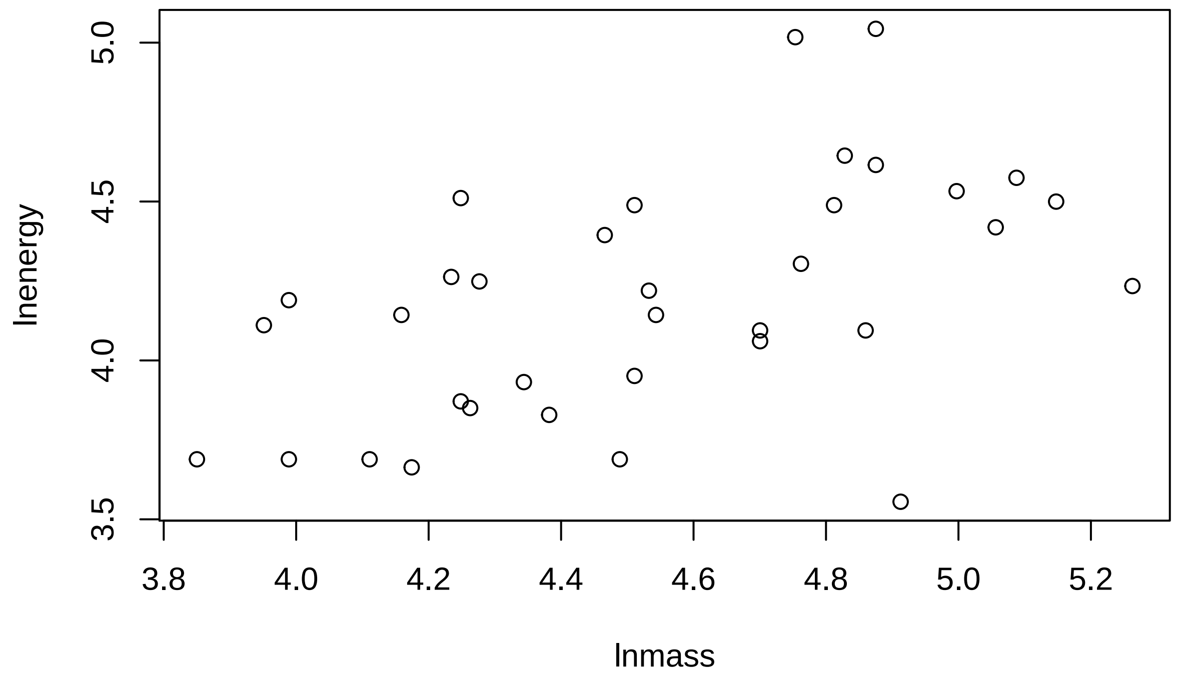
Plotting basics and scatterplots

The plot() function is the base plotting function in R.

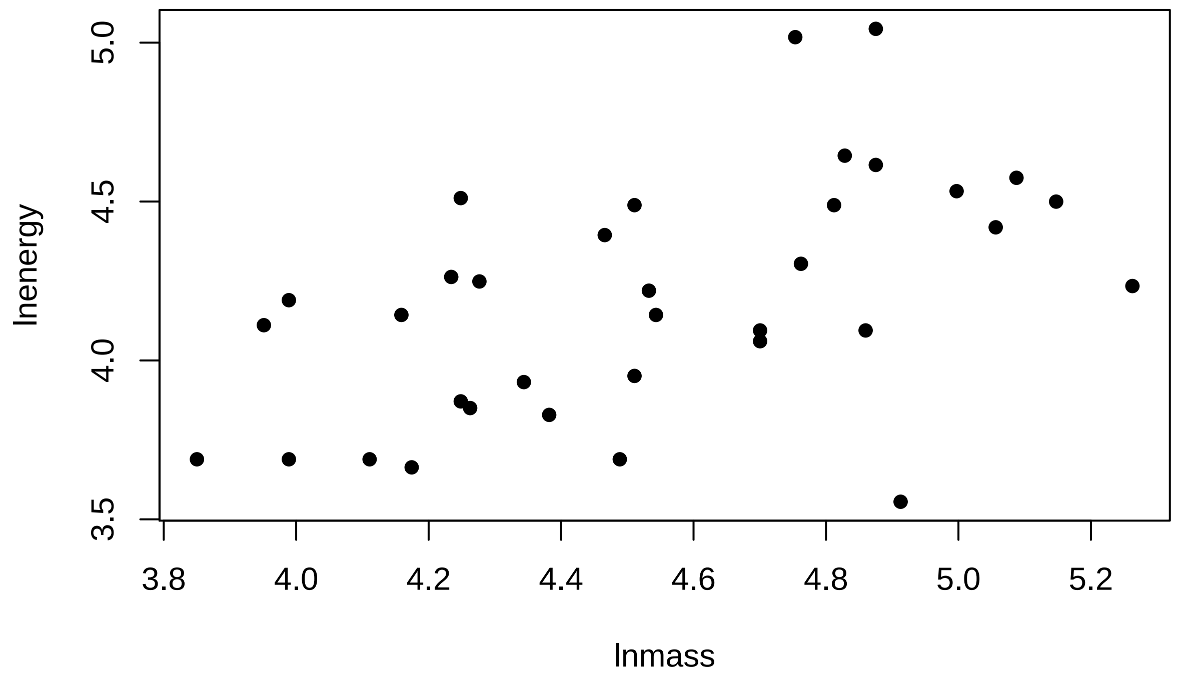
Use y~x to plot a response variable (y) against an explanatory (x). Use the data argument to specify where x and y are stored, if they are not their own vectors already.

Change point size, type, and colour

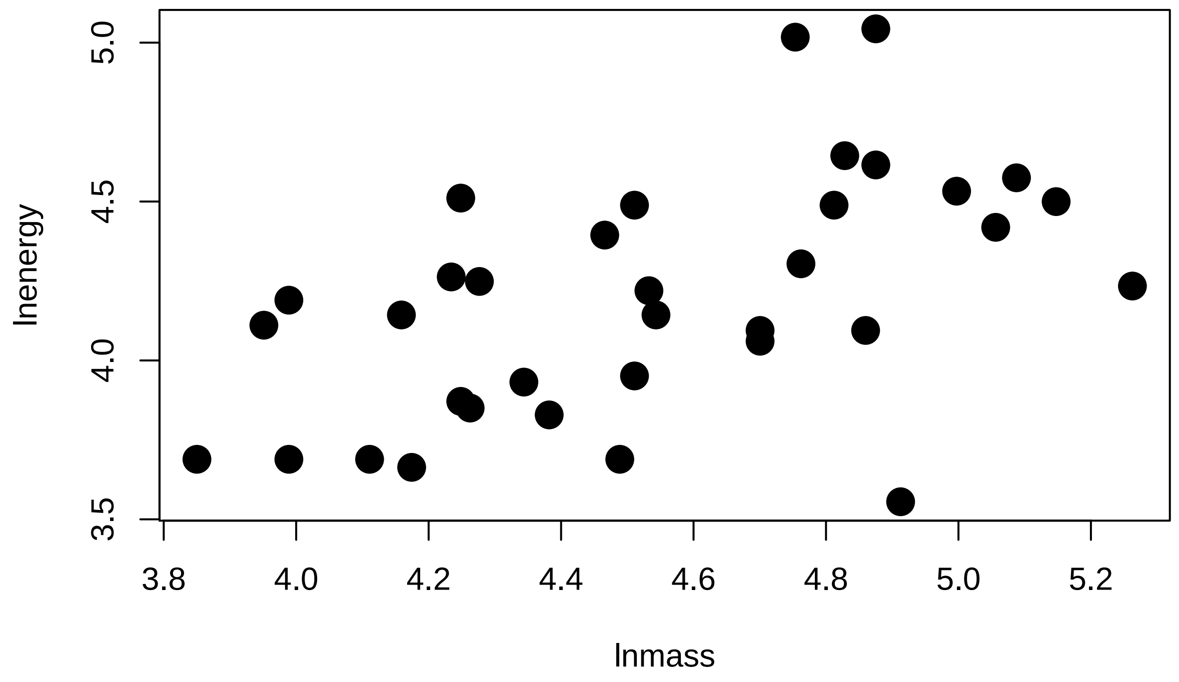
```
#Example data
MoleRats<-read.csv("c:/Users/gjosg/Dropbox/R_Handbook_2020_UVic/Data/MoleRats.csv", header=TRUE)

MoleRats$caste<-factor(MoleRats$caste) #ensure my grouping variable is a factor for easier plotting

plot(lnenergy~lnmass, data=MoleRats) #basic scatterplot
```



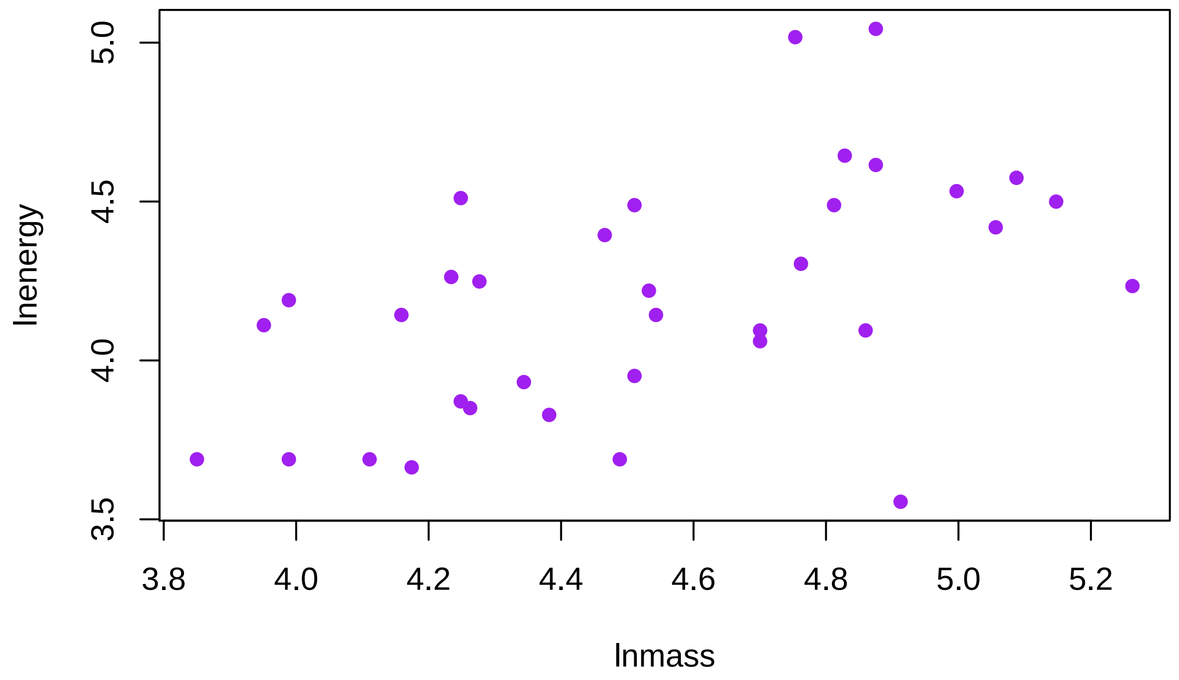
```
plot(lnenergy~lnmass, data=MoleRats, pch=16) #pch changes point type. Try a few different numbers.
```



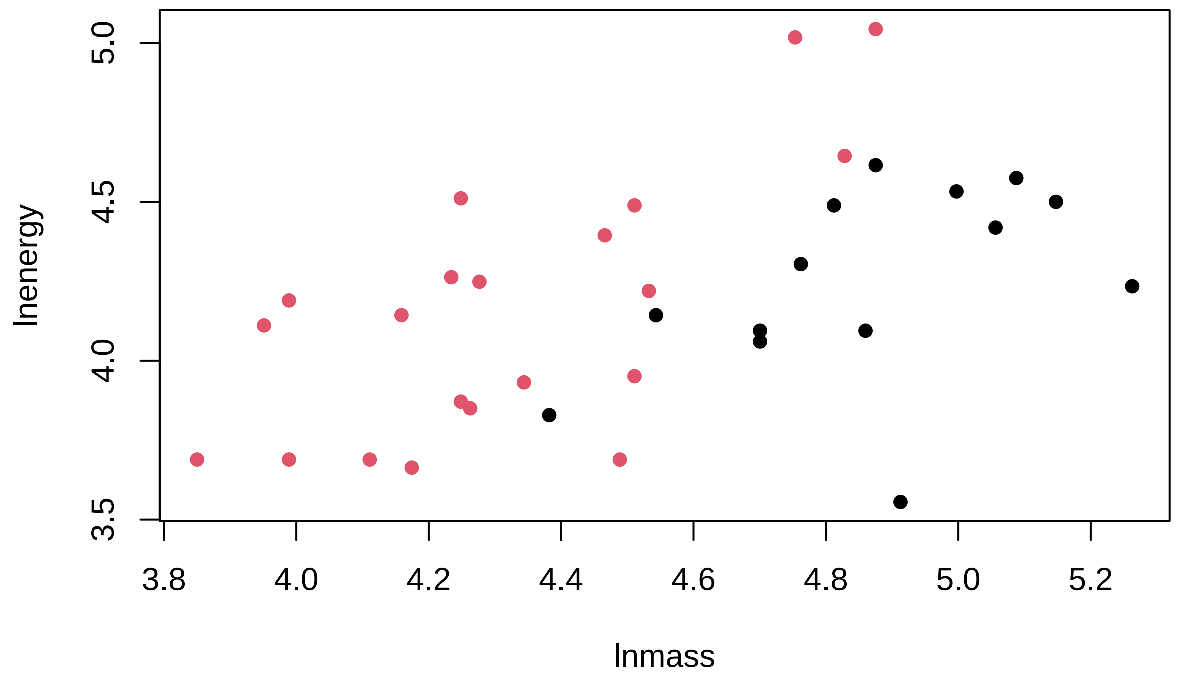
```
plot(lnenergy~lnmass, data=MoleRats, pch=16, cex=2) #cex changes point size. Try a few different number
```



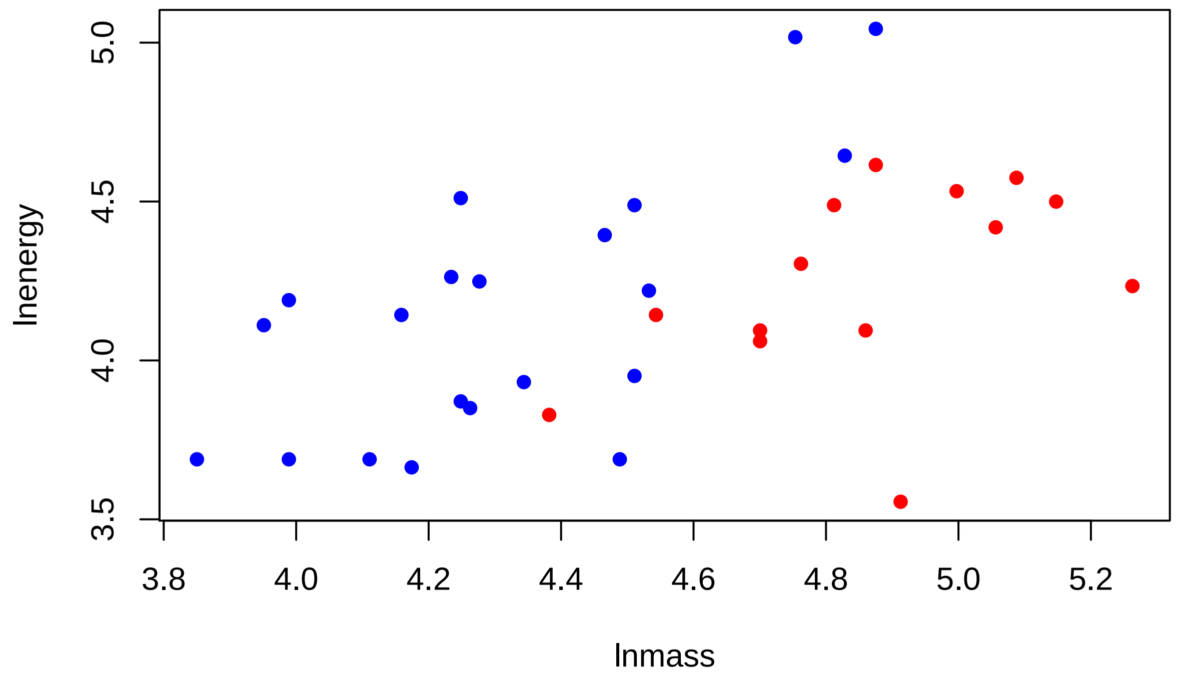
```
plot(lnenergy~lnmass, data=MoleRats, pch=16, col="purple") #col changes colour
```



```r
plot(lnenergy~lnmass, data=MoleRats, pch=16, col=as.numeric(MoleRats$caste)) #col changes colour based
```



```r
####CUSTOM COLOURS#############

#Make your own vector of colours that will be referenced in the col argument.

colours<-c("red", "blue")#define colours you want to use

plot(lnenergy~lnmass, data=MoleRats, pch=16, col=colours[as.numeric(MoleRats$caste)]) #more customizabl
```



Change axes and boxes around plots

The easiest way to customize an axis is to remove the default one in the plot command, using yaxis="n" or xaxis="n" and then using the axis() function re-make the axis as you like. Use axis(1) for the x-axis and axis(2) for the y-axis. The top of a plot is axis(3) and the right (second y-axis) is axis(4).

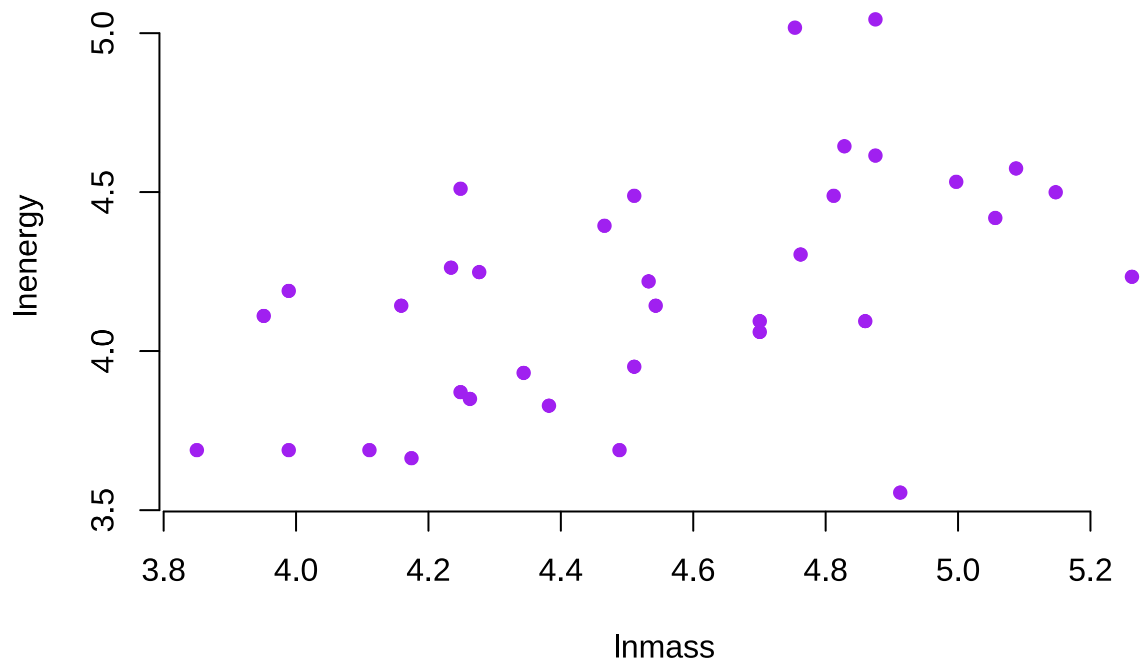
TO customize what labels are show, use the labels and at arguments. The labels argument specifies the labels, and the at argument specifies where along the axis you want to put the labels. They must be the same length (ie you cannot have more labels than places where you want to put the labels).

Use las=2 to flip axis labels 90 degrees. You can use las=2 directly in the plot() function, but it will flip BOTH the x- and y-axis. Often, you just want to flip the y-axis, so I suppress it with yaxt="n" and remake it with axis(2, las=2). See example in the code chunk below.

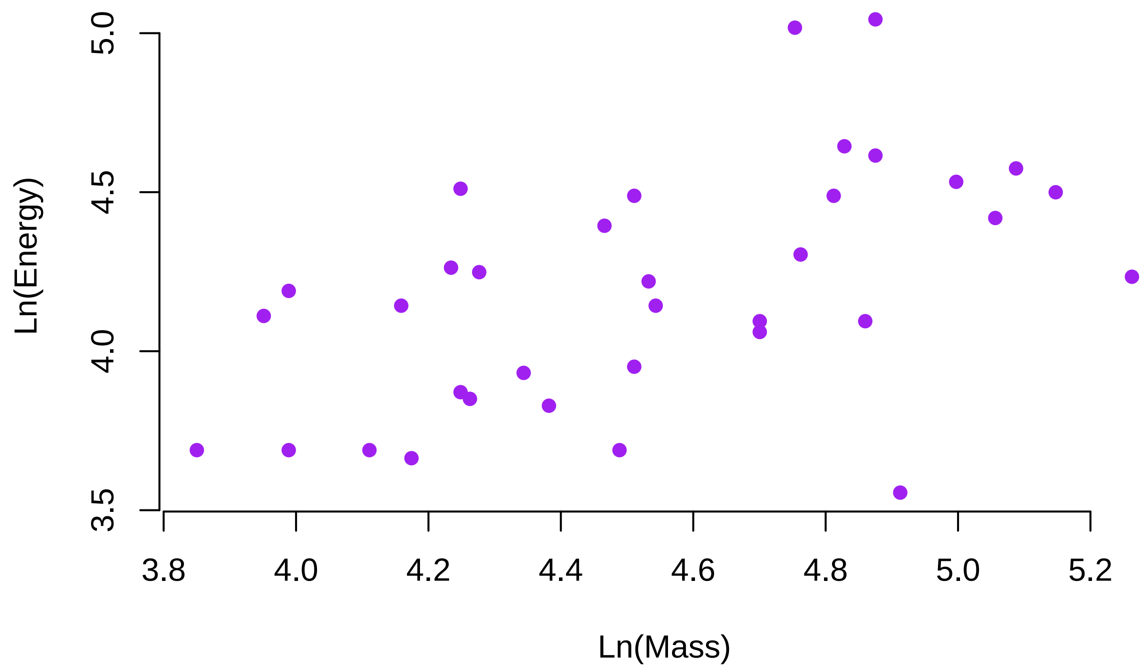
Use ylim and xlim arguments in the plot() function change y-axis and x-axis limits, respectively

Use ylab and xlab arguments in the plot() function change y-axis and x-axis titles, respectively (another way is to use mtext below).

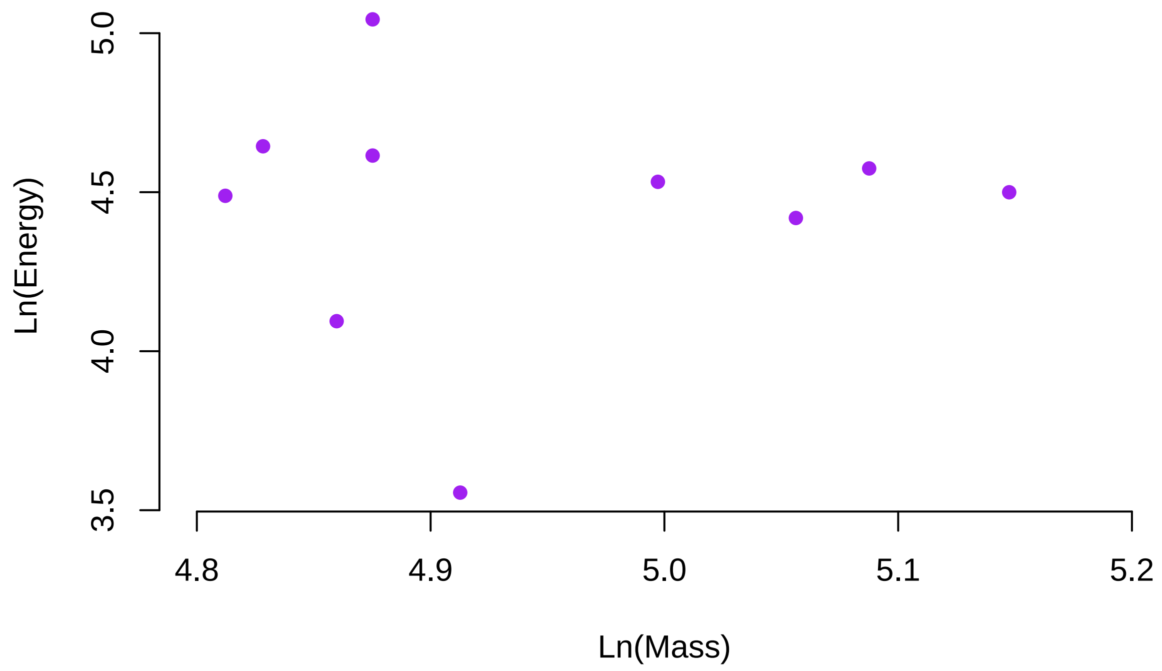
```
#Remove boxes around a plot:

plot(lnenergy~lnmass, data=MoleRats, pch=16, col="purple", bty="n") #bty="n" removes box around the plo
```



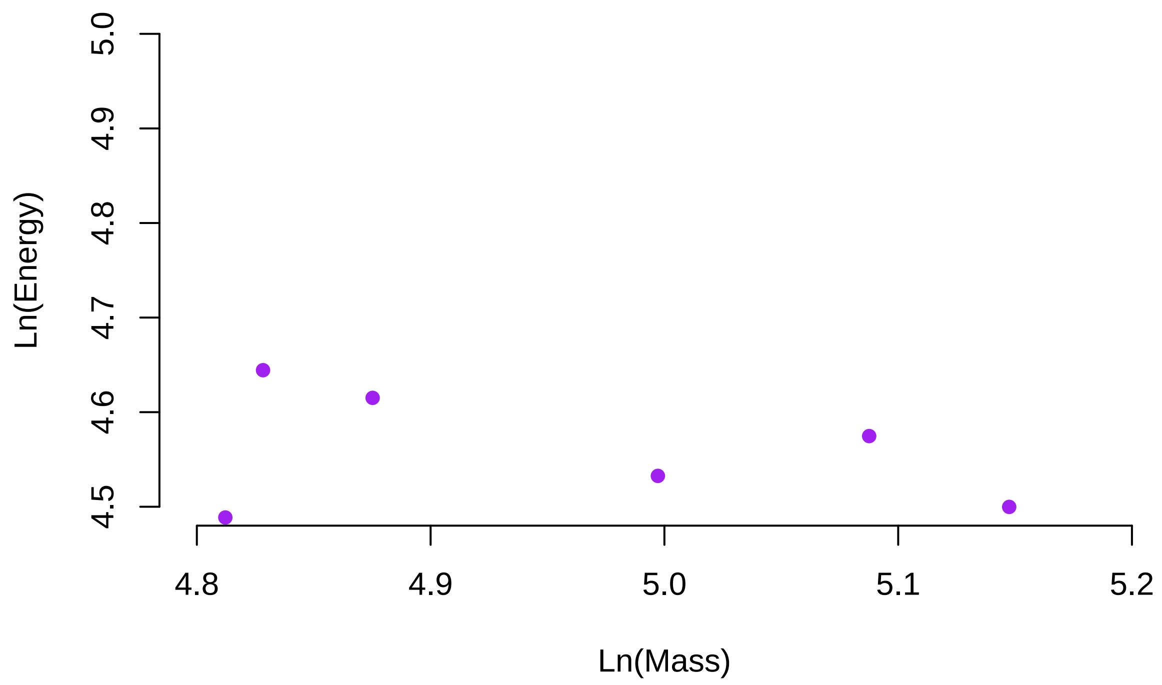
```
#Add axis titles
plot(lnenergy~lnmass, data=MoleRats, pch=16, col="purple", bty="n", ylab="Ln(Energy)", xlab="Ln(Mass)")
```



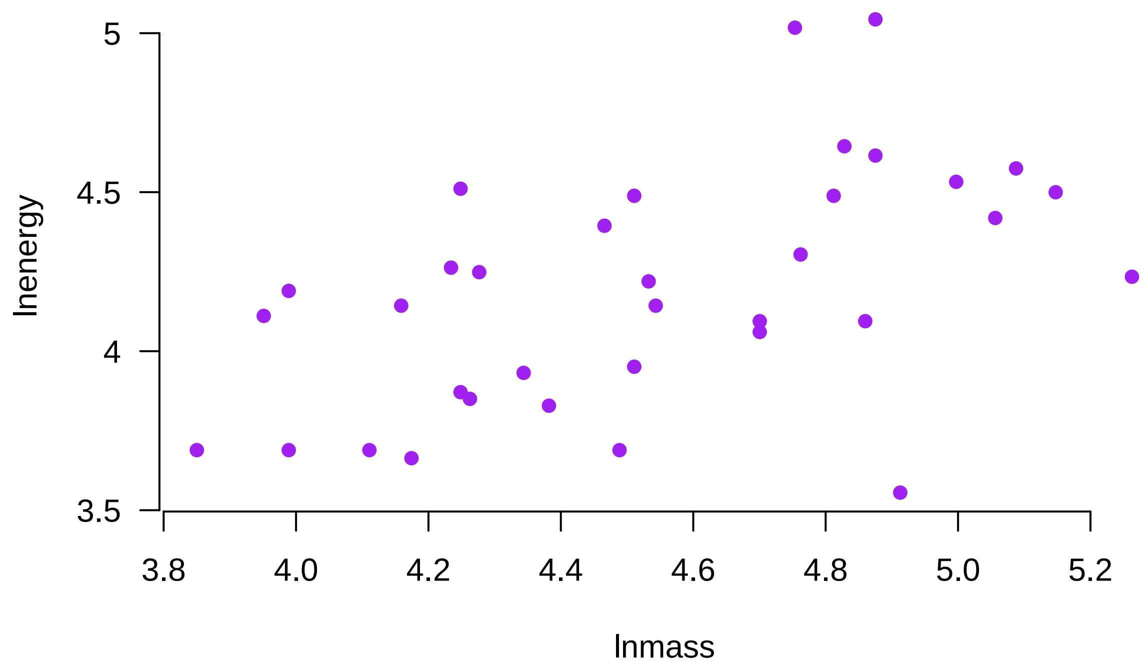
```
#Change axis limits
plot(lnenergy~lnmass, data=MoleRats, pch=16, col="purple", bty="n", ylab="Ln(Energy)", xlab="Ln(Mass)",
```



```
plot(lnenergy~lnmass, data=MoleRats, pch=16, col="purple", bty="n", ylab="Ln(Energy)", xlab="Ln(Mass)",
```



```
#Remove and substitute axes:
plot(lnenergy~lnmass, data=MoleRats, pch=16, col="purple", bty="n", yaxt="n") #yaxt="n" removes the y-a
axis(2, labels=seq(3.5,5.0,0.5), at=seq(3.5,5.0,0.5), las=2) #adds the y-axis back (x-axis is "1", y-ax
```

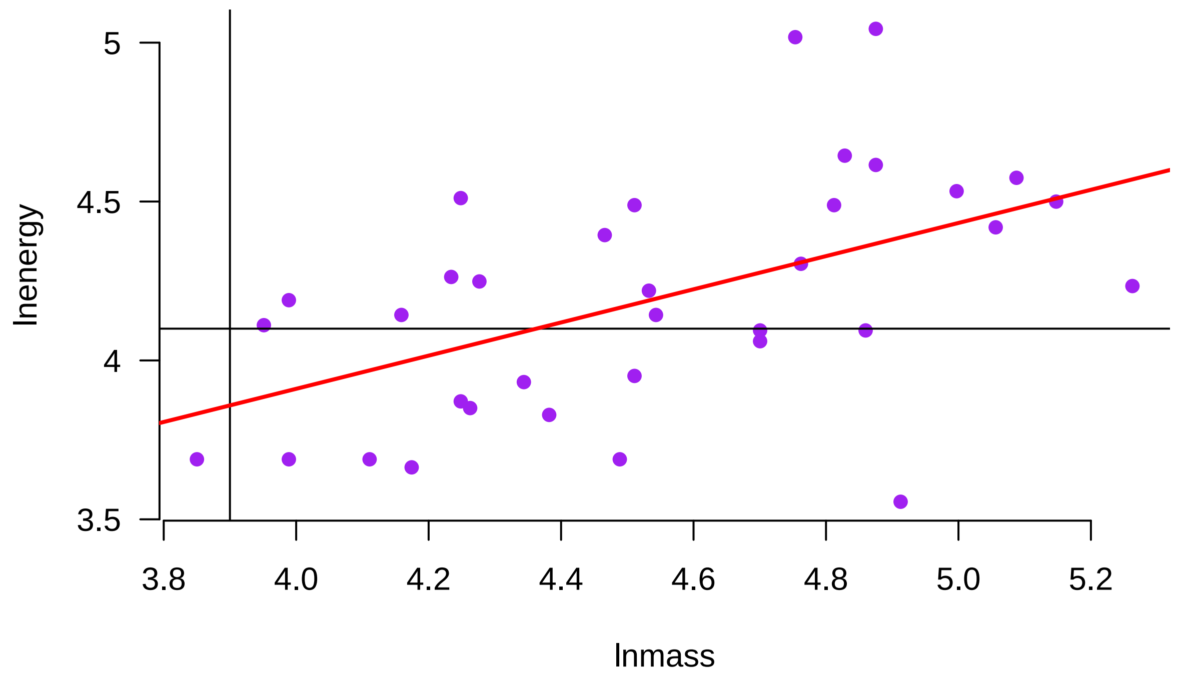


Add horizontal, vertical, regression lines

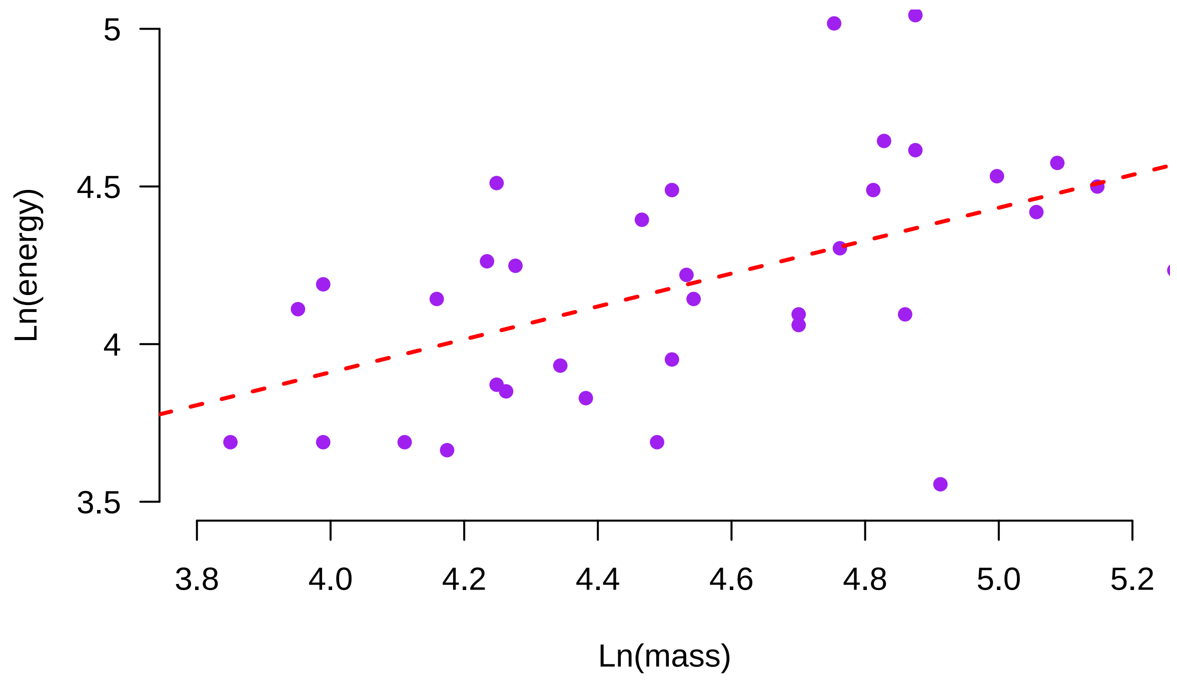
abline adds lines to a plot. Use h for a horizontal line and v for a vertical line. To a regression line, place the regression object (see regression below) the abline().

lwd changes the thickness of a line (higher numbers are thicker) lty changes the type of line (eg 2 is dashed). Examples: http://www.sthda.com/english/wiki/line-types-in-r-lty

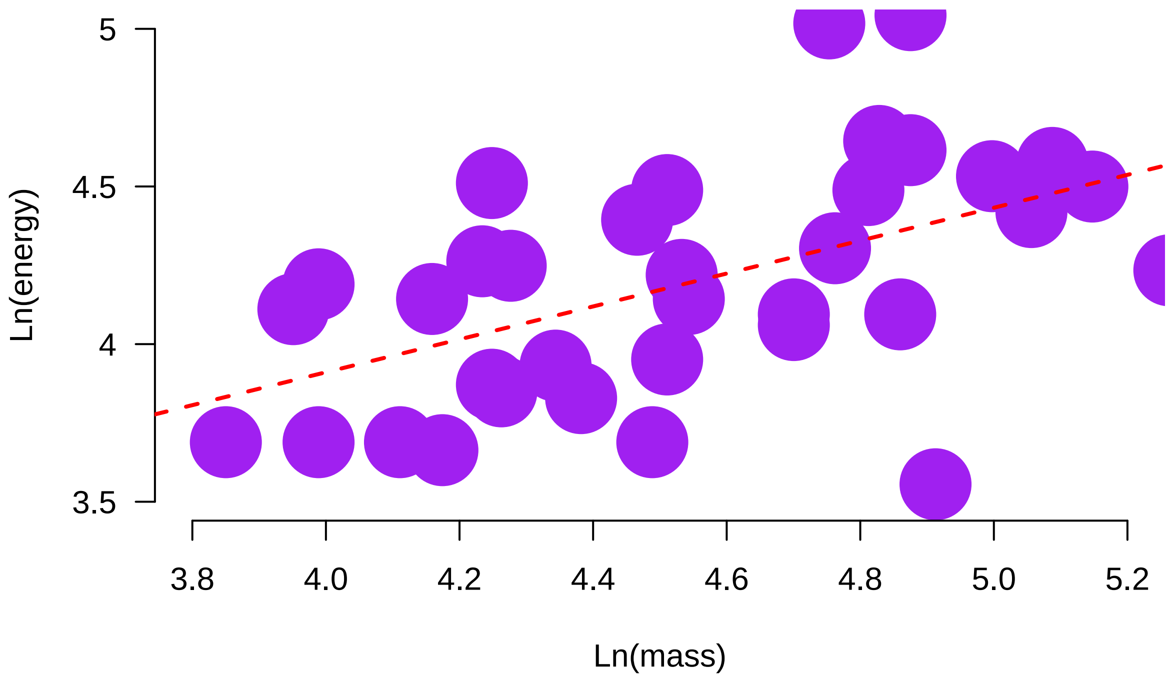
```r
plot(lnenergy~lnmass, data=MoleRats, pch=16, col="purple", bty="n", yaxt="n")
axis(2, labels=seq(3.5,5.0,0.5), at=seq(3.5,5.0,0.5), las=2)
abline(h=4.1) #adds horizontal line at x=4.1
abline(v=3.9) #adds vertical line at y=3.9
abline(lm(lnenergy~lnmass, data=MoleRats), col="red", lwd=2) #adds red regression line. Use lwd to chan
```



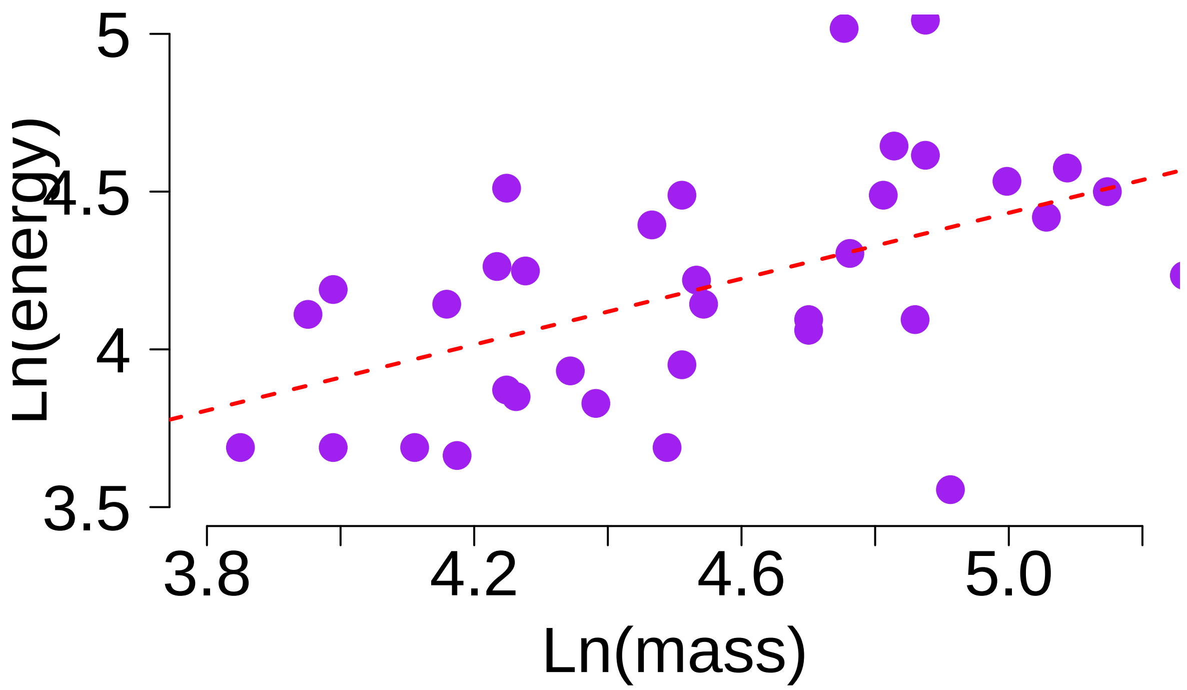
```r
plot(lnenergy~lnmass, data=MoleRats, pch=16, col="purple", bty="n", yaxt="n", ylab="Ln(energy)", xlab="
axis(2, labels=seq(3.5,5.0,0.5), at=seq(3.5,5.0,0.5), las=2)
abline(lm(lnenergy~lnmass, data=MoleRats), col="red", lwd=2, lty=2) #Use lty to change line type (1 is
```



```
#Use cex to change the size of elements
plot(lnenergy~lnmass, data=MoleRats, pch=16, col="purple", bty="n", yaxt="n", ylab="Ln(energy)", xlab="I
axis(2, labels=seq(3.5,5.0,0.5), at=seq(3.5,5.0,0.5), las=2)
abline(lm(lnenergy~lnmass, data=MoleRats), col="red", lwd=2, lty=2)
```



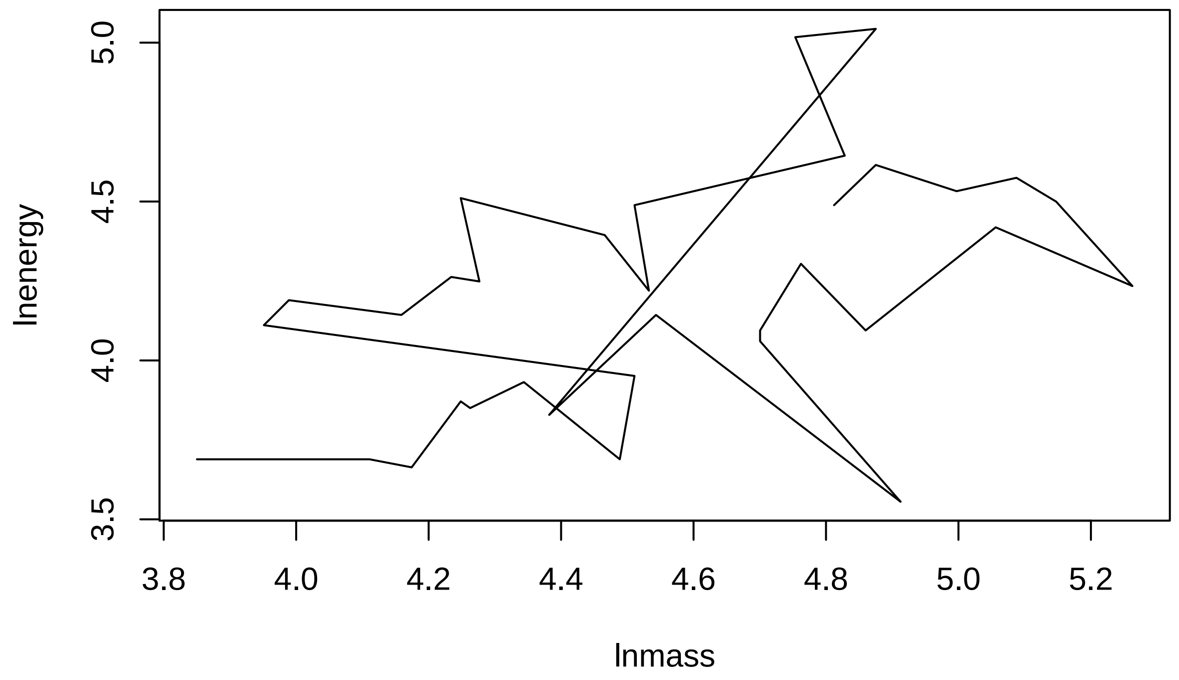
```
plot(lnenergy~lnmass, data=MoleRats, pch=16, col="purple", bty="n", yaxt="n", ylab="Ln(energy)", xlab="
axis(2, labels=seq(3.5,5.0,0.5), at=seq(3.5,5.0,0.5), las=2, cex.axis=2)
abline(lm(lnenergy~lnmass, data=MoleRats), col="red", lwd=2, lty=2)
```



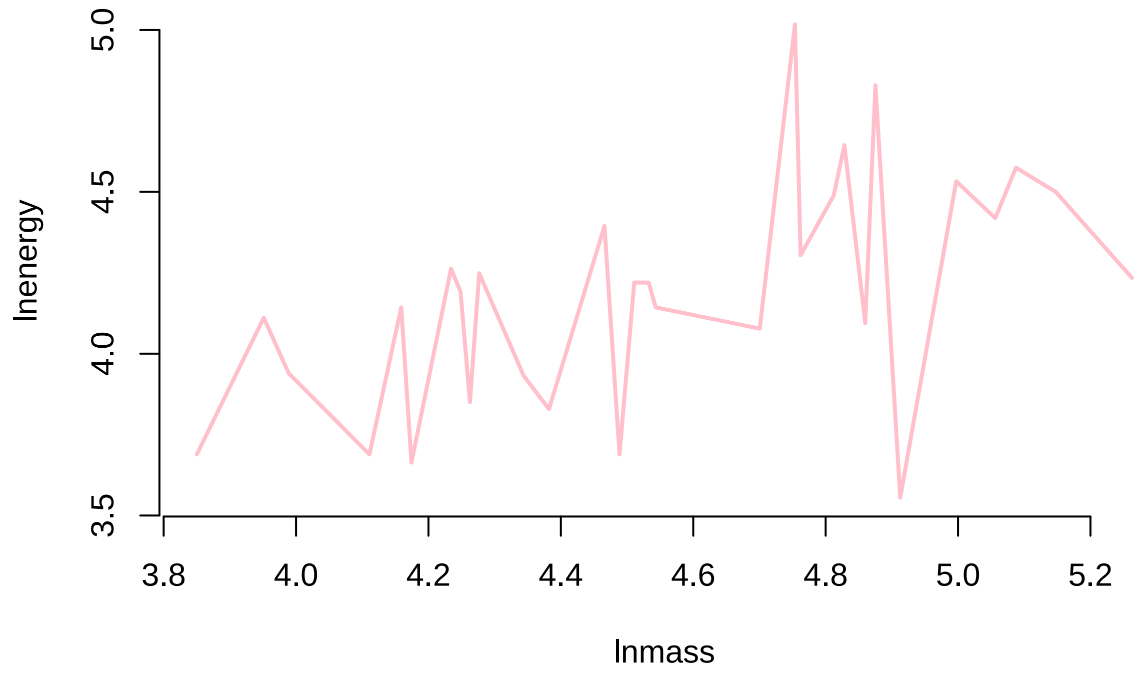
Line plot

If you want lines instead of points, remove pch and add type="l" to your plot.

```r
plot(lnenergy~lnmass, data=MoleRats, type="l")
```



```
#As you can see this does not work well with multiple points at each value of x.

#A line plot of averages:
means.rat<-aggregate(lnenergy~lnmass, data=MoleRats, mean)

plot(lnenergy~lnmass, data=means.rat, type="l", col="pink", lwd=2, bty="n")
```



```
#use col to change colour, lwd to change line thickness, lty to change line type
```

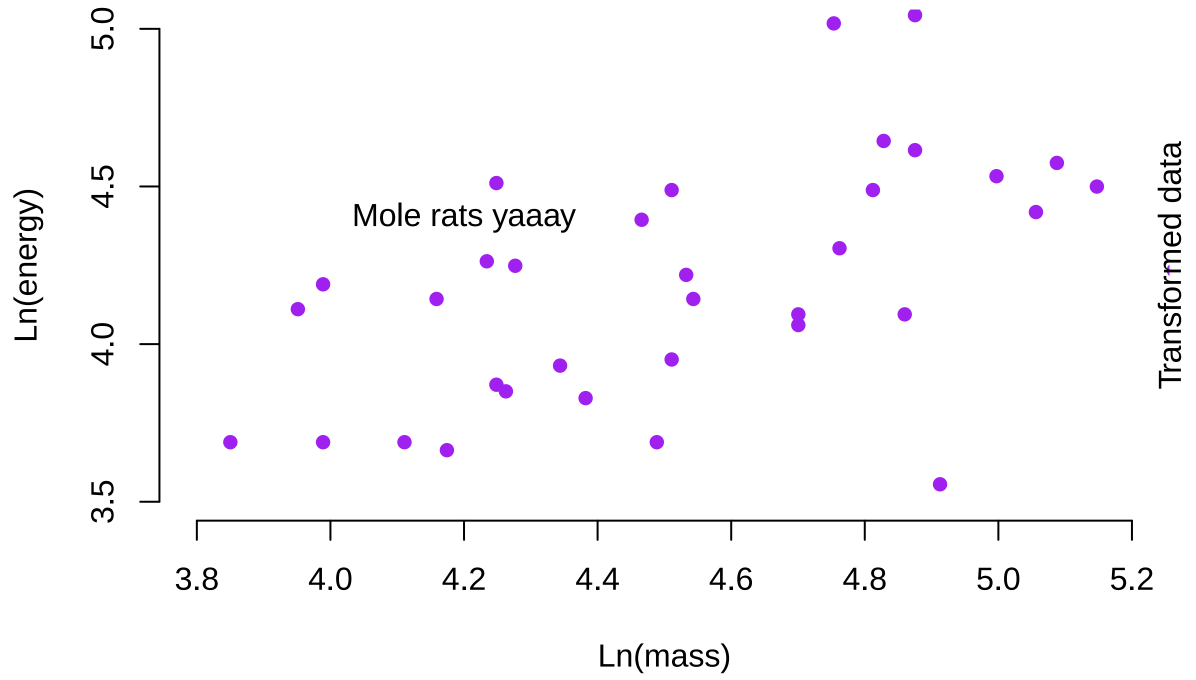
Adding text to plots

Use mtext() to add text to the sides of a plot, and text() to add text to the body of a plot (ie on the plot itself).

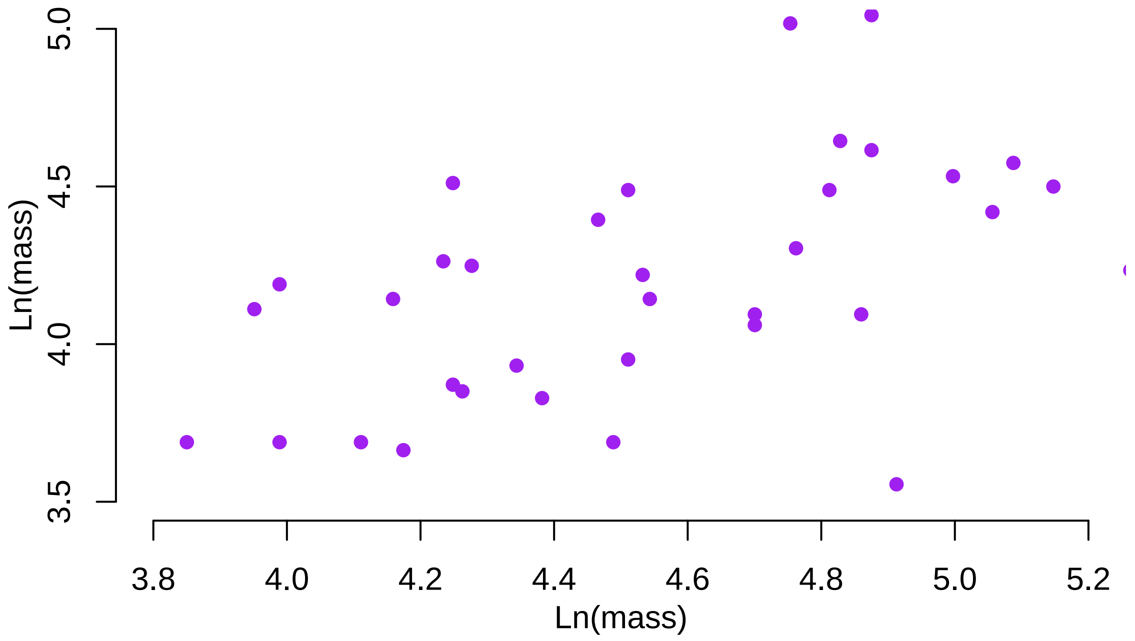
You specify side in mtext(). Use mtext(side=1) for x-axis, mtext(side=2) for y-axis, mtext(side=3) for the top of a plot, mtext(side=4) for the second y-axis on the right of the plot. Use line to move the text up or down (relative to reading direction) - 0 is default; positive numbers move it up from this default and negative numbers move the text down. Use adj to move the text left or right - adj=0 is furthest left, adj=1 is furthest right, and adj=0.5 is centered.

Use text() to add text to particular spot in a plot. The first number in text() should be the x coordinate and the second number the y coordinate.

```
plot(lnenergy~lnmass, data=MoleRats, pch=16, col="purple", bty="n", ylab="Ln(energy)", xlab="Ln(mass)",
mtext("Transformed data", side=4, line=-0.5, adj=0.5) #side indicates which side of the plot (1 = x-axi
text(4.2,4.4, "Mole rats yaaay") #places text at the specified location (x,y) - in this case 4.2 on the
```



```
#Can use cex argument to change text size.

#play with side, line, and adj to see what happens.

#If you do not like the position of the default y- or x-axis title, you can also make it with mtext(), y
plot(lnenergy~lnmass, data=MoleRats, pch=16, col="purple", bty="n", ylab="", xlab="", ylim=c(3.5,5), xli
mtext("Ln(mass)", side=1, line=2, adj=0.5) #x-axis title
mtext("Ln(mass)", side=2, line=2, adj=0.5) #y-axis title
```



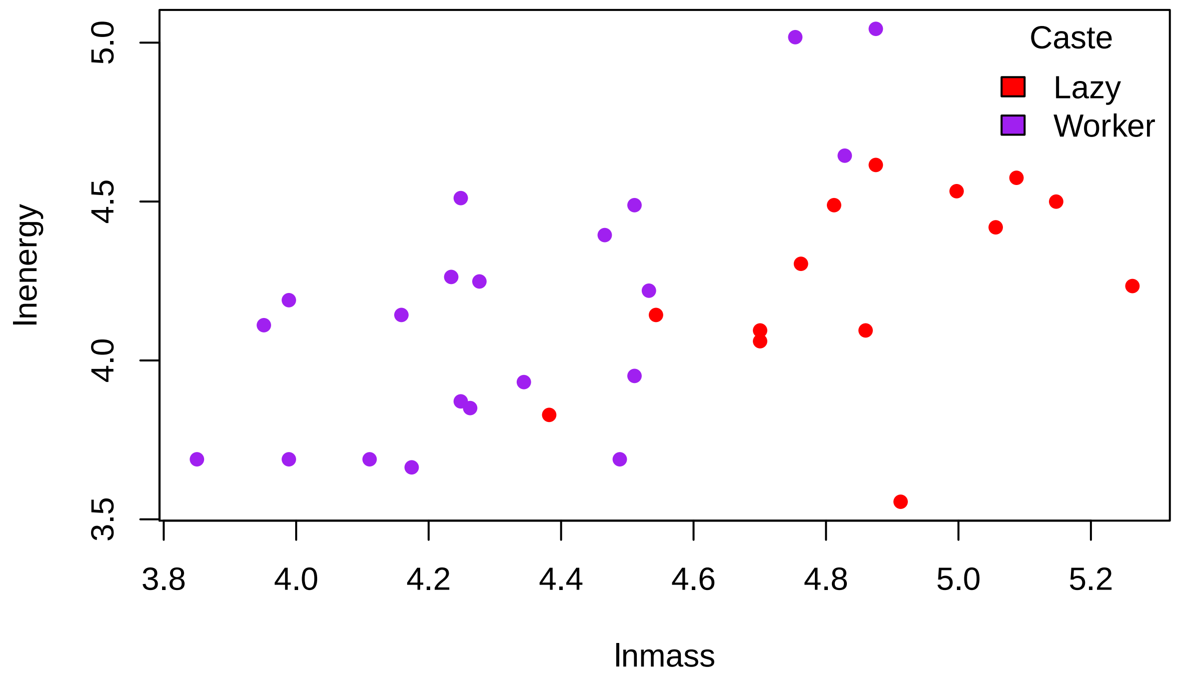
Add a legend

Use legend() after your plot to add a legend.

You specify the text (ie the labels), and then pch for point types and col for colours if you have points, or just fill for the colours if you have boxes/bars filled with colour rather than points.

The first thing to go after the brackets will be the legend position, either coordinates (ie 4,5 for x=4, y=5) or "bottomright", "bottom", "bottomleft", "left", "topleft", "top", "topright", "right" and "center".

```
colours<-c("red", "purple")#define colours you want to use
plot(lnenergy~lnmass, data=MoleRats, pch=16, col=colours[as.numeric(MoleRats$caste)]) #more customizabl
legend("topright", c("Lazy", "Worker"), fill=colours, bty="n", title="Caste") #use bty="n" to get rid o
```



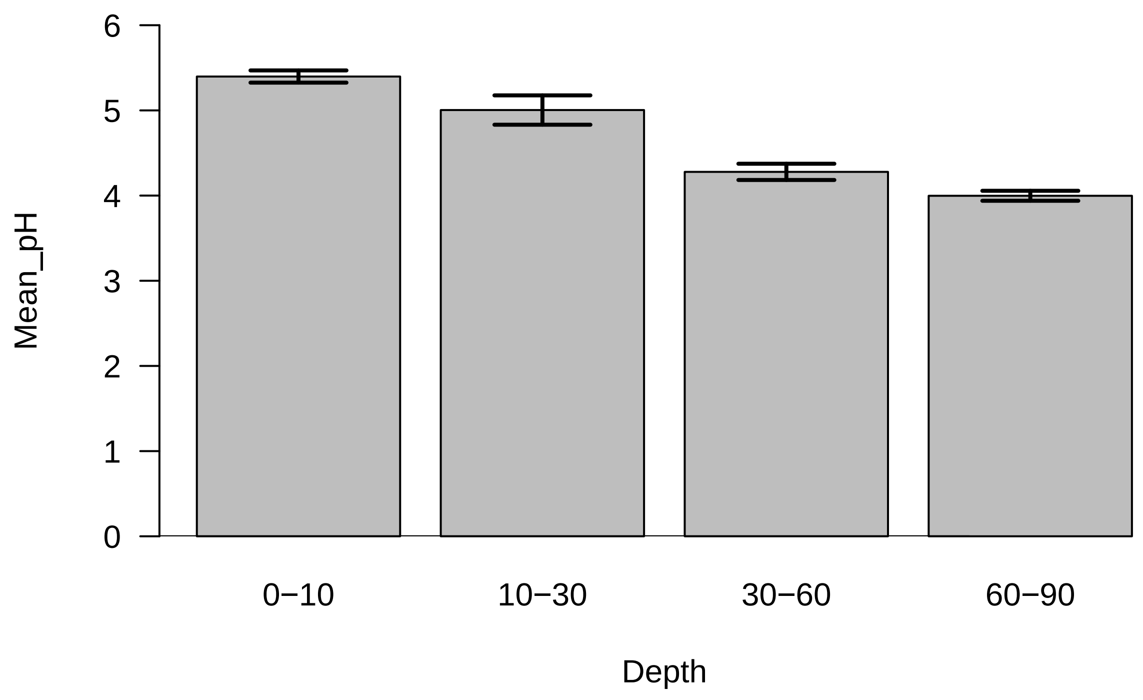
```
#Can use cex argument to change text size
```

Bar plot

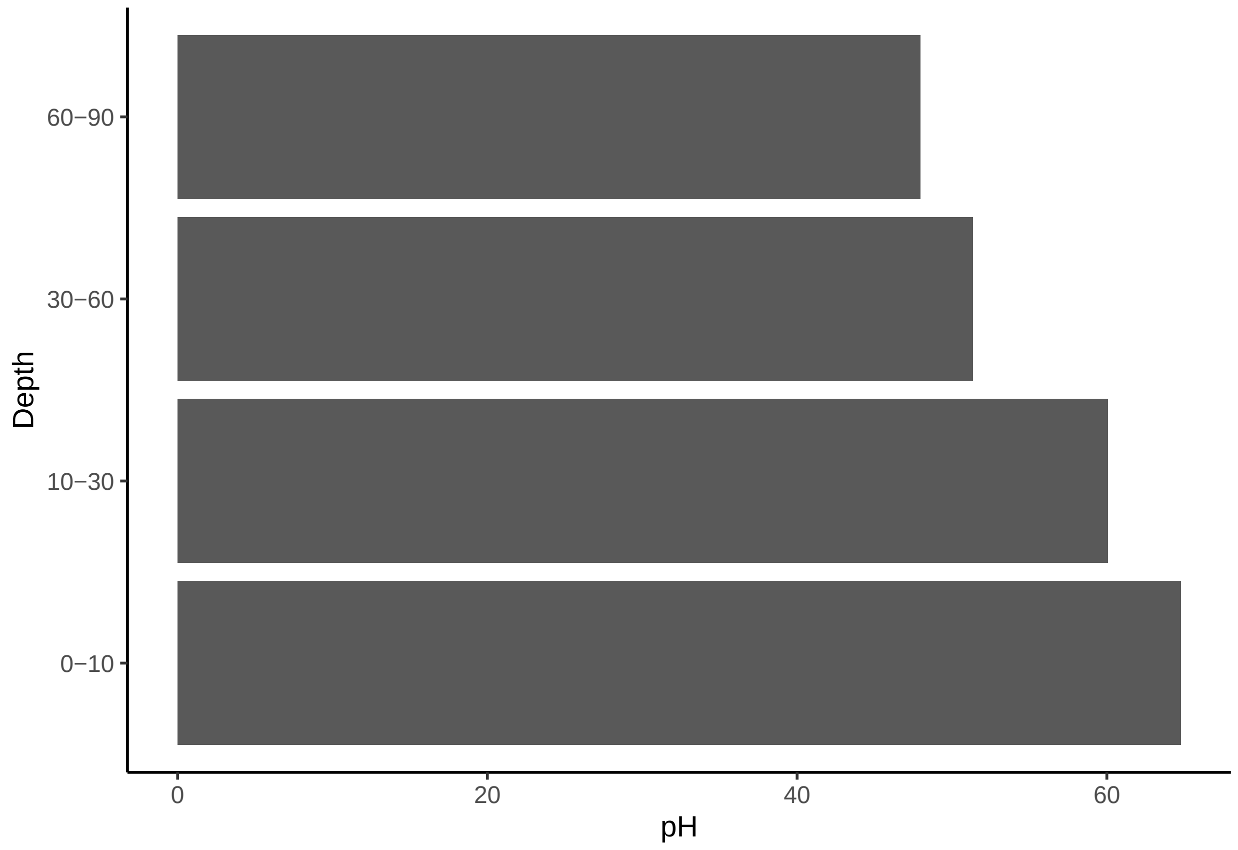
```
#Example data:
library(carData)
data(Soils)

#Simple bar chart:

#First calculate summary statistic to be plotted (eg mean) with error (eg standard error):
#Calculate mean and standard error:
data_summary<-aggregate(pH~Depth, Soils, mean)
data_summary$SE_pH<-aggregate(pH~Depth, Soils, function(x) sd(x)/sqrt(length(x)))[,2]
names(data_summary)<-c("Depth", "Mean_pH", "SE_pH") #make sure the data frame has sensible column names

#Use base plotting to plot the summary statistic (eg mean):
b<-barplot(Mean_pH~Depth, data=data_summary, ylim=c(0,6), yaxt="n") #I stopped the function from making
axis(2, labels=0:6, at=0:6, las=2) #y-axis is axis 2 in R. las=2 rotates the labels 90. degrees for bet
#Add error bars:
arrows(x0=b, y0=data_summary$Mean_pH-data_summary$SE_pH, x1=b, y1=data_summary$Mean_pH+data_summary$SE_
segments(0,0,4,0) #draws a line for the x-axis (ie a line from (0,0) to (4,0))
```

```r
#Use ggplot to plot the summary statistic (eg mean):
library(ggplot2)
```

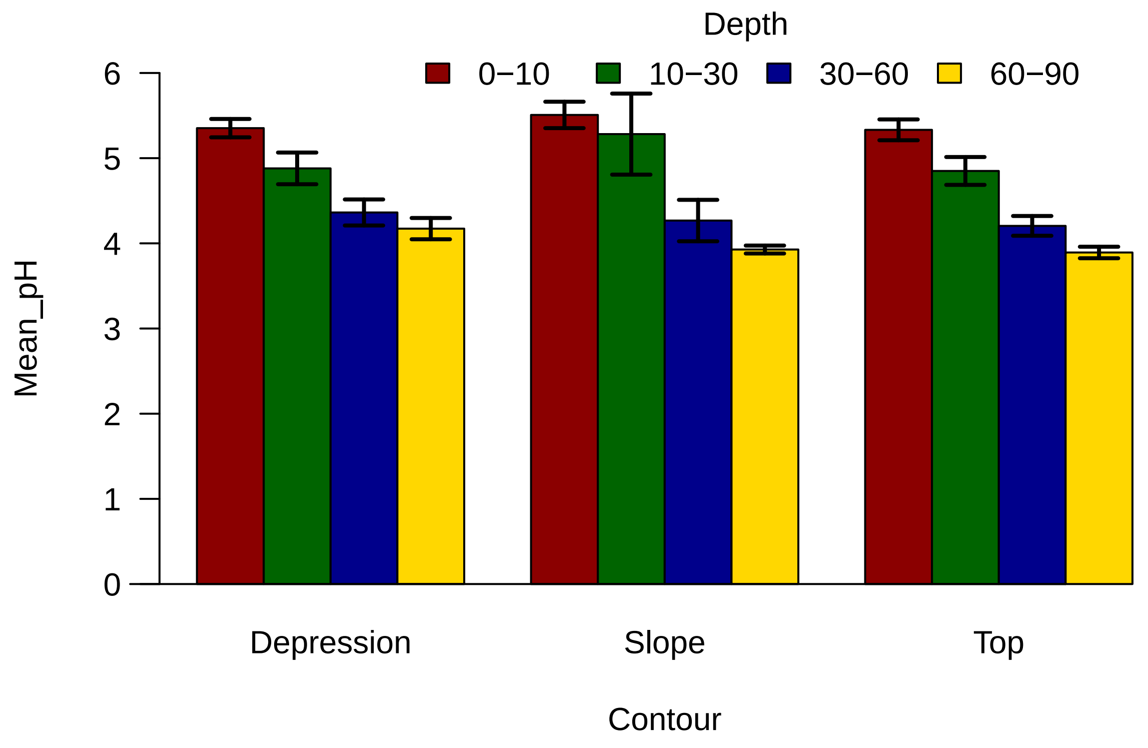
```
## Warning: package 'ggplot2' was built under R version 4.0.3
```



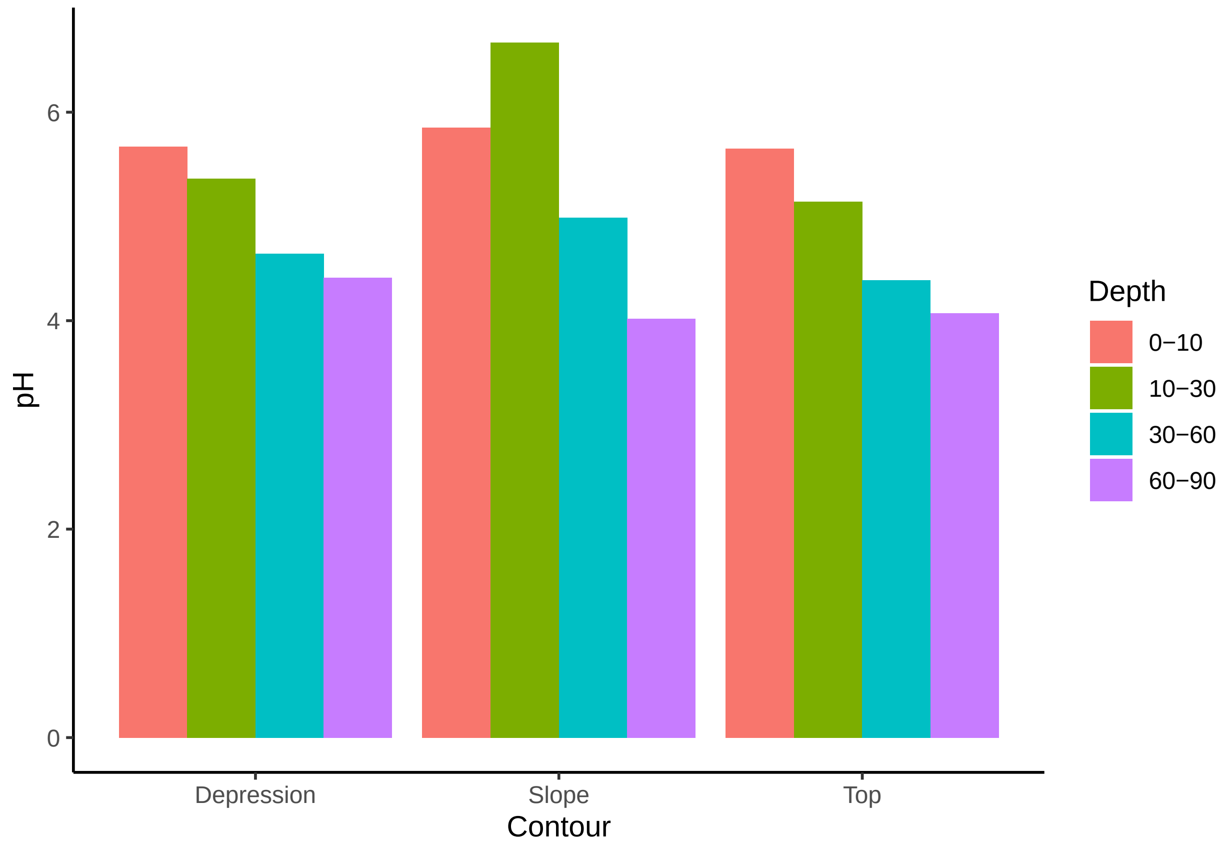
```r
g<-ggplot(Soils,aes(x=Depth, y=pH)) + #specifies the data and x and y variables. NOTE: ggplot takes the
    geom_bar(stat="identity") + #specifies the type of plot (a bar plot).
   theme_classic() #makes plot look professional


#can also make horizontal:
g + coord_flip()
```



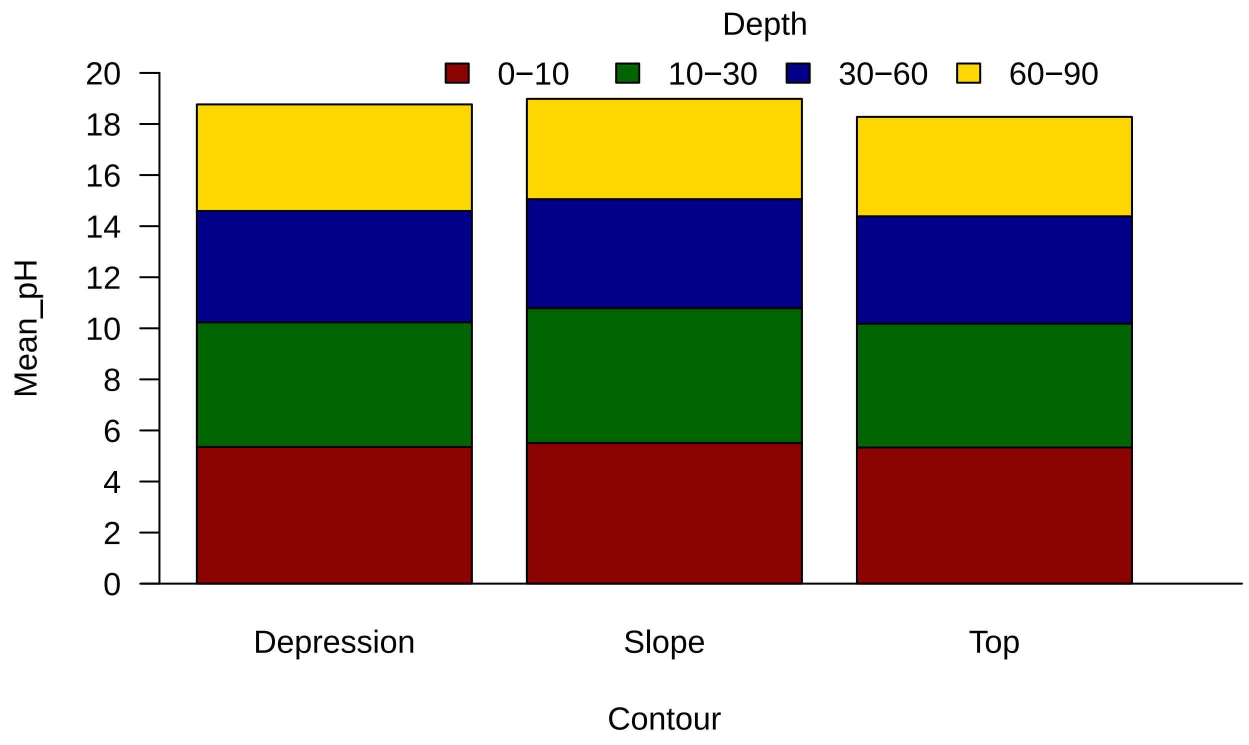
```
#Add error bars:
p <- ggplot(data_summary, aes(x=Depth, y=Mean_pH)) +
   geom_bar(stat="identity") +
  geom_errorbar(aes(ymin=Mean_pH-SE_pH, ymax=Mean_pH+SE_pH), width=.2) + #geom_errorbar adds error bars
  theme_classic()
###################################
#Bar charts of multiple variables:
##################################
#Calculate mean and standard error:
data_summary2<-aggregate(pH~Depth+Contour, Soils, mean)
data_summary2$SE_pH<-aggregate(pH~Depth+Contour, Soils, function(x) sd(x)/sqrt(length(x)))[,3] #SE colu
names(data_summary2)<-c("Depth","Contour", "Mean_pH", "SE_pH") #make sure the data frame has sensible c

#Bars side by side base plot:
par(xpd = TRUE) #xpd=TRUE lets me draw outside of plotting window - needed to make legend fit
b<-barplot(Mean_pH~Depth+Contour, beside=TRUE, data=data_summary2, ylim=c(0,6), col=c("darkred", "darkg
axis(2, labels=0:6, at=0:6, las=2) #y-axis is axis 2 in R. las=2 rotates the labels 90. degrees for bet
#Add error bars:
arrows(x0=b, y0=data_summary2$Mean_pH-data_summary2$SE_pH, x1=b, y1=data_summary2$Mean_pH+data_summary2
segments(0,0,14,0) #draws a line for the x-axis (ie a line from (0,0) to (14,0))
legend(4,6.9, c("0-10", "10-30", "30-60", "60-90"), fill=c("darkred", "darkgreen", "darkblue", "gold"),
```



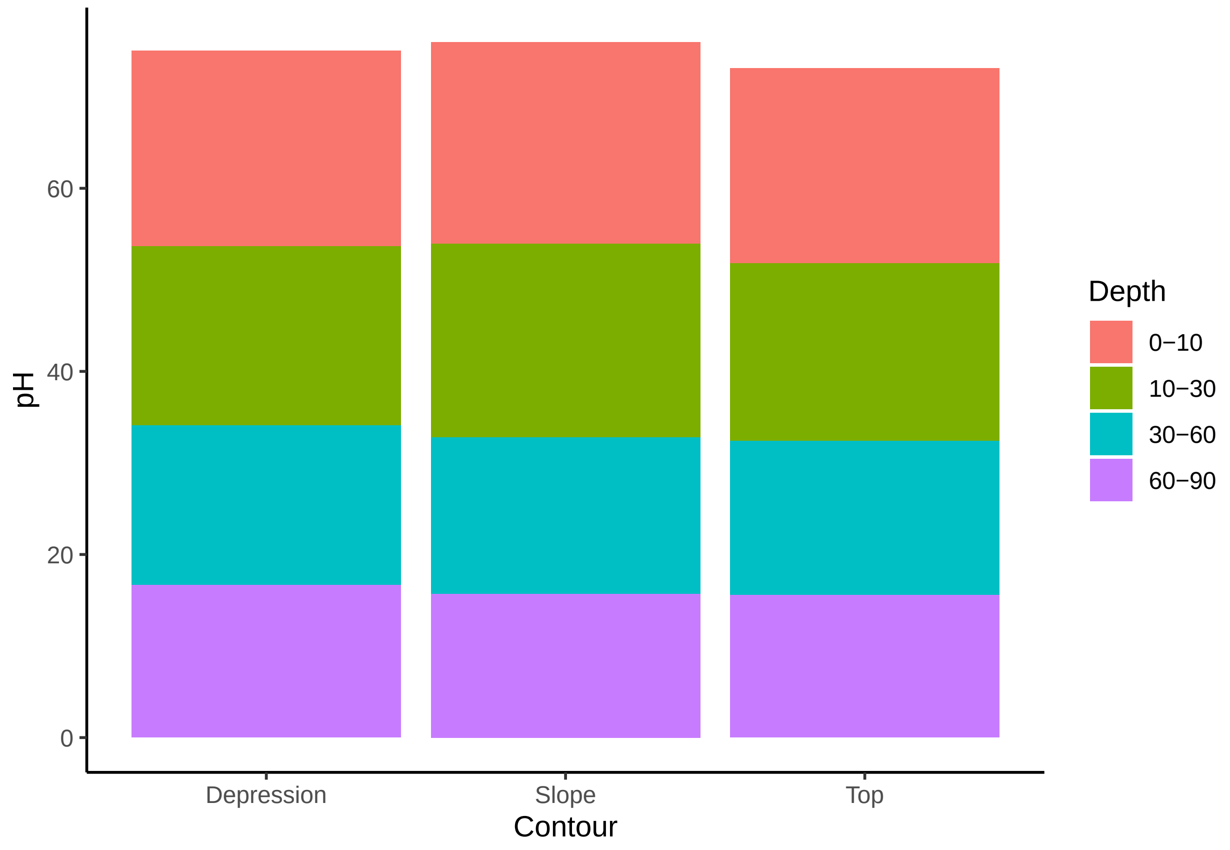
```
#Bars side by side ggplot:
# Use position=position_dodge()
ggplot(data=Soils, aes(x=Contour, y=pH, fill=Depth)) +
geom_bar(stat="identity", position=position_dodge()) + #position=position_dodge() specifies side by sid
  theme_classic()
```



```
#Bars stacked base plot:
par(xpd = TRUE) #xpd=TRUE lets me draw outside of plotting window - needed to make legend fit
b<-barplot(Mean_pH~Depth+Contour, beside=FALSE, data=data_summary2, ylim=c(0,20), col=c("darkred", "dar
axis(2, labels=seq(0,20,2), at=seq(0,20,2), las=2) #y-axis is axis 2 in R. las=2 rotates the labels 90.
#Add error bars:
segments(0,0,4,0) #draws a line for the x-axis (ie a line from (0,0) to (4,0))
legend(1,23, c("0-10", "10-30", "30-60", "60-90"), fill=c("darkred", "darkgreen", "darkblue", "gold"), 
```

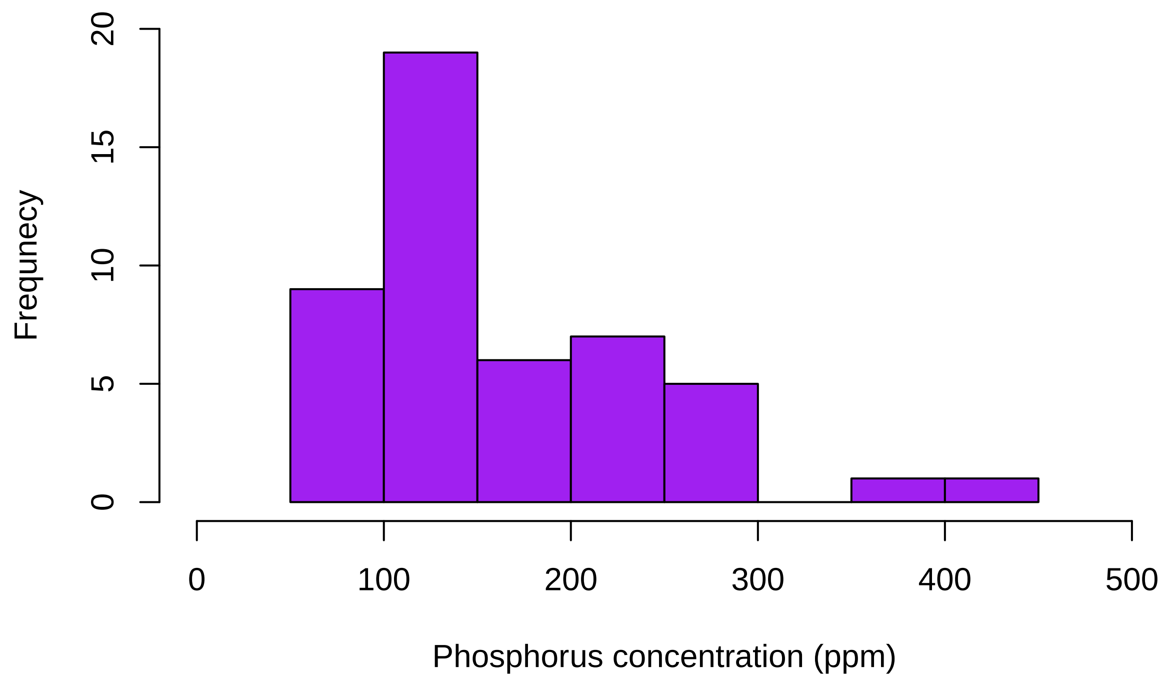


```
#Bars stacked ggplot:
# Use position=position_dodge()
ggplot(data=Soils, aes(x=Contour, y=pH, fill=Depth)) +
geom_bar(stat="identity", position=position_stack()) + #position=position_dodge() specifies side by sid
  theme_classic()
```



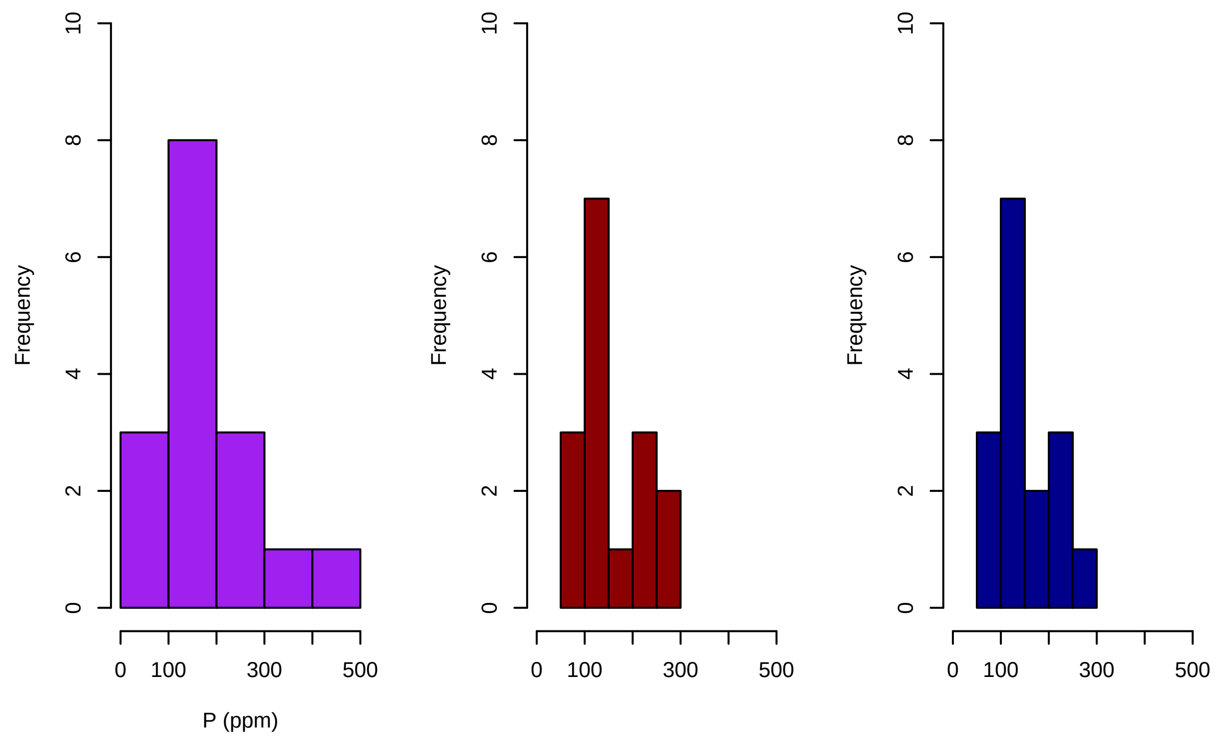
Histograms

```
#make a histogram of soil phosphorus:
hist(Soils$P, col="purple", ylab="Frequnecy",
    xlab="Phosphorus concentration (ppm)", main="", xlim=c(0,500), ylim=c(0,20))
```

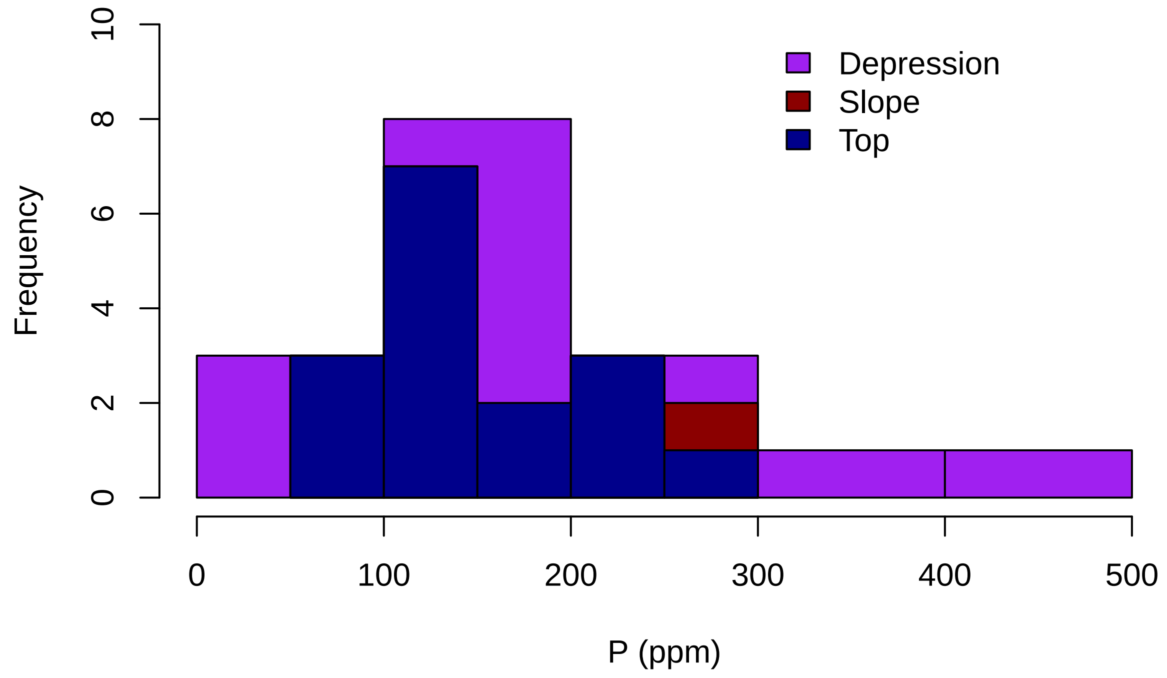


Histograms by group:

```
#subset to get a data frame for each group:
Depression <- subset(Soils, Contour=="Depression")
Slope <- subset(Soils, Contour=="Slope")
Top <- subset(Soils, Contour=="Top")

#all in separate panels of the same plot
par(mfrow=c(1,3)) #This makes my plots appear in a grid
#(1 by 3 - change the numbers in () if want different, ie (2,2) is 2x2)

#histogram for depression type contour
hist(Depression$P, col="purple", ylim = c(0,10),
     xlim=c(0,500), xlab="P (ppm)", main="")
#histogram for slope type contour
hist(Slope$P, col="darkred", ylim = c(0,10),
     xlim=c(0,500), xlab="", main="")
#histogram for top type contour
hist(Top$P, col="darkblue", ylim = c(0,10),
     xlim=c(0,500), xlab="", main="")
```



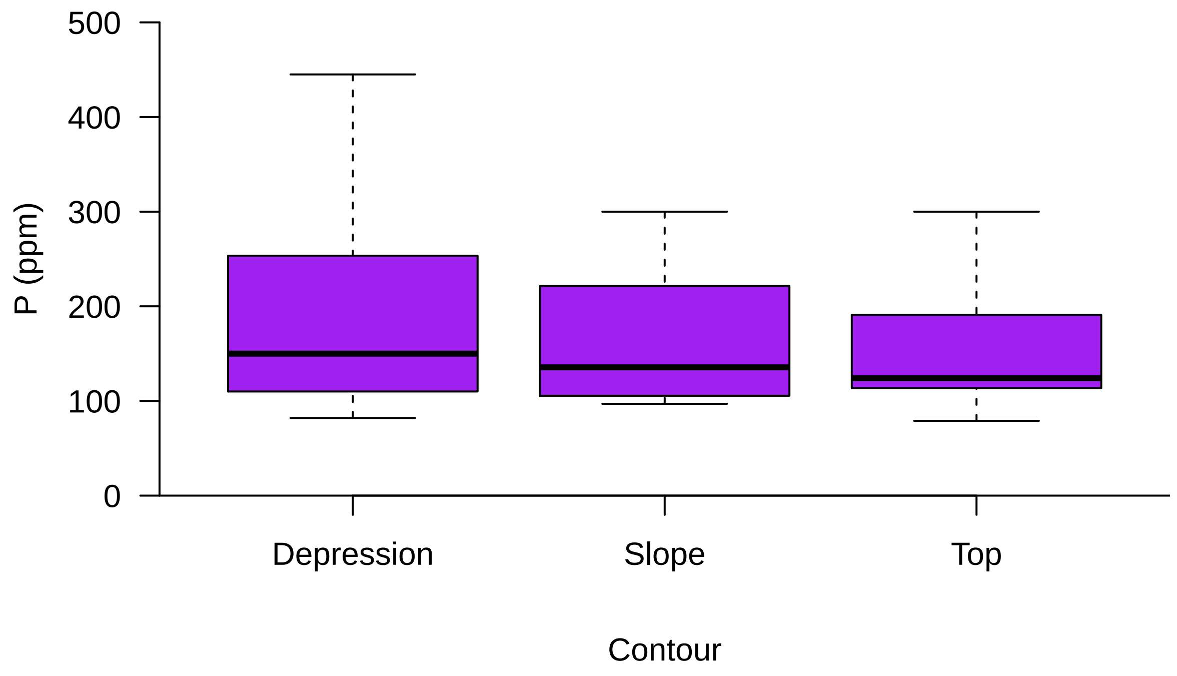
```
par(mfrow=c(1,1)) # return to just a single panel per plot
#all in one plot:
hist(Depression$P, col="purple", ylim = c(0,10),
     xlim=c(0,500), xlab="P (ppm)", main="")
hist(Slope$P, col="darkred", ylim = c(0,10),
     xlim=c(0,500), xlab="", main="", add=TRUE)
hist(Top$P, col="darkblue", ylim = c(0,10),
     xlim=c(0,500), xlab="", main="", add=TRUE)
legend(300,10, c("Depression", "Slope", "Top"),
       fill=c("purple", "darkred", "darkblue"), bty="n") # make a legend at x=300, y=10
```



```
#add=TRUE adds plots below to the plot above
```

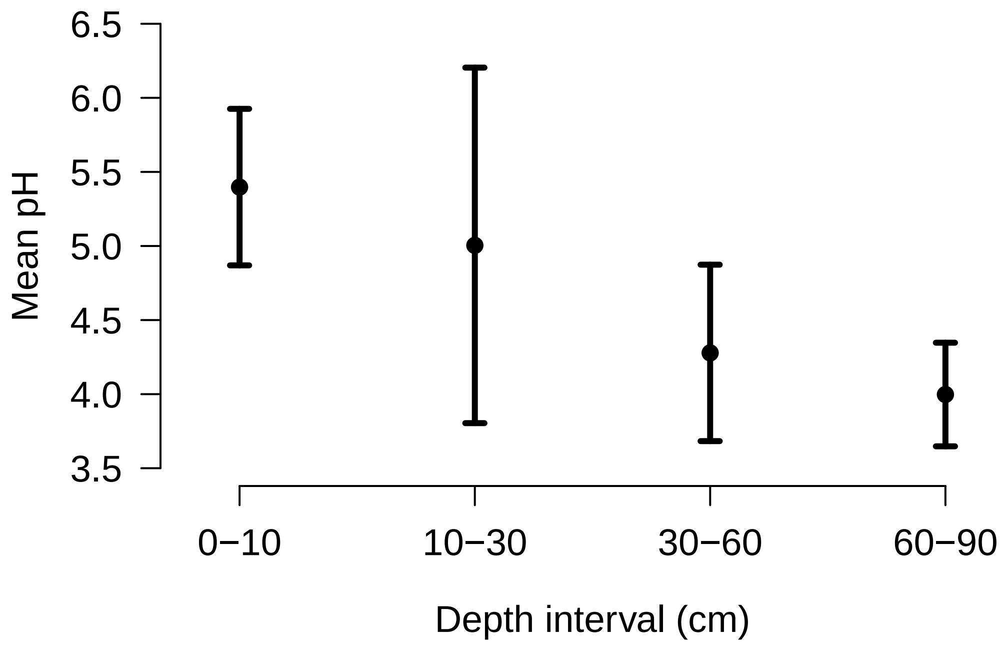
Boxplots

```
#make a boxplot of soil phosphorus by contour type:
boxplot(P~Contour, data=Soils, col="purple", frame=F, ylim=c(0,500), axes=F, ylab="P (ppm)", xlab="Conto
#frame=F removes box around the plot in a boxplot

axis(1, labels=levels(Soils$Contour), at=1:3, pos=0) #1 is for x-axis, 2 is for y-axis.
#levels(Contour) gives me the levels of my grouping variable (Contour) which I want as labels for my x-
#at=1:3 means put the labels at positions 1,2, and 3 on the x-axis (ie where the boxes are for each gro
#pos=0 puts the x-axis at zero on the y-axis (rather than slightly below)

axis(2, labels=seq(0,500,100), at=seq(0,500,100), las=2) #las=2 turns the labels 90 degrees so they go
#seq() means from first number to second by the third number - ie give me all numbers from 0 to 500 goi

segments(0,0,5,0)# adds a line along the x-axis so it intersects with the y-axis and looks nice. Argume
```

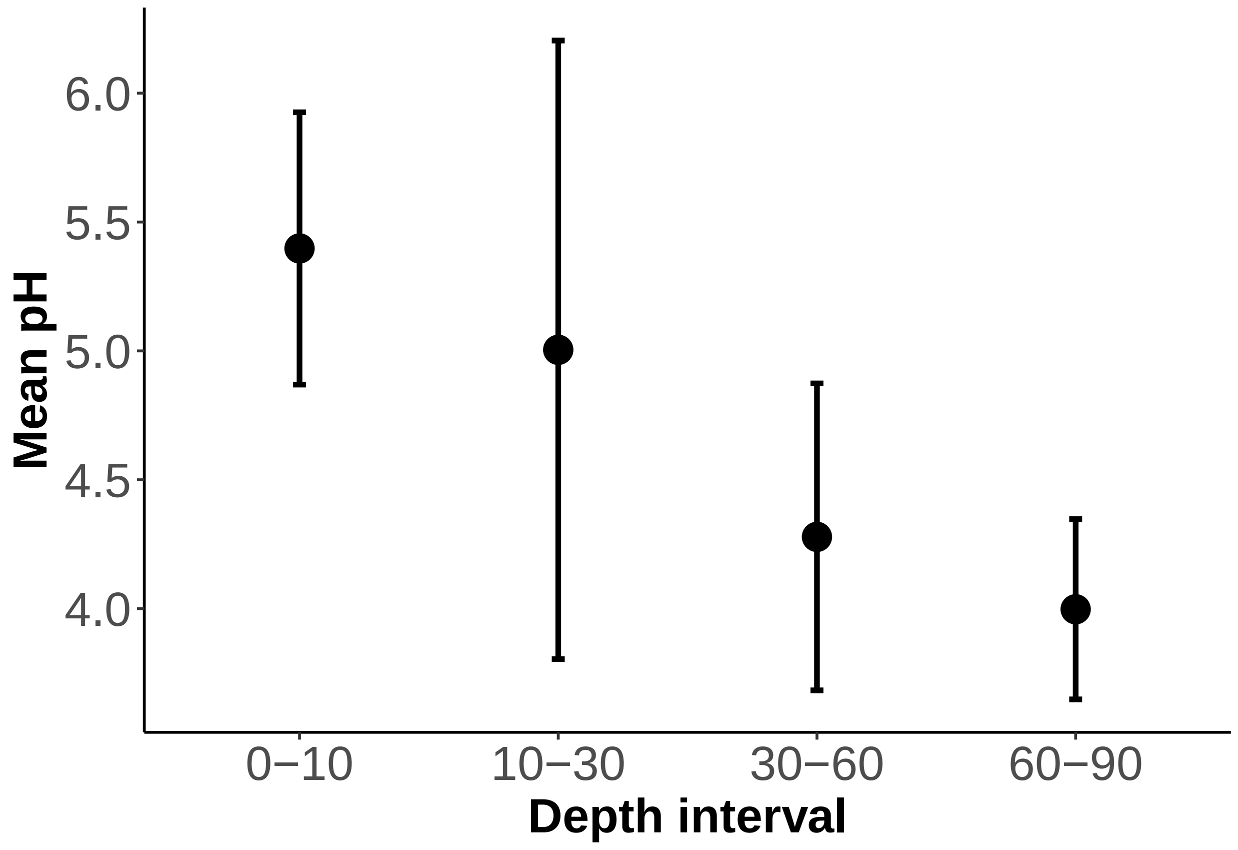


Interval plots

```
#Two methods:

#using base R:
#use cex (and cex.axis, cex.lab) to change size of things - bigger numbers make bigger text and thicker
par(mar=c(5,5,5,5)) #makes the plotting window look nice - mar means change the margins of the plot, so

plot(0, type="n", ylab="Mean pH", xlab="Depth interval (cm)", xaxt="n",
      bty="n", cex.axis=1.2, cex.lab=1.2, ylim=c(3.5,6.5), xlim=c(0.8,4.2), las=2)#makes an empty plot
#The points representing the means will go at 1, 2, 3, and 4 along the x-axis to spread them out (and t

#use points function to place means on the plot:
points(Mean_pH~Depth, data=pH_mean_CI, pch=16, cex=1.2)
#When plotting, the categorical Depth will be turned into numbers so R knows where to put the points. T
#pch changes the type of point. Change the number to 17 or 18 or other numbers to see what happens.

axis(1, labels=pH_mean_CI$Depth, at=1:4, cex.axis=1.2)
#use the arrows function (code=3 means use bars as "arrows").
#places the error bars at 1:4 (which means from 1 to 4 going up by 1), same place as the points for the
arrows(1:4, pH_mean_CI$Lower_CI,
       1:4, pH_mean_CI$Upper_CI,
       length=0.05, angle=90, code=3, lwd=3) #lwd changes thickness of the lines.
```



```r
#Using ggplot2:
#install.packages("ggplot2")
library(ggplot2)

ggplot(pH_mean_CI, aes(x = Depth,
                              y = Mean_pH)) +
  geom_errorbar(aes(ymin = Lower_CI,
                    ymax = Upper_CI),
                width = 0.05,
                size  = 1) +
  geom_point(shape = 16, size  = 5) +
  theme_bw() + #remove plot background colour
  theme(axis.title   = element_text(face  = "bold")) +
  ylab("Mean pH") + xlab("Depth interval") +
  #These next lines remove grid lines in plot and box but keep axes
  theme(panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.border = element_blank(),
        axis.line = element_line(colour = "black"),
        axis.text=element_text(size=18),#font size on axis labels
        axis.title=element_text(size=18,face="bold")) #font size on axis title
```



Multi-panel plots

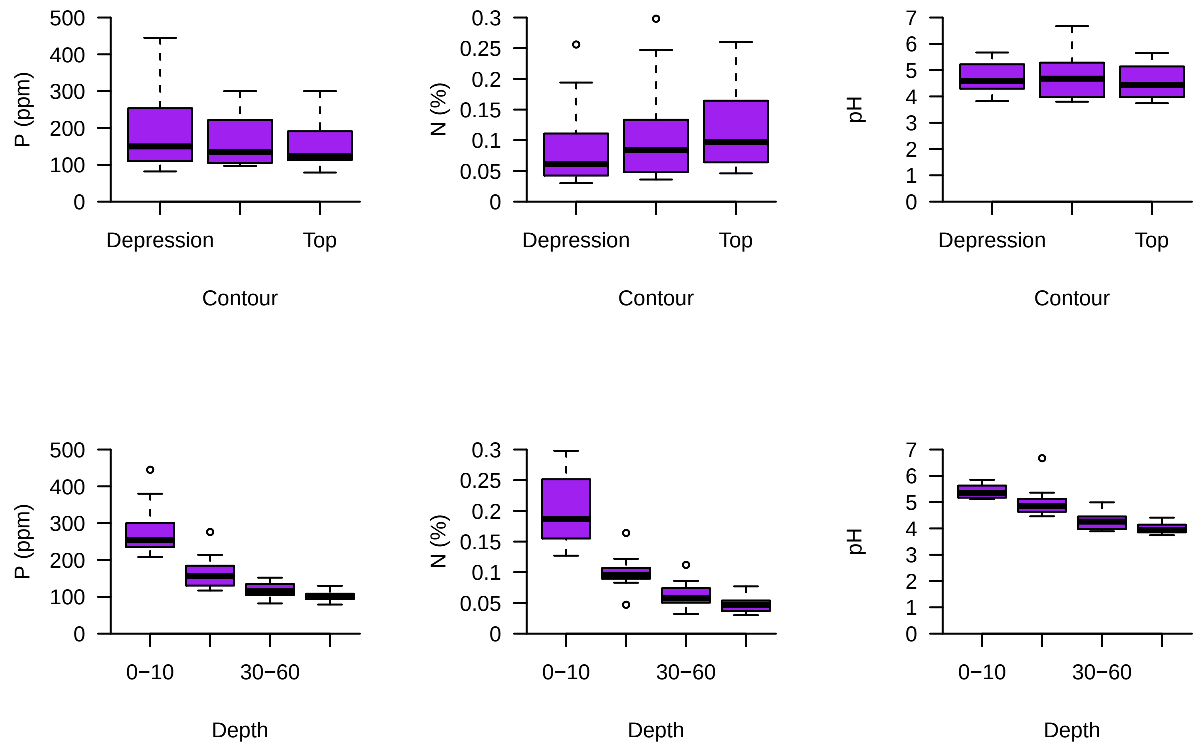
Example using Soils data.

We have pH, N, and P we can plot, by contour and by depth - that can be a 2 by 3 multi-panel plot.

Two ways:

1. Use par(mfrow=c(nrow, ncol)) where nrow is number of rows and ncol is number of columns in your multi-panel.

```
#Use Soils data as example:
library(carData)
data(Soils)

par(mfrow=c(2,3))
#boxplot of P by Contour:
b1<-boxplot(P~Contour, data=Soils, col="purple", frame=F, ylim=c(0,500), axes=F, ylab="P (ppm)", xlab="
axis(1, labels=levels(Soils$Contour), at=1:3, pos=0)
axis(2, labels=seq(0,500,100), at=seq(0,500,100), las=2)
segments(0,0,3.5,0)

#boxplot of N by Contour:
b2<-boxplot(N~Contour, data=Soils, col="purple", frame=F, ylim=c(0,0.3), axes=F,ylab="N (%)", xlab="Con
axis(1, labels=levels(Soils$Contour), at=1:3, pos=0)
axis(2, labels=seq(0,0.3,0.05), at=seq(0,0.3,0.05),las=2)
segments(0,0,3.5,0)

#boxplot of pH by Contour:
b3<-boxplot(pH~Contour, data=Soils, col="purple", frame=F, ylim=c(0,7), axes=F, ylab="pH", xlab="Contou
axis(1, labels=levels(Soils$Contour), at=1:3, pos=0)
axis(2, labels=seq(0,7,1), at=seq(0,7,1), las=2)
segments(0,0,3.5,0)

#boxplot of P by Depth:
b4<-boxplot(P~Depth, data=Soils, col="purple", frame=F, ylim=c(0,500), axes=F, ylab="P (ppm)", xlab="De
axis(1, labels=levels(Soils$Depth), at=1:4, pos=0)
axis(2, labels=seq(0,500,100), at=seq(0,500,100), las=2)
segments(0,0,4.5,0)

#boxplot of N by Depth:
b5<-boxplot(N~Depth, data=Soils, col="purple", frame=F, ylim=c(0,0.3), axes=F,ylab="N (%)", xlab="Depth
axis(1, labels=levels(Soils$Depth), at=1:4, pos=0)
axis(2, labels=seq(0,0.3,0.05), at=seq(0,0.3,0.05),las=2)
segments(0,0,4.5,0)

#boxplot of pH by Contour:
b6<-boxplot(pH~Depth, data=Soils, col="purple", frame=F, ylim=c(0,7), axes=F, ylab="pH", xlab="Depth")
axis(1, labels=levels(Soils$Depth), at=1:4, pos=0)
axis(2, labels=seq(0,7,1), at=seq(0,7,1), las=2)
segments(0,0,4.5,0)
```



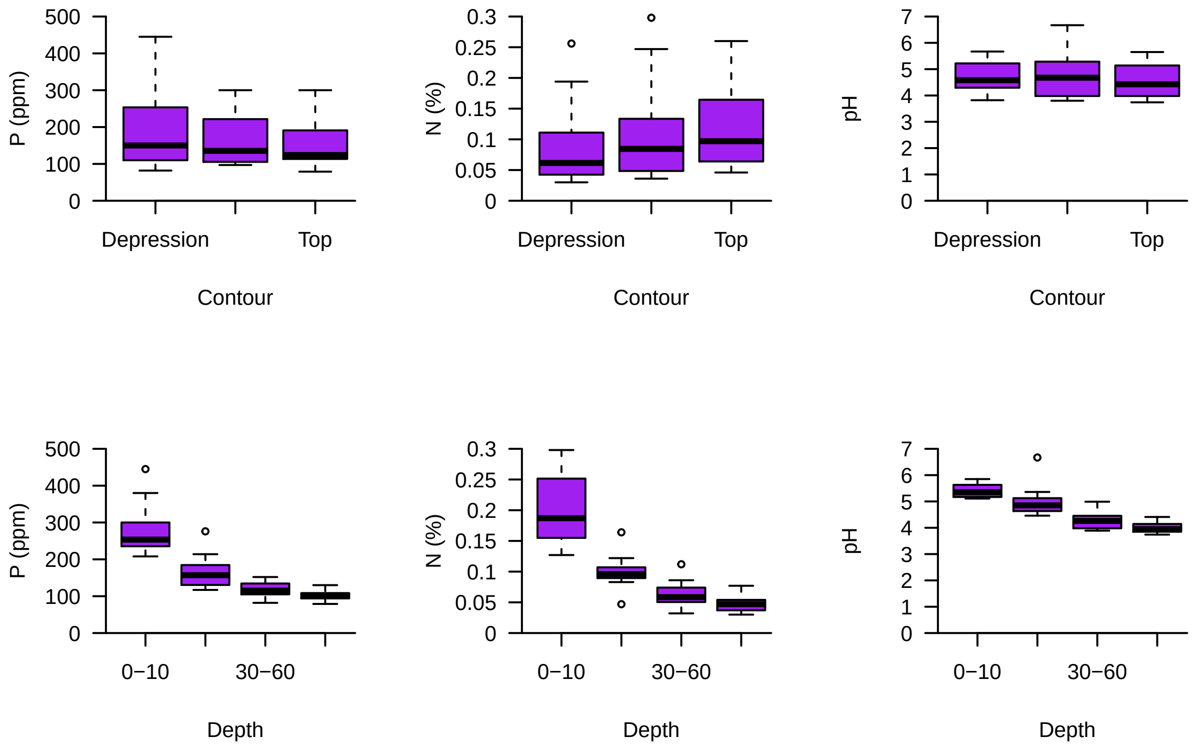
2. Specify the position of plots in a matrix:

```r
#Take a look at the matrix:
matrix(1:6, byrow=TRUE, ncol=3) #use ncol to specify number of columns
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
```

```r
#First plot will be upper left, second upper middle, third, upper left, etc.

nf<-layout(matrix(1:6, byrow=TRUE, ncol=3)) #use a matrix to specify layout

#call the plots again
#boxplot of P by Contour:
b1<-boxplot(P~Contour, data=Soils, col="purple", frame=F, ylim=c(0,500), axes=F, ylab="P (ppm)", xlab="
axis(1, labels=levels(Soils$Contour), at=1:3, pos=0)
axis(2, labels=seq(0,500,100), at=seq(0,500,100), las=2)
segments(0,0,3.5,0)

#boxplot of N by Contour:
b2<-boxplot(N~Contour, data=Soils, col="purple", frame=F, ylim=c(0,0.3), axes=F,ylab="N (%)", xlab="Con
axis(1, labels=levels(Soils$Contour), at=1:3, pos=0)
axis(2, labels=seq(0,0.3,0.05), at=seq(0,0.3,0.05),las=2)
segments(0,0,3.5,0)
```

```
#boxplot of pH by Contour:
b3<-boxplot(pH~Contour, data=Soils, col="purple", frame=F, ylim=c(0,7), axes=F, ylab="pH", xlab="Contou
axis(1, labels=levels(Soils$Contour), at=1:3, pos=0)
axis(2, labels=seq(0,7,1), at=seq(0,7,1), las=2)
segments(0,0,3.5,0)

#boxplot of P by Depth:
b4<-boxplot(P~Depth, data=Soils, col="purple", frame=F, ylim=c(0,500), axes=F, ylab="P (ppm)", xlab="De
axis(1, labels=levels(Soils$Depth), at=1:4, pos=0)
axis(2, labels=seq(0,500,100), at=seq(0,500,100), las=2)
segments(0,0,4.5,0)

#boxplot of N by Depth:
b5<-boxplot(N~Depth, data=Soils, col="purple", frame=F, ylim=c(0,0.3), axes=F,ylab="N (%)", xlab="Depth
axis(1, labels=levels(Soils$Depth), at=1:4, pos=0)
axis(2, labels=seq(0,0.3,0.05), at=seq(0,0.3,0.05),las=2)
segments(0,0,4.5,0)

#boxplot of pH by Contour:
b6<-boxplot(pH~Depth, data=Soils, col="purple", frame=F, ylim=c(0,7), axes=F, ylab="pH", xlab="Depth")
axis(1, labels=levels(Soils$Depth), at=1:4, pos=0)
axis(2, labels=seq(0,7,1), at=seq(0,7,1), las=2)
segments(0,0,4.5,0)
```



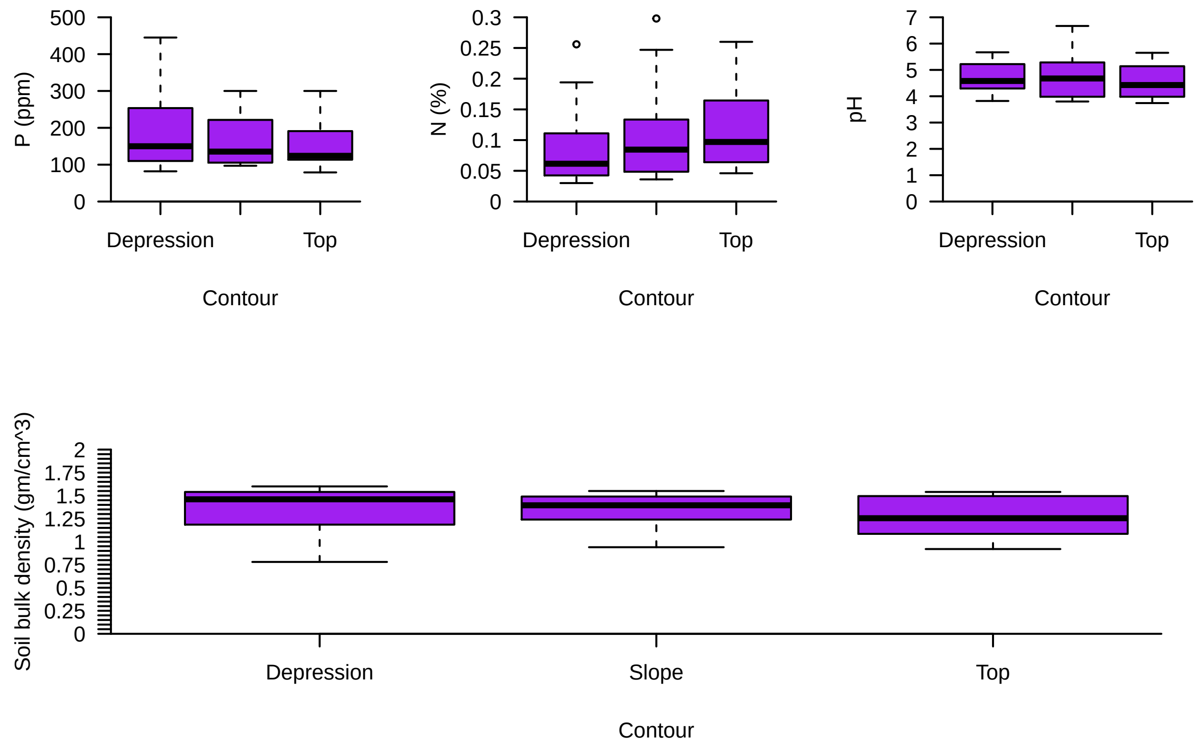
You can also use layout() and matrix() to change the relative size of the plots by having one plot take up more positions in the matrix.

Example: I want the P,N, and pH plots by contour on top, but add a fourth of density by contour which spans the bottom of the multi-panel plot.

```
#Take a look at the matrix:
matrix(c(1:3,4,4,4), byrow=TRUE, ncol=3) #use ncol to specify number of columns. I have three 4's becau
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    4    4
```

```
nf<-layout(matrix(c(1:3,4,4,4), byrow=TRUE, ncol=3)) #use a matrix to specify layout


#boxplot of P by Contour:
boxplot(P~Contour, data=Soils, col="purple", frame=F, ylim=c(0,500), axes=F, ylab="P (ppm)", xlab="Conto
axis(1, labels=levels(Soils$Contour), at=1:3, pos=0)
axis(2, labels=seq(0,500,100), at=seq(0,500,100), las=2)
segments(0,0,3.5,0)

#boxplot of N by Contour:
boxplot(N~Contour, data=Soils, col="purple", frame=F, ylim=c(0,0.3), axes=F,ylab="N (%)", xlab="Contour
axis(1, labels=levels(Soils$Contour), at=1:3, pos=0)
axis(2, labels=seq(0,0.3,0.05), at=seq(0,0.3,0.05),las=2)
segments(0,0,3.5,0)

#boxplot of pH by Contour:
boxplot(pH~Contour, data=Soils, col="purple", frame=F, ylim=c(0,7), axes=F, ylab="pH", xlab="Contour")
axis(1, labels=levels(Soils$Contour), at=1:3, pos=0)
axis(2, labels=seq(0,7,1), at=seq(0,7,1), las=2)
segments(0,0,3.5,0)

#boxplot of Soil density by Contour:
boxplot(Dens~Contour, data=Soils, col="purple", frame=F, ylim=c(0,2), axes=F, ylab="Soil bulk density (
axis(1, labels=levels(Soils$Contour), at=1:3, pos=0)
axis(2, labels=seq(0,2,0.05), at=seq(0,2,0.05), las=2)
segments(0,0,3.5,0)
```

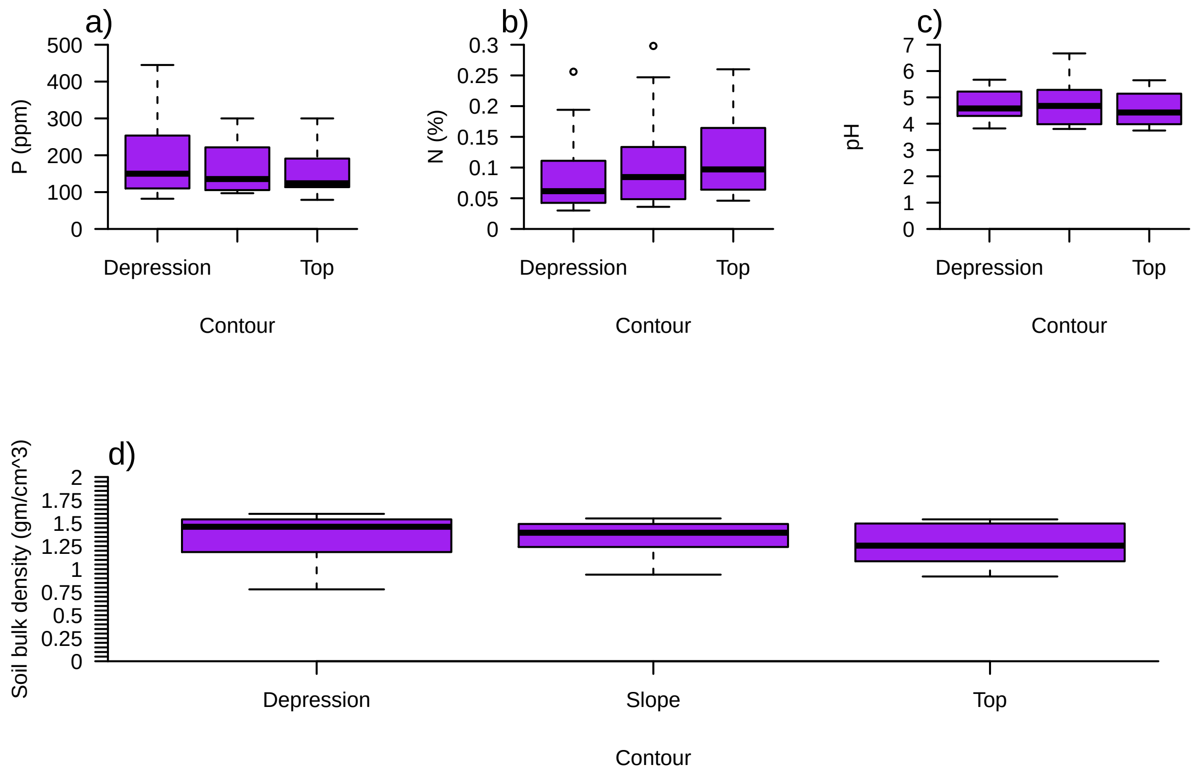


Add a), b), c) labels to multi-panel plots

Use mtext to label multi-panel plots

```
nf<-layout(matrix(c(1:3,4,4,4), byrow=TRUE, ncol=3)) #use a matrix to specify layout

#boxplot of P by Contour:
boxplot(P~Contour, data=Soils, col="purple", frame=F, ylim=c(0,500), axes=F, ylab="P (ppm)", xlab="Conte
axis(1, labels=levels(Soils$Contour), at=1:3, pos=0)
axis(2, labels=seq(0,500,100), at=seq(0,500,100), las=2)
segments(0,0,3.5,0)
mtext("a)", side=3, adj=-0.1, line=0)

#boxplot of N by Contour:
boxplot(N~Contour, data=Soils, col="purple", frame=F, ylim=c(0,0.3), axes=F,ylab="N (%)", xlab="Contour
axis(1, labels=levels(Soils$Contour), at=1:3, pos=0)
axis(2, labels=seq(0,0.3,0.05), at=seq(0,0.3,0.05),las=2)
segments(0,0,3.5,0)
mtext("b)", side=3, adj=-0.1, line=0)

#boxplot of pH by Contour:
boxplot(pH~Contour, data=Soils, col="purple", frame=F, ylim=c(0,7), axes=F, ylab="pH", xlab="Contour")
axis(1, labels=levels(Soils$Contour), at=1:3, pos=0)
axis(2, labels=seq(0,7,1), at=seq(0,7,1), las=2)
segments(0,0,3.5,0)
```

```
mtext("c)", side=3, adj=-0.1, line=0)

#boxplot of Soil density by Contour:
boxplot(Dens~Contour, data=Soils, col="purple", frame=F, ylim=c(0,2), axes=F, ylab="Soil bulk density (g
axis(1, labels=levels(Soils$Contour), at=1:3, pos=0)
axis(2, labels=seq(0,2,0.05), at=seq(0,2,0.05), las=2)
segments(0,0,3.5,0)
mtext("d)", side=3, adj=0, line=0)
```



Change margins of a plot

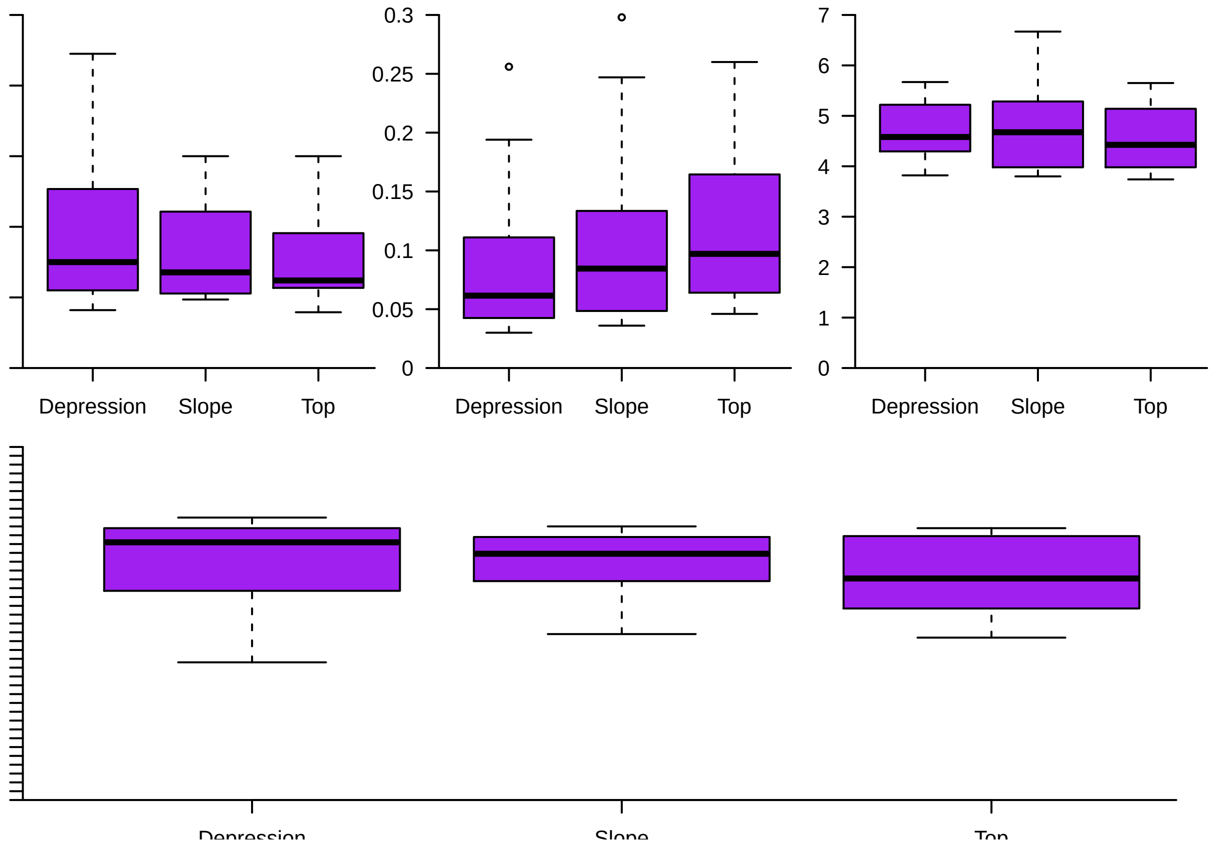
Use par(mar=c(#,#,#,#)) to change the inner margins and par(oma=c(#,#,#,#)) to change outer margins. First number is bottom, second is left, third is top, fourth is right.

Use par(mar=c(), oma=c()) to change both together.

```
#Take a look at the matrix:
matrix(c(1:3,4,4,4), byrow=TRUE, ncol=3) #use ncol to specify number of columns. I have three 4's becau
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    4    4
```

```
nf<-layout(matrix(c(1:3,4,4,4), byrow=TRUE, ncol=3)) #use a matrix to specify layout

par(mar=c(1,1,1,1)) #change the margins

#boxplot of P by Contour:
boxplot(P~Contour, data=Soils, col="purple", frame=F, ylim=c(0,500), axes=F, ylab="P (ppm)", xlab="Conto
axis(1, labels=levels(Soils$Contour), at=1:3, pos=0)
axis(2, labels=seq(0,500,100), at=seq(0,500,100), las=2)
segments(0,0,3.5,0)

#boxplot of N by Contour:
boxplot(N~Contour, data=Soils, col="purple", frame=F, ylim=c(0,0.3), axes=F,ylab="N (%)", xlab="Contour
axis(1, labels=levels(Soils$Contour), at=1:3, pos=0)
axis(2, labels=seq(0,0.3,0.05), at=seq(0,0.3,0.05),las=2)
segments(0,0,3.5,0)

#boxplot of pH by Contour:
boxplot(pH~Contour, data=Soils, col="purple", frame=F, ylim=c(0,7), axes=F, ylab="pH", xlab="Contour")
axis(1, labels=levels(Soils$Contour), at=1:3, pos=0)
axis(2, labels=seq(0,7,1), at=seq(0,7,1), las=2)
segments(0,0,3.5,0)

#boxplot of Soil density by Contour:
boxplot(Dens~Contour, data=Soils, col="purple", frame=F, ylim=c(0,2), axes=F, ylab="Soil bulk density (
axis(1, labels=levels(Soils$Contour), at=1:3, pos=0)
axis(2, labels=seq(0,2,0.05), at=seq(0,2,0.05), las=2)
segments(0,0,3.5,0)
```



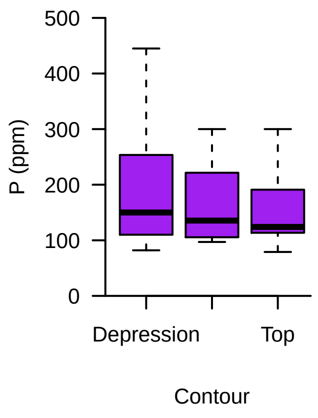
The inner marings are smaller so the plots are closer together, but all the labels are cut off. Trying making the marings bigger, especially those with labels.

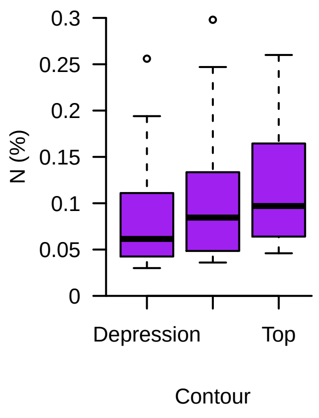
```r
#Take a look at the matrix:
matrix(c(1:3,4,4,4), byrow=TRUE, ncol=3) #use ncol to specify number of columns. I have three 4's becau
```

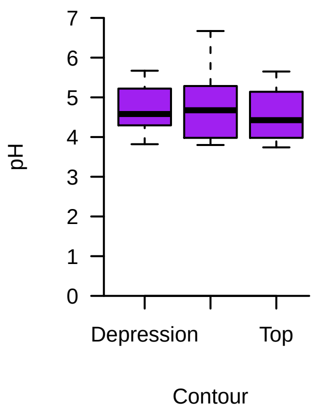
```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    4    4
```

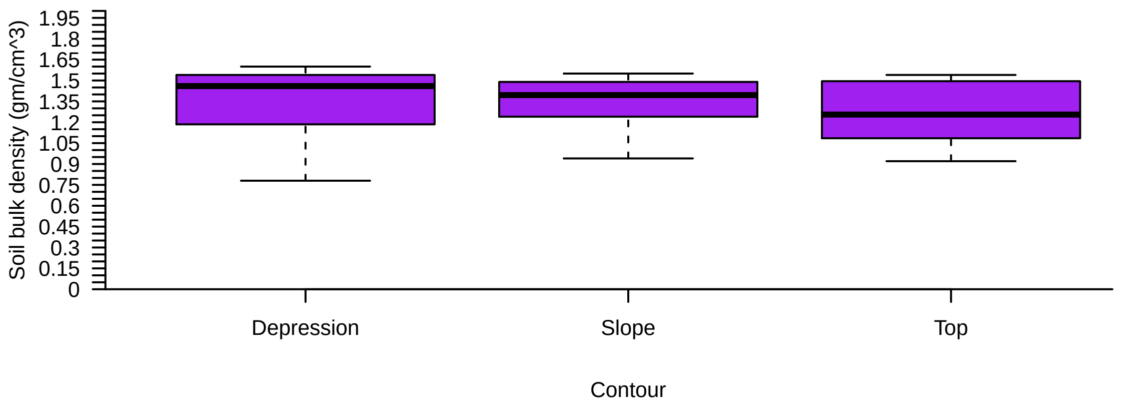
```r
nf<-layout(matrix(c(1:3,4,4,4), byrow=TRUE, ncol=3)) #use a matrix to specify layout

par(mar=c(4,4,1.2,4)) #change the margins

#boxplot of P by Contour:
boxplot(P~Contour, data=Soils, col="purple", frame=F, ylim=c(0,500), axes=F, ylab="P (ppm)", xlab="Conto
axis(1, labels=levels(Soils$Contour), at=1:3, pos=0)
axis(2, labels=seq(0,500,100), at=seq(0,500,100), las=2)
segments(0,0,3.5,0)

#boxplot of N by Contour:
boxplot(N~Contour, data=Soils, col="purple", frame=F, ylim=c(0,0.3), axes=F,ylab="N (%)", xlab="Contour
axis(1, labels=levels(Soils$Contour), at=1:3, pos=0)
axis(2, labels=seq(0,0.3,0.05), at=seq(0,0.3,0.05),las=2)
segments(0,0,3.5,0)
```

```r
#boxplot of pH by Contour:
boxplot(pH~Contour, data=Soils, col="purple", frame=F, ylim=c(0,7), axes=F, ylab="pH", xlab="Contour")
axis(1, labels=levels(Soils$Contour), at=1:3, pos=0)
axis(2, labels=seq(0,7,1), at=seq(0,7,1), las=2)
segments(0,0,3.5,0)

#boxplot of Soil density by Contour:
boxplot(Dens~Contour, data=Soils, col="purple", frame=F, ylim=c(0,2), axes=F, ylab="Soil bulk density (
axis(1, labels=levels(Soils$Contour), at=1:3, pos=0)
axis(2, labels=seq(0,2,0.05), at=seq(0,2,0.05), las=2)
segments(0,0,3.5,0)
```









Basic statistical models and tests

Linear regression, ANOVAs, and t-tests are all linear models and can be done with the function lm (ie linear model). However, ANOVAs and t-tests also have their own functions.

Generalized linear models are extensions of the linear model for non-normal distributions.

Linear or generalized mixed models incorporate random effects. Generalized additive models and polynomial regression account for non-linearity in the response.

Non-linear models incorporate non-linearity in the parameters.

T-tests

One sample-test

```
#This is a subset of data of stable isotopes for nitrogen taken from 10 sixgill shark embryo livers and

pup_liver_isotopes<-c(12.78,12.52,12.69,12.43,12.23,12.33, 12.29,12.21,12.40,12.55) # c() is concatenat
mom_liver_value<-13.05

t.test(pup_liver_isotopes, mu=mom_liver_value, alternative = "two.sided") #two-tailed one sample t-test
```

```
##
##  One Sample t-test
##
## data:  pup_liver_isotopes
## t = -10.027, df = 9, p-value = 3.498e-06
## alternative hypothesis: true mean is not equal to 13.05
## 95 percent confidence interval:
##  12.30606 12.57994
## sample estimates:
## mean of x
##    12.443
```

```
#Test assumptions:
#Normality:
shapiro.test(pup_liver_isotopes) # p-value > 0.05 so data are normally distributed
```

```
##
##  Shapiro-Wilk normality test
##
## data:  pup_liver_isotopes
## W = 0.94613, p-value = 0.623
```

Paired t-test

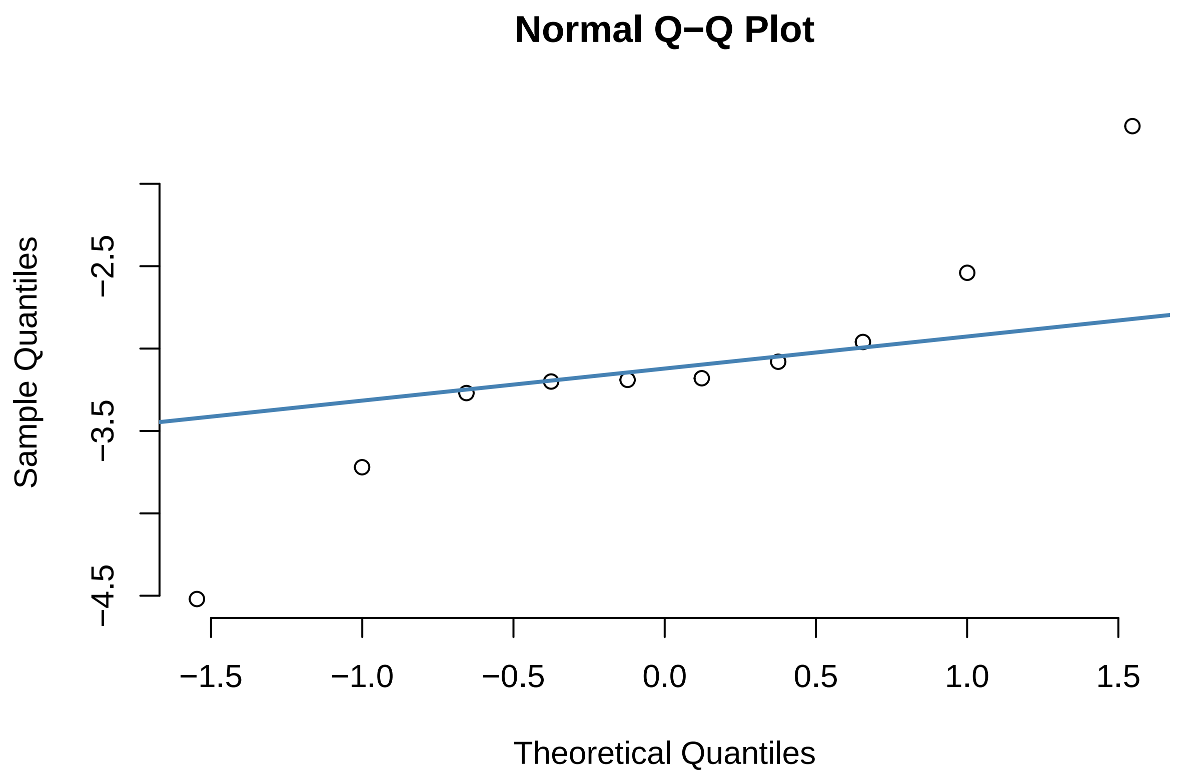
```
#What about comparing nitrogen isotope values between liver and muscle, a sample of each taken from eac

pup_liver_isotopes<-c(12.78,12.52,12.69,12.43,12.23,12.33, 12.29,12.21,12.40,12.55) # same as above

pup_muscle_isotopes<-c(14.43,15.72,15.77,15.39,15.42,16.85,15.56,15.93,15.58,15.09) # order is the same

t.test(pup_liver_isotopes, pup_muscle_isotopes, paired=TRUE) #each vector must be numeric with the orde
```

```
##
##  Paired t-test
##
## data:  pup_liver_isotopes and pup_muscle_isotopes
## t = -13.445, df = 9, p-value = 2.907e-07
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -3.657796 -2.604204
## sample estimates:
## mean of the differences
##                  -3.131
```

```
#Test assumptions:
#Normality of differences:
pup_differences<-(pup_liver_isotopes-pup_muscle_isotopes) #set up a numeric vector with the differences
shapiro.test(pup_differences) # p-value > 0.05 so the differences are normally distributed
```

```
##
##  Shapiro-Wilk normality test
##
## data:  pup_differences
## W = 0.91767, p-value = 0.3379
```

```
#QQ plot on the differences
qqnorm(pup_differences, pch = 1, frame = FALSE)
qqline(pup_differences, col = "steelblue", lwd = 2)
```



Two-sample t-test

```r
#Two marine protected areas in South Africa, one designed for fish with lots of kelp and reef habitat, 

#Here are two ways to have your data stored in R:
#You can have them each as vectors:
WhaleSanctuary=c(1.8816523, 1.8389828, 1.3239195, 0.6931472,  0.6615632,1.2299186, 1.9963156, 1.0045784

KelpMPA=c(1.3835889, 1.7118451, 1.0567522, 0.9973048, 1.9783942, 1.1156818, 1.3214234, 1.9923252, 1.298

#You can also store them together in a data frame (see Data frame section):
MPA_Data<-data.frame(MPA=c(rep("WhaleSanctuary", length(WhaleSanctuary)), rep("KelpMPA", length(KelpMPA

#With group1, group2 as seperate vectors:
t.test(WhaleSanctuary, KelpMPA, var.equal = TRUE)
```

```
##
##  Two Sample t-test
##
## data:  WhaleSanctuary and KelpMPA
## t = -0.4908, df = 18, p-value = 0.6295
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.5833327  0.3623993
## sample estimates:
## mean of x mean of y
##  1.305367  1.415833
```

```r
#or with y~x and x is categorical with only two groups (binary):
t.test(ShannonDiversity~MPA, data= MPA_Data, var.equal=TRUE)# as a data frame
```

```
##
##  Two Sample t-test
##
## data:  ShannonDiversity by MPA
## t = 0.4908, df = 18, p-value = 0.6295
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.3623993  0.5833327
## sample estimates:
##        mean in group KelpMPA mean in group WhaleSanctuary
##                      1.415833                     1.305367
```

```r
# ~ symbol means "as a funciton of" - used to code many statistical tests in R

#Output shows mean Shannon diversity of KelpMPA is greater but NOT significantly based on alpha=0.05.

#Test assumptions:
#Normality of each broup:
shapiro.test(WhaleSanctuary)
```

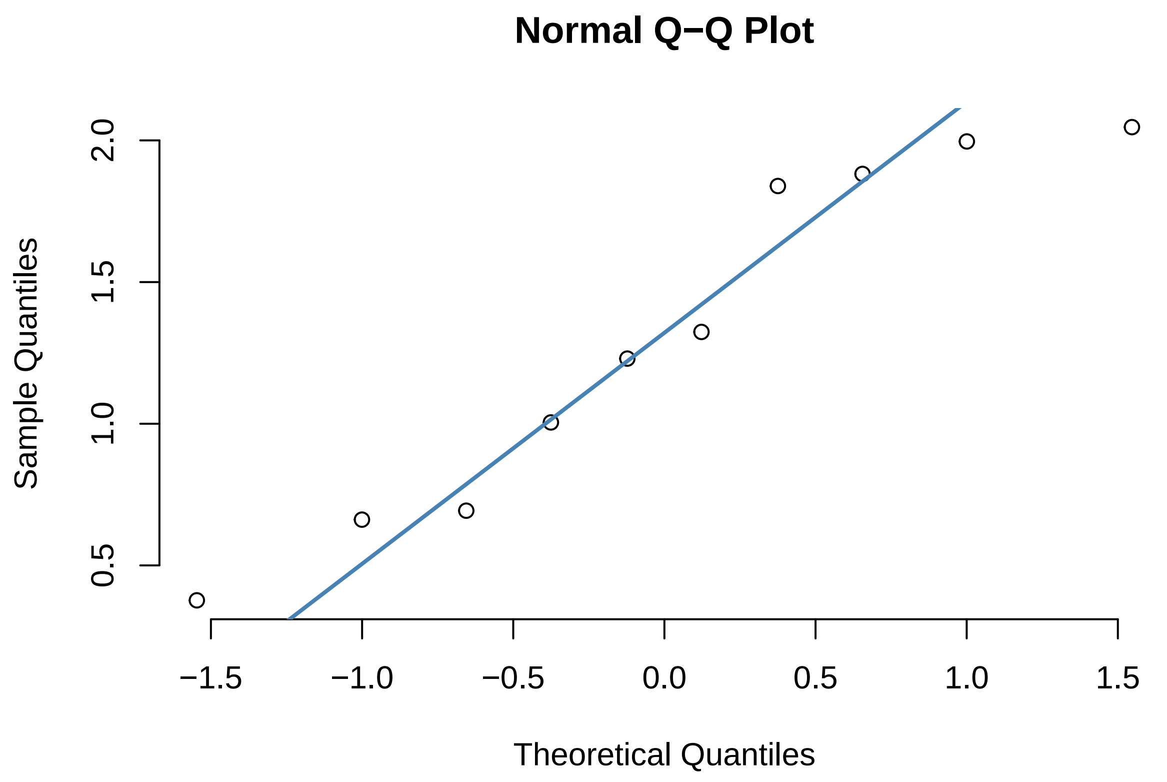
```
##
##  Shapiro-Wilk normality test
```

```
##
## data:  WhaleSanctuary
## W = 0.91132, p-value = 0.2902
```

```
shapiro.test(KelpMPA)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  KelpMPA
## W = 0.87925, p-value = 0.1279
```

```
#p-value > 0.05 in both so they are normally distributed

#QQ plot for checking normality (example is just for Whale Sanctuary)
qqnorm(WhaleSanctuary, pch = 1, frame = FALSE)
qqline(WhaleSanctuary, col = "steelblue", lwd = 2)
```



```
#Equal variances
#With group1, group2 as separate objects:
var.test(WhaleSanctuary, KelpMPA) #performs F-test
```

```
##
##  F test to compare two variances
```

```
##
## data:  WhaleSanctuary and KelpMPA
## F = 2.9062, num df = 9, denom df = 9, p-value = 0.1278
## alternative hypothesis: true ratio of variances is not equal to 1
## 95 percent confidence interval:
##   0.7218591 11.7003468
## sample estimates:
## ratio of variances
##           2.906201
```

```
#or with y~x and x is categorical with only two groups (binary):
var.test(ShannonDiversity~MPA, data=MPA_Data)
```

```
##
##  F test to compare two variances
##
## data:  ShannonDiversity by MPA
## F = 0.34409, num df = 9, denom df = 9, p-value = 0.1278
## alternative hypothesis: true ratio of variances is not equal to 1
## 95 percent confidence interval:
##  0.08546755 1.38531184
## sample estimates:
## ratio of variances
##          0.3440919
```

```
bartlett.test(ShannonDiversity~MPA, data=MPA_Data) #Perform Bartlett's test
```

```
##
##  Bartlett test of homogeneity of variances
##
## data:  ShannonDiversity by MPA
## Bartlett's K-squared = 2.3191, df = 1, p-value = 0.1278
```

```
library(car) #for Levene's test - remember to run install.packages("car") once to install the package o
leveneTest(ShannonDiversity~MPA, data=MPA_Data) #Levene's test
```

```
## Warning in leveneTest.default(y = y, group = group, ...): group coerced to
## factor.
```

```
## Levene's Test for Homogeneity of Variance (center = median)
##       Df F value  Pr(>F)
## group  1  4.1301 0.05715 .
##       18
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
#If assumption of equality of variance is NOT meet, perform Welch's two-sample t-test by changing to va
t.test(ShannonDiversity~MPA, data= MPA_Data, var.equal=FALSE)
```

```
##
##  Welch Two Sample t-test
```

```
##
## data:  ShannonDiversity by MPA
## t = 0.4908, df = 14.538, p-value = 0.6309
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.3706017  0.5915351
## sample estimates:
##        mean in group KelpMPA mean in group WhaleSanctuary
##                     1.415833                     1.305367
```

Non-parametric versions of the t-test

NOTE: Check out Generalized Linear Models (GLMs) below that do not assume normality but are still parametric - especially if you have count, proportion, or presence/absence or yes/no data. These are preferable to non-parametric tests.

```
# Mann-Whitney U Test/Wilcoxon rank sum test for two-sample t-test
#With y~x and x is binary (categorical with two groups) as the argument:
wilcox.test(ShannonDiversity~MPA, data=MPA_Data) #see two-sample t-test section for the data
```

```
##
##  Wilcoxon rank sum exact test
##
## data:  ShannonDiversity by MPA
## W = 54, p-value = 0.7959
## alternative hypothesis: true location shift is not equal to 0
```

```
#or with group1, group2 as seperate vectors as arguments:
wilcox.test(KelpMPA, WhaleSanctuary)
```

```
##
##  Wilcoxon rank sum exact test
##
## data:  KelpMPA and WhaleSanctuary
## W = 54, p-value = 0.7959
## alternative hypothesis: true location shift is not equal to 0
```

```
# Wilcoxon signed rank test for paired samples:
wilcox.test(pup_liver_isotopes, pup_muscle_isotopes,paired=TRUE) #see paired sample t-test section for
```

```
##
##  Wilcoxon signed rank exact test
##
## data:  pup_liver_isotopes and pup_muscle_isotopes
## V = 0, p-value = 0.001953
## alternative hypothesis: true location shift is not equal to 0
```

ANOVA

```r
#Read in data on soils
library(carData) #remember to install.packages(carData) before first use
data("Soils") #some packages have data sets saved in them. The data() function retrieves them - in this
head(Soils)
```

```
##   Group Contour Depth Gp Block   pH     N Dens   P    Ca   Mg    K   Na Conduc
## 1     1     Top  0-10 T0     1 5.40 0.188 0.92 215 16.35 7.65 0.72 1.14   1.09
## 2     1     Top  0-10 T0     2 5.65 0.165 1.04 208 12.25 5.15 0.71 0.94   1.35
## 3     1     Top  0-10 T0     3 5.14 0.260 0.95 300 13.02 5.68 0.68 0.60   1.41
## 4     1     Top  0-10 T0     4 5.14 0.169 1.10 248 11.92 7.88 1.09 1.01   1.64
## 5     2     Top 10-30 T1     1 5.14 0.164 1.12 174 14.17 8.12 0.70 2.17   1.85
## 6     2     Top 10-30 T1     2 5.10 0.094 1.22 129  8.55 6.92 0.81 2.67   3.18
```

```r
#Focusing on pH - measurements at different contours of topography and soil depths at four different ar

#Run an ANOVA, saving the output to an object I made up called a.out
#works as y ~ x1 + x2 or y ~ x1*x2 for interations
#Remember ~ means "is a function of"
a.out<-aov(pH~Contour*Depth+Block, data=Soils) # the * specifies an interaction between Depth and Conto
#could also write as: aov(pH~Contour+Depth+Block+Contour:Depth, data=Soils). The : symbol is for specif
#Both A:B and A*B would work for interaction
#Three-way interation would be A:B:C or A*B*C

summary(a.out) #get the ANOVA table with p-values
```

```
##               Df Sum Sq Mean Sq F value   Pr(>F)    
## Contour        2  0.261   0.130   1.013   0.3743    
## Depth          3 14.959   4.986  38.742 6.41e-11 ***
## Block          3  1.232   0.411   3.192   0.0362 *  
## Contour:Depth  6  0.516   0.086   0.668   0.6759    
## Residuals     33  4.247   0.129                     
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
#If you want the coefficients table (ie ANOVA expressed as a linear model)
a.model.out<-lm(pH~Contour*Depth+Block, data=Soils)
summary(a.model.out) # gives you the effect sizes, the differences between each group and the strength
```

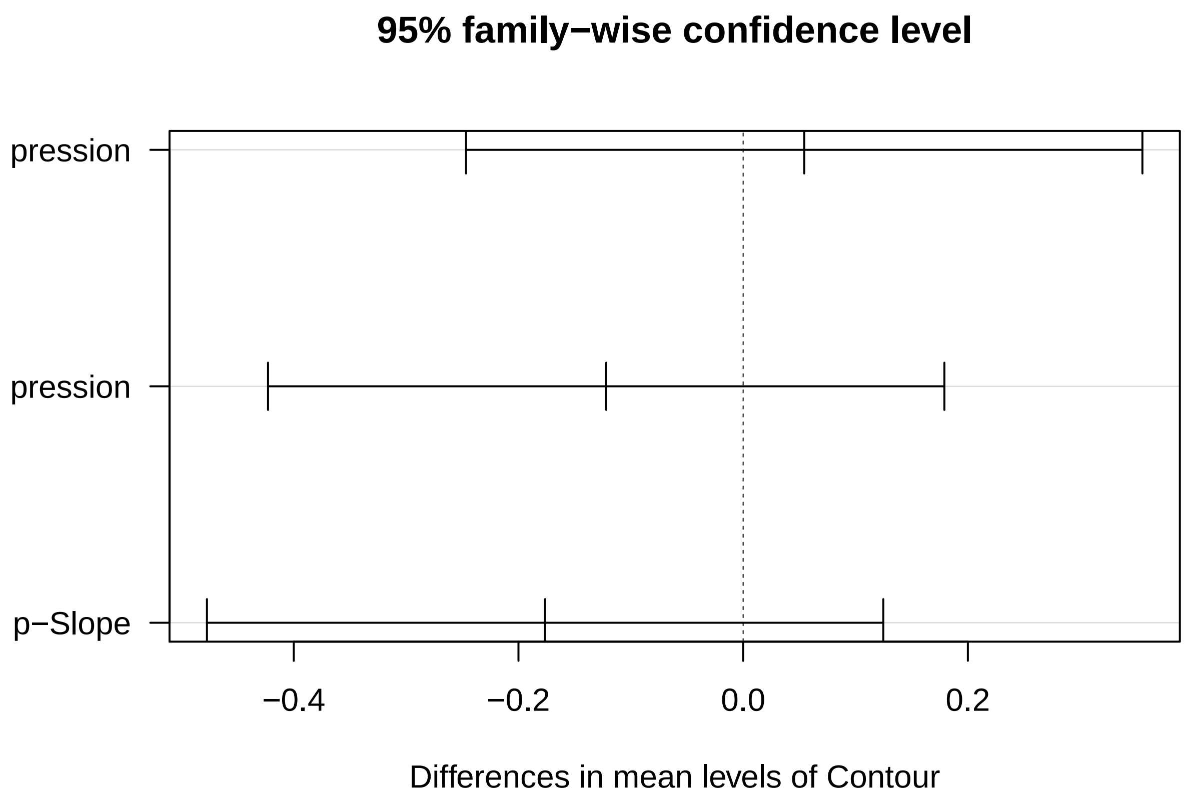
```
## 
## Call:
## lm(formula = pH ~ Contour * Depth + Block, data = Soils)
## 
## Residuals:
##      Min       1Q   Median       3Q      Max 
## -0.62396 -0.16813  0.02688  0.13063  1.15354 
## 
## Coefficients:
##                         Estimate Std. Error t value Pr(>|t|)    
## (Intercept)               5.1573     0.2006  25.716  < 2e-16 ***
## ContourSlope              0.1550     0.2537   0.611 0.545380    
## ContourTop               -0.0200     0.2537  -0.079 0.937636    
## Depth10-30                -0.4725     0.2537  -1.863 0.071443 .
## Depth30-60                -0.9900     0.2537  -3.903 0.000443 ***
## Depth60-90                -1.1800     0.2537  -4.652 5.13e-05 ***
## Block2                     0.2433     0.1465   1.661 0.106103
## Block3                     0.1083     0.1465   0.740 0.464730
## Block4                     0.4292     0.1465   2.930 0.006105 **
## ContourSlope:Depth10-30    0.2475     0.3588   0.690 0.495091
## ContourTop:Depth10-30     -0.0100     0.3588  -0.028 0.977930
## ContourSlope:Depth30-60   -0.2500     0.3588  -0.697 0.490776
## ContourTop:Depth30-60     -0.1375     0.3588  -0.383 0.703979
## ContourSlope:Depth60-90   -0.4000     0.3588  -1.115 0.272921
## ContourTop:Depth60-90     -0.2600     0.3588  -0.725 0.473728
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3588 on 33 degrees of freedom
## Multiple R-squared:  0.7998, Adjusted R-squared:  0.7149
## F-statistic: 9.417 on 14 and 33 DF,  p-value: 7.405e-08
```

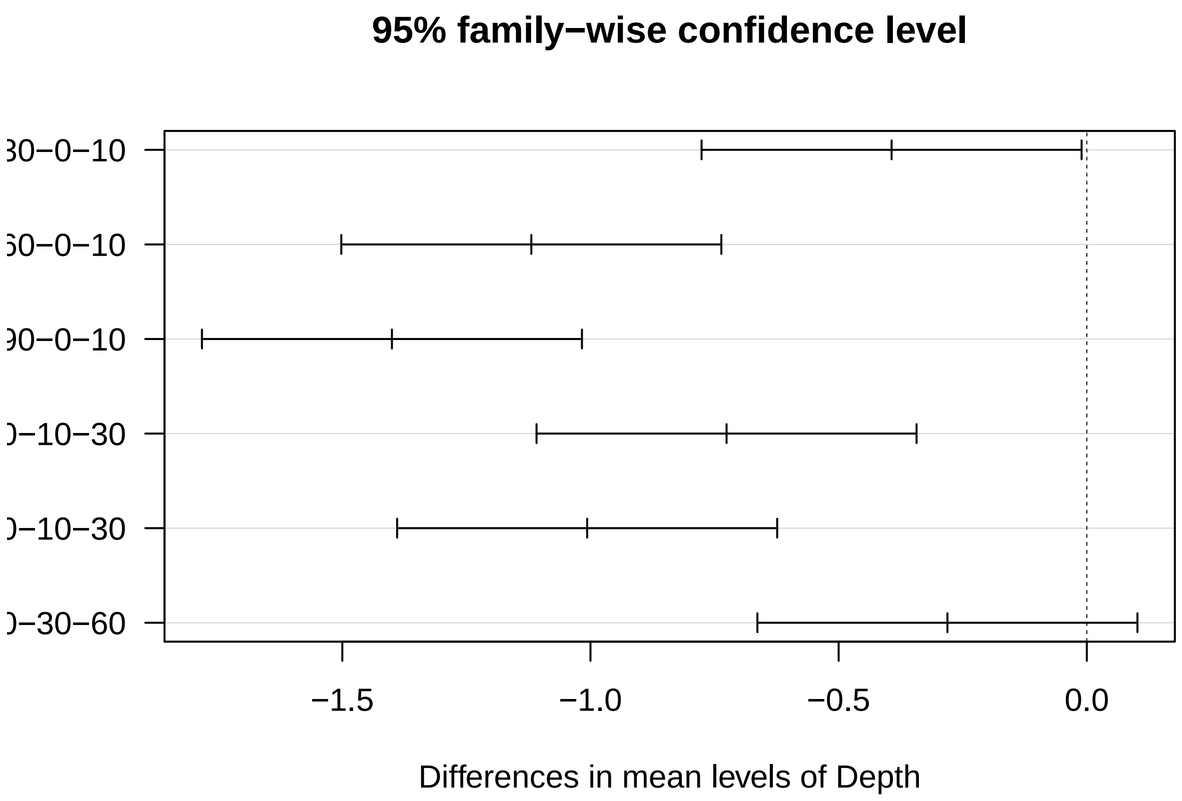
```
#The interaction was not significant, so what if I want to run without it?
a.out2<-update(a.out, ~.-Contour:Depth) # the period (.) means everything. This says update a.out, but
summary(a.out2)
```

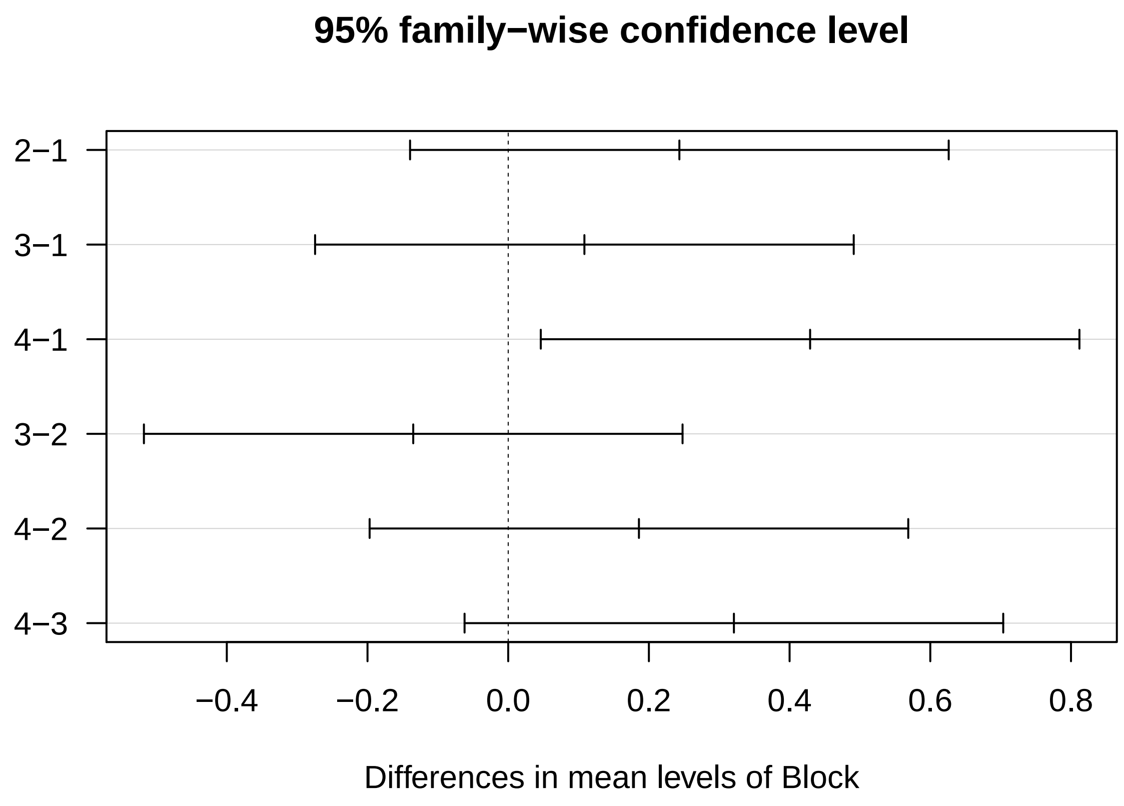
```
##             Df Sum Sq Mean Sq F value   Pr(>F)
## Contour      2  0.261   0.130   1.067   0.3538
## Depth        3 14.959   4.986  40.827 4.13e-12 ***
## Block        3  1.232   0.411   3.364   0.0282 *
## Residuals   39  4.763   0.122
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Tukey post-hoc test

```
tukey.plot.test<-TukeyHSD(a.out2) #run a Tukey post-hoc test for the above ANOVA
plot(tukey.plot.test, las = 1) # the significant comparisons have error bars NOT overlapping with zero.
```

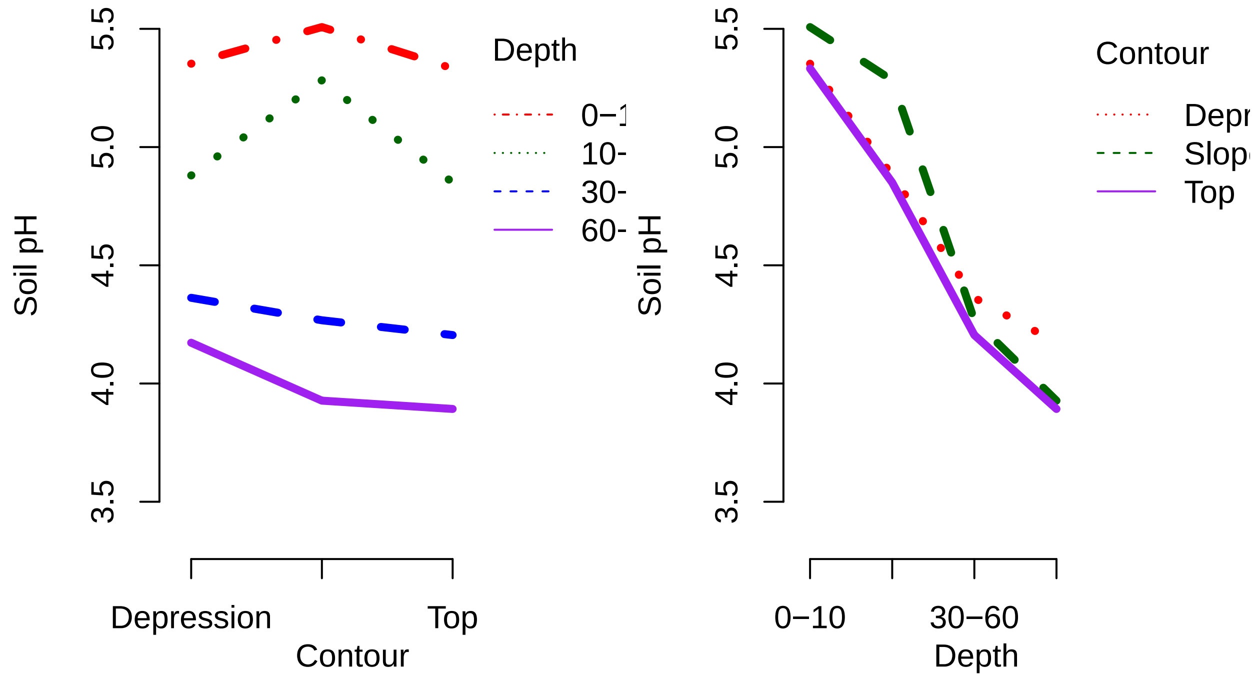

95% family−wise confidence level
pression
pression
p−Slope
−0.4
−0.2
0.0
0.2
Differences in mean levels of Contour


95% family−wise confidence level
30−0−10
60−0−10
90−0−10
0−10−30
0−10−30
0−30−60
−1.5
−1.0
−0.5
0.0
Differences in mean levels of Depth



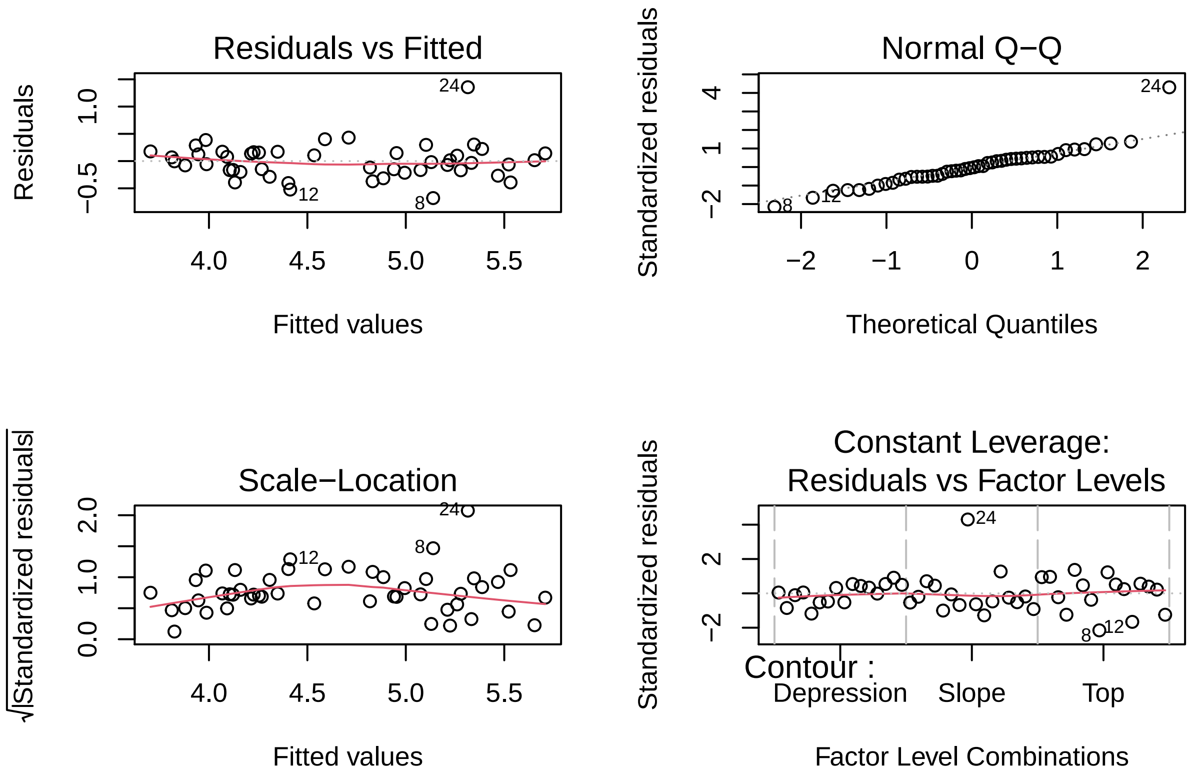
Interaction plots

```r
par(mfrow=c(1,2)) #set plotting window to be 1 row and 2 columns
#Show how the effect of contour on soil pH changes at different depths
interaction.plot(x.factor=Soils$Contour,
                 trace.factor=Soils$Depth,
                 response=Soils$pH,
                 trace.label="Depth", xlab="Contour",
                 ylab="Soil pH",
                 col=c("red", "darkgreen", "blue", "purple"),
                 lwd=4, bty="n", ylim=c(3.5,5.5), xtick=TRUE)

#Show how the effect of depth on soil pH changes at different contours
interaction.plot(x.factor=Soils$Depth,
                 trace.factor=Soils$Contour,
                 response=Soils$pH,
                 trace.label="Contour", xlab="Depth",
                 ylab="Soil pH",
                 col=c("red", "darkgreen", "purple"),
                 lwd=4, bty="n", ylim=c(3.5,5.5), xtick=TRUE)
```



Model diagnostics (are assumptions met?)

```
#Examine diagnostic plots (ie Q-Q plot and residual vs fitted) for the ANOVA:
par(mfrow=c(2,2))
plot(a.out2) # Examine residual vs fitted for issues with equal variance
```



```
#Testing normality of residuals (although can use Q-Q plot above)
shapiro.test(a.out2$residuals)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  a.out2$residuals
## W = 0.90079, p-value = 0.0006667
```

Non-parametric Kruskal-Wallis Test when normality not met

NOTE: Check out Generalized Linear Models (GLMs) below that do not assume normality but are still parametric - especially if you have count, proportion, or presence/absence or yes/no data. These are preferable to non-parametric tests.

```
kruskal.test(pH~Depth, data = Soils) #Remember: assumption of equality of variance still required
```

```
##
##  Kruskal-Wallis rank sum test
##
## data:  pH by Depth
## Kruskal-Wallis chi-squared = 36.976, df = 3, p-value = 4.656e-08
```

```
#pairwise test with Bonferroni correction for multiple testing:
#the function requires the list of y values to be seperated from the categories of the factor (Contour)
pairwise.wilcox.test(Soils$pH, Soils$Depth,
                     p.adjust.method = "bonferroni") # get p-values of each comparison
```

```
## Warning in wilcox.test.default(xi, xj, paired = paired, ...): cannot compute
## exact p-value with ties

## Warning in wilcox.test.default(xi, xj, paired = paired, ...): cannot compute
## exact p-value with ties

## Warning in wilcox.test.default(xi, xj, paired = paired, ...): cannot compute
## exact p-value with ties

## Warning in wilcox.test.default(xi, xj, paired = paired, ...): cannot compute
## exact p-value with ties

## Warning in wilcox.test.default(xi, xj, paired = paired, ...): cannot compute
## exact p-value with ties

## Warning in wilcox.test.default(xi, xj, paired = paired, ...): cannot compute
## exact p-value with ties

##
##  Pairwise comparisons using Wilcoxon rank sum test with continuity correction
##
## data:  Soils$pH and Soils$Depth
##
##       0-10    10-30   30-60
## 10-30 0.01917 -       -
## 30-60 0.00022 0.00354 -
## 60-90 0.00022 0.00022 0.14569
##
## P value adjustment method: bonferroni
```

Simple linear regression

Linear regression builds on ANOVAs, letting explanatory variables be continuous numbers. In fact, an ANOVA is just a linear regression for categorical variables.

Linear regression estimates the linear relationship (ie intercept and slope) between a continuous response variable and an explanatory variable. The slope represents your effect size (strength of th relationship).

You can get significance of the relationship (ie p-values) two ways: 1. ANOVA using the anova() function. 2. One-sample t-test asking if the slope is significantly different from zero using the summary() function.

For linear regression, these two methods are mathematically the same.

Linear regression assumptions: 1. Independence of observations 2. Variance in the response is the same for all values of the explanatory variable (homoscedasticity/equality of variance). 3. Residuals are normally distributed.

The basic function in R for linear regression is lm() - linear model. Use: lm(response~explanatory, data=name of data frame)

Example: can you predict brain size from body size in mammals?

```r
#install.packages("MASS")
library(MASS) #example mammal data is in this package
data(mammals) #load the data from the MASS package
head(mammals) #see first six rows
```

```
##                    body brain
## Arctic fox        3.385  44.5
## Owl monkey        0.480  15.5
## Mountain beaver   1.350   8.1
## Cow             465.000 423.0
## Grey wolf        36.330 119.5
## Goat             27.660 115.0
```

```r
#Run the regression of brain size against body size:
lm.mammals<-lm(brain~body, data=mammals)

#p-value by anova:
anova(lm.mammals)
```

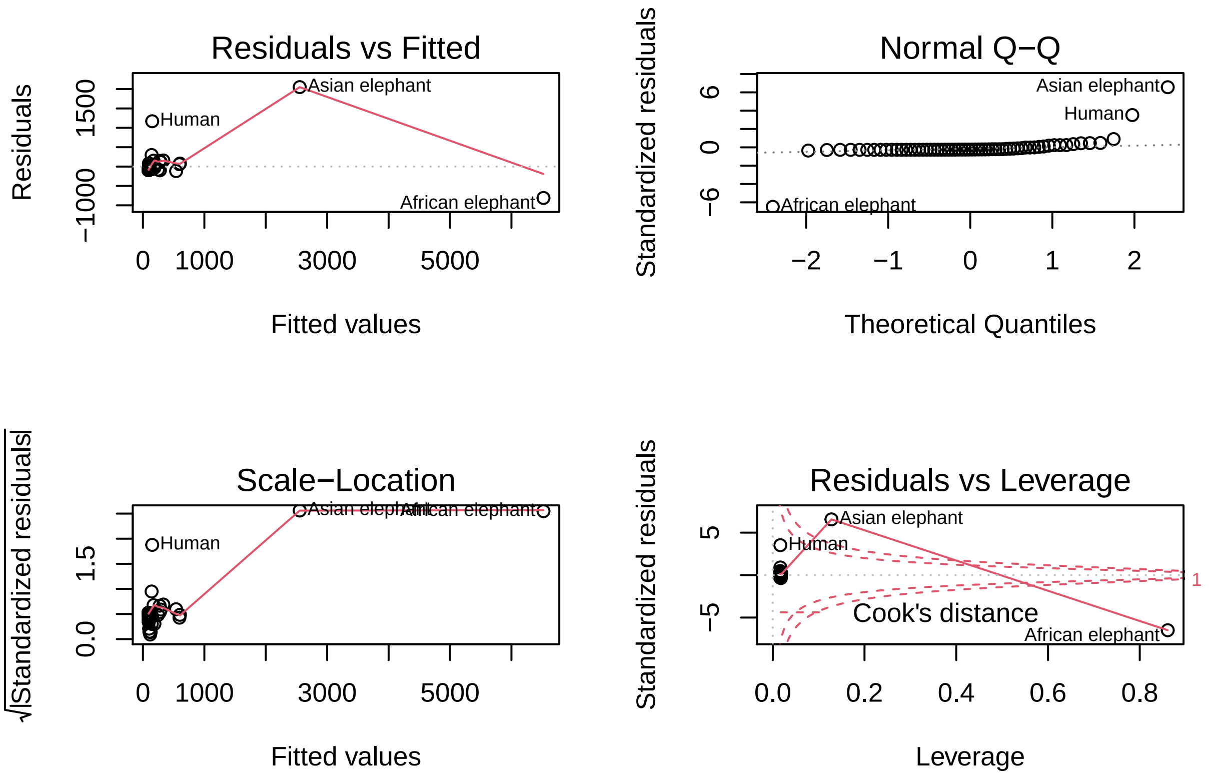
```
## Analysis of Variance Table
##
## Response: brain
##           Df   Sum Sq  Mean Sq F value    Pr(>F)
## body       1 46068314 46068314  411.19 < 2.2e-16 ***
## Residuals 60  6722239   112037
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
#slope (effect size), intercept, and p-value by summary:
summary(lm.mammals)
```

```
##
## Call:
## lm(formula = brain ~ body, data = mammals)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -810.07  -88.52  -79.64  -13.02 2050.33
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 91.00440   43.55258    2.09   0.0409 *
## body         0.96650    0.04766   20.28   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 334.7 on 60 degrees of freedom
## Multiple R-squared:  0.8727, Adjusted R-squared:  0.8705
## F-statistic: 411.2 on 1 and 60 DF,  p-value: < 2.2e-16
```

Model diagnostics

```
par(mfrow=c(2,2)) # plotting window is a 2x2 grid for four plots
plot(lm.mammals) #the two elephants are obvious outliers
```



```
shapiro.test(lm.mammals$residuals) #the residuals are not normally distributed
```

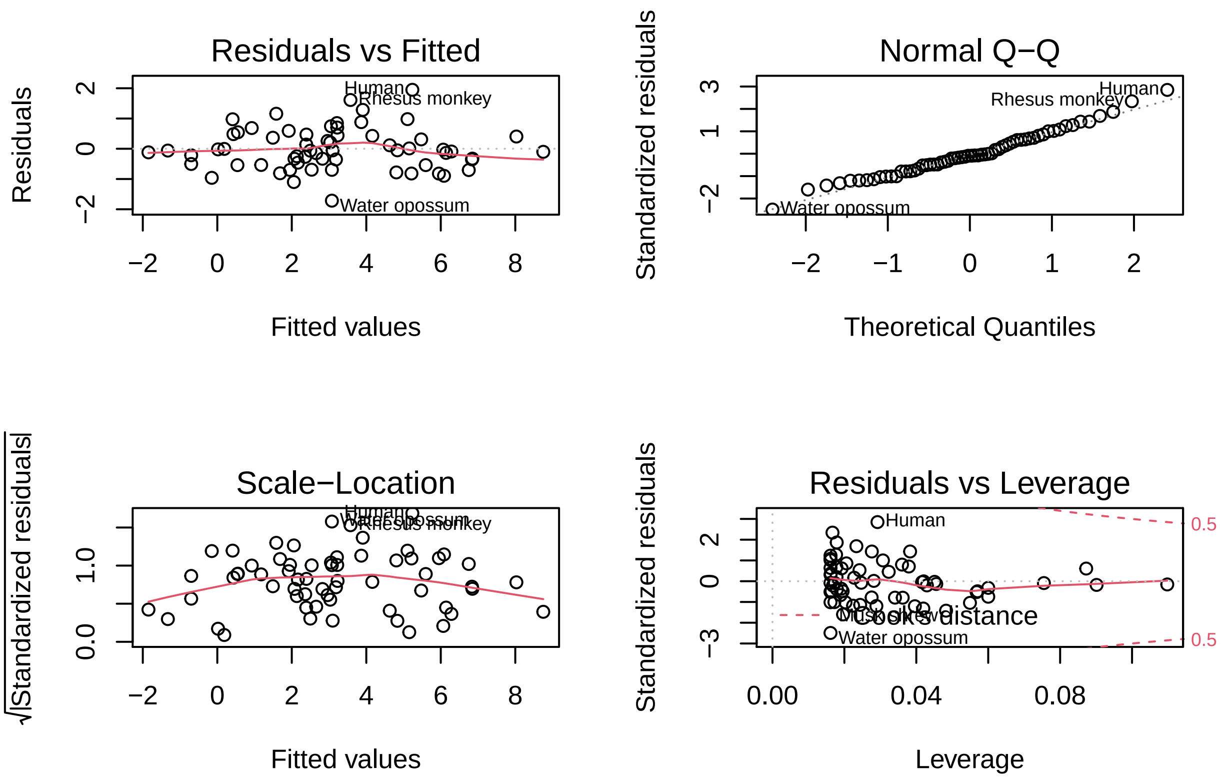
```
##
##  Shapiro-Wilk normality test
##
## data:  lm.mammals$residuals
## W = 0.41112, p-value = 2.316e-14
```

Regression on transformed data

The two elephants had such large body sizes they became outliers. Perhaps the relationship is better expressed between log(body size) and log(brain size). Taking the log reduces the influence of large observations (making the data less skewed).

```
#The linear regression with each variable log-transformed:
#in R, log means ln (ie base e).

lm.log_mammals<-lm(log(brain)~log(body), data=mammals)
```

```
par(mfrow=c(2,2))
plot(lm.log_mammals) # the skewness is reduced, the plots are easier to read
```



```
shapiro.test(lm.log_mammals$residuals) #the residuals are now normally distributed
```

```
##
##  Shapiro-Wilk normality test
##
## data:  lm.log_mammals$residuals
## W = 0.98268, p-value = 0.5293
```

```
#slope, intercept, and p-value:
summary(lm.log_mammals)
```

```
##
## Call:
## lm(formula = log(brain) ~ log(body), data = mammals)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.71550 -0.49228 -0.06162  0.43597  1.94829
##
## Coefficients:
```

```
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.13479    0.09604   22.23   <2e-16 ***
## log(body)    0.75169    0.02846   26.41   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6943 on 60 degrees of freedom
## Multiple R-squared:  0.9208, Adjusted R-squared:  0.9195
## F-statistic: 697.4 on 1 and 60 DF,  p-value: < 2.2e-16
```

Mutliple regression

Multiple regression is used to estimate the relationship between multiple explanatory variables and a continuous response variable.

The explanatory variables can be continuous or categorical. A mix of continuous and categorical explanatory variables is often called an ANCOVA.

Interactions among explanatory variables in how they affect the response can also be explored. Use: lm(response~X1+X2+X1*X2+…, data=name of data). Here X1, etc. refer to explanatory variables. The asterisk denotes an interaction.

Example: Predicting zooplankton biomass at Guadeloupe. A transect line was run across a reef at Guadeloupe and zooplanton biomass and environmental variables were measured.

Raw data available from http://www.esapubs.org/archive/ecol/E085/050/#suppl-1.htm#anchorFilelist

```
zooplankton_data<-read.csv("c:/Users/gjosg/Dropbox/R_Handbook_2020_UVic/Data/guadeloupe.csv", header=TRU

head(zooplankton_data)
```

```
##      Site Transect_Dist Ln_zoo_biomass_small Ln_zoo_biomass_large Diss_Oxygen
## 1 Site_01          0.00              1.04416               0.0334        7.70
## 2 Site_02          0.24              1.98965               0.7159        7.06
## 3 Site_03          0.36              1.33184              -0.0356        7.91
## 4 Site_04          0.60              1.10327               0.3514        7.56
## 5 Site_05          0.60              0.41871              -0.3624        7.18
## 6 Site_06          0.69              1.35841               0.4725        6.89
##   Salinity Wind_speed Phytoplankton_biomass Turbidity Swell_height
## 1    36.40        3.0                0.3781       1.5          0.2
## 2    36.86        3.4                0.7576       1.6          0.2
## 3    37.02        5.5                0.5857       1.6          0.3
## 4    36.92        6.7                0.7035       1.2          0.4
## 5    36.65        7.0                0.7035       1.2          0.4
## 6    36.85        6.7                0.5857       1.2          0.4
```

```
# Response variable: log-transformed zooplankton biomasses of two size classes
# Spatial variable: coordinate (km) of the sampling site along the transect.
# Environmental variables: dissolved oxygen (mg/L), salinity (psu), wind speed (m/s), phytoplankton bio

#First combine biomass of the two size classes into single biomass measurement:
zooplankton_data$Ln_zoo_biomass<-log(exp(zooplankton_data$Ln_zoo_biomass_small)+exp(zooplankton_data$Ln_

#Model zooplankton biomass as a function of distance along transect, dissolved oxygen, salinity, wind s
```

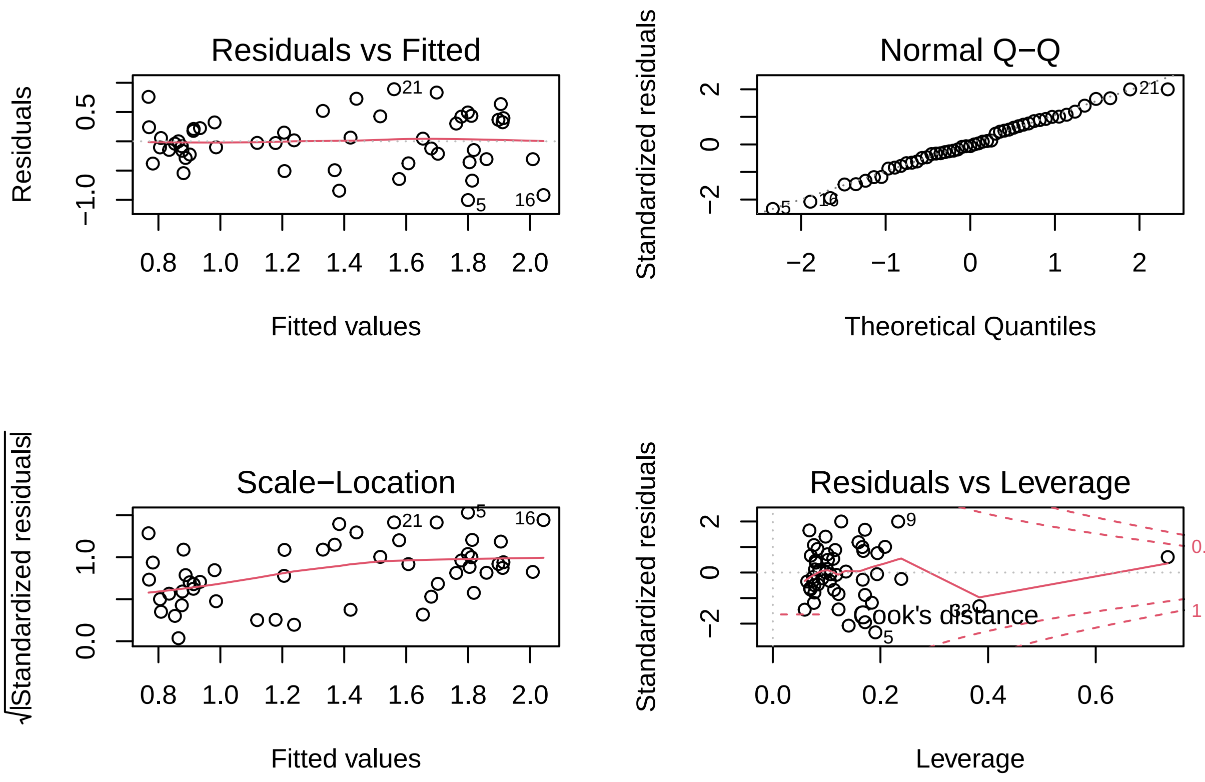
```r
#run the multiple regression model:
lm.multiple<-lm(Ln_zoo_biomass~Transect_Dist+Diss_Oxygen*Salinity+Wind_speed+Phytoplankton_biomass, data

#get the slopes, intercept, and p-values:
summary(lm.multiple)
```

```
##
## Call:
## lm(formula = Ln_zoo_biomass ~ Transect_Dist + Diss_Oxygen * Salinity +
##     Wind_speed + Phytoplankton_biomass, data = zooplankton_data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.00373 -0.29471 -0.02697  0.32406  0.88978
##
## Coefficients:
##                        Estimate Std. Error t value Pr(>|t|)
## (Intercept)           302.57544  194.30749   1.557   0.1266
## Transect_Dist          -0.16595    0.08923  -1.860   0.0696 .
## Diss_Oxygen           -43.50459   27.32714  -1.592   0.1185
## Salinity               -8.05392    5.22850  -1.540   0.1306
## Wind_speed              0.02065    0.03494   0.591   0.5575
## Phytoplankton_biomass  -0.07659    0.45382  -0.169   0.8668
## Diss_Oxygen:Salinity    1.16578    0.73533   1.585   0.1200
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4764 on 44 degrees of freedom
## Multiple R-squared:  0.4753, Adjusted R-squared:  0.4038
## F-statistic: 6.643 on 6 and 44 DF,  p-value: 4.723e-05
```

```r
anova(lm.multiple)
```

```
## Analysis of Variance Table
##
## Response: Ln_zoo_biomass
##                       Df Sum Sq Mean Sq F value    Pr(>F)
## Transect_Dist          1 7.0437  7.0437 31.0348 1.436e-06 ***
## Diss_Oxygen            1 1.2451  1.2451  5.4858   0.02376 *
## Salinity               1 0.1115  0.1115  0.4912   0.48707
## Wind_speed             1 0.0704  0.0704  0.3103   0.58032
## Phytoplankton_biomass  1 0.0054  0.0054  0.0236   0.87867
## Diss_Oxygen:Salinity   1 0.5705  0.5705  2.5135   0.12004
## Residuals             44 9.9863  0.2270
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
#model diagnostic plots
par(mfrow=c(2,2))
plot(lm.multiple)
```



```r
#normality of the residuals:
shapiro.test(lm.multiple$residuals)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  lm.multiple$residuals
## W = 0.9889, p-value = 0.9121
```

ANCOVA

ANCOVA is a special kind of linear regression where a continuous covariate with the potential to confound an effect of interest is controlled for in the analysis. YOu can also control for interactions between categorical and continuous variables.

ANCOVA assumptions: 1. Independence of data 2. Linear relationship between response and the continuous covariate 3. Normality of residuals 4. Variance is equal across for all values of categorical and continuous variables

Additionally, if you do not include an interaction between categorical and continuous explanatory variables: 5. Slope is the same across all levels of categorical factor.

Use: lm(y~X1+X2, data=data); y=response, X1 is categorical factor, X2 is continuous variable With inter-action: lm(y~X1*X2, data=data)

Example: naked mole rats differ in activity levels (workers vs lazy caste members). You would expect differences in energy expenditure between these two castes. However, the two castes may differ in body size, which would also affect energy expenditure. If lazy rats are bigger, they will expend more energy independent of their activity level. The effect of body size needs to be controlled to see the effect of caste membership.

```
MoleRats<-read.csv("c:/Users/gjosg/Dropbox/R_Handbook_2020_UVic/Data/MoleRats.csv", header=TRUE)
head(MoleRats)
```

```
##    caste   lnmass lnenergy
## 1 worker 3.850148 3.688879
## 2 worker 3.988984 3.688879
## 3 worker 4.110874 3.688879
## 4 worker 4.174387 3.663562
## 5 worker 4.248495 3.871201
## 6 worker 4.262680 3.850148
```

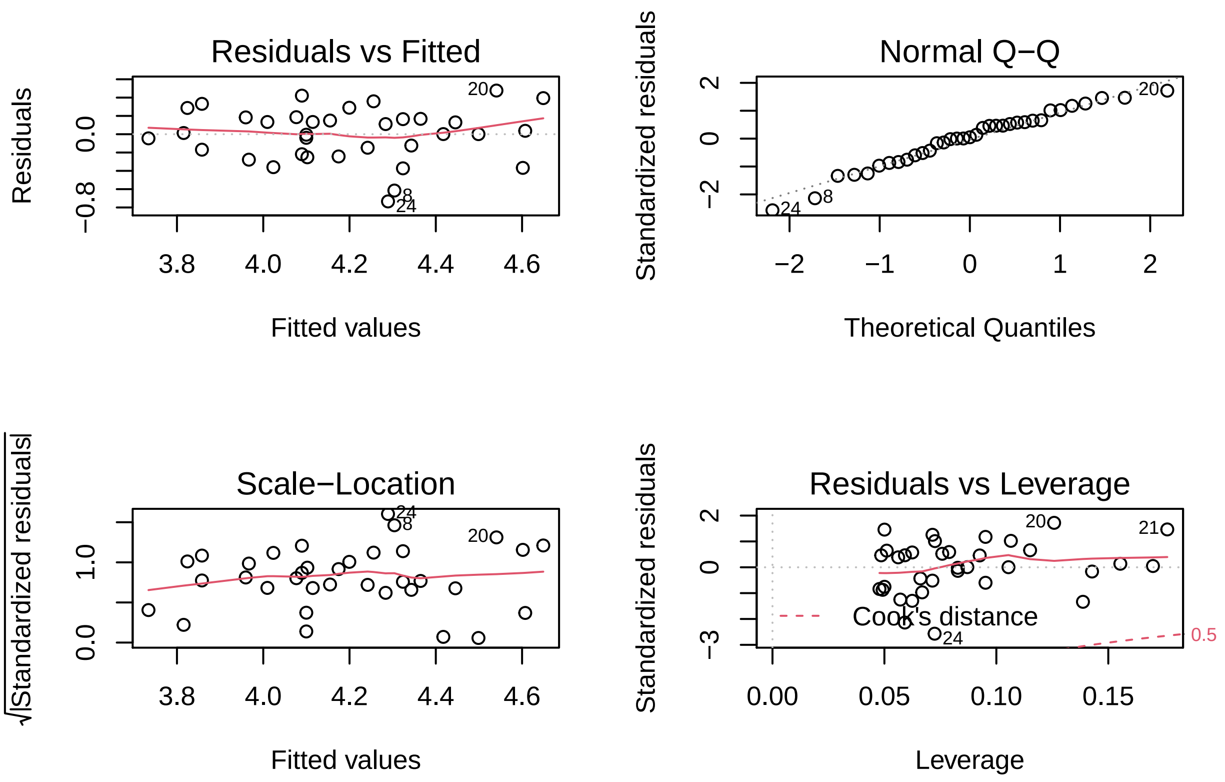
```
ancova.out<-lm(lnenergy~lnmass+caste, data=MoleRats) #no interaction
ancova.int<-lm(lnenergy~lnmass*caste, data=MoleRats) #interaction allows for different slope for differ

summary(ancova.int) #slopes, intercepts, p-values
```

```
##
## Call:
## lm(formula = lnenergy ~ lnmass * caste, data = MoleRats)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.72004 -0.17990  0.05631  0.19551  0.43128
##
## Coefficients:
##                     Estimate Std. Error t value Pr(>|t|)
## (Intercept)           1.2939     1.6691   0.775   0.4441
## lnmass                0.6069     0.3428   1.771   0.0865 .
## casteworker          -1.5713     1.9518  -0.805   0.4269
## lnmass:casteworker    0.4186     0.4147   1.009   0.3206
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2965 on 31 degrees of freedom
## Multiple R-squared:  0.4278, Adjusted R-squared:  0.3725
## F-statistic: 7.727 on 3 and 31 DF,  p-value: 0.0005391
```

```
summary(ancova.out) #slopes, intercepts, p-values
```

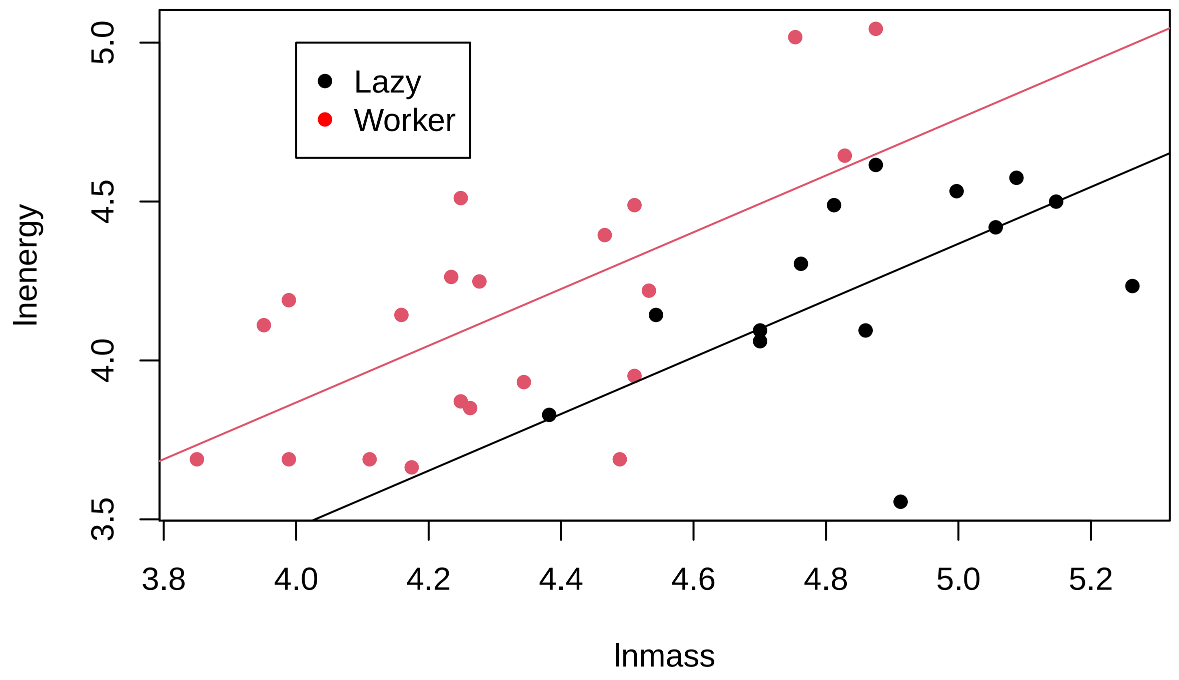
```
##
## Call:
## lm(formula = lnenergy ~ lnmass + caste, data = MoleRats)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.73388 -0.19371  0.01317  0.17578  0.47673
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.09687    0.94230  -0.103   0.9188
## lnmass       0.89282    0.19303   4.625 5.89e-05 ***
## casteworker  0.39334    0.14611   2.692   0.0112 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2966 on 32 degrees of freedom
## Multiple R-squared:  0.409,  Adjusted R-squared:  0.3721
## F-statistic: 11.07 on 2 and 32 DF,  p-value: 0.0002213
```

```
#Diagnostic plots
par(mfrow=c(2,2))
plot(ancova.out)
```



Plot of data for ANCOVA

```
plot(lnenergy~lnmass, col=as.factor(caste), data=MoleRats, pch=16) #I factor "caste", as that means it
abline(ancova.out$coefficients[1], ancova.out$coefficients[2]) # line for lazy mole rats (which are the
abline(ancova.out$coefficients[1]+ancova.out$coefficients[3], ancova.out$coefficients[2], col=2) #adds
legend(4,5,c("Lazy", "Worker"), col=c("black", "red"), pch=16) #adds a legend at x=4 y=5
```



Generalized linear models

Generalized linear models (GLMs) model relationships in data that are not normal. For instance, count data inherently have a variance that increases with the mean. Modelling based on the Poisson or negative binomial distribution is more appropriate than the normal distribution.

Poisson regression

The Poisson distribution is for count data where the variance equals the mean.

Use: glm(y~X1+X2+…, family = "poisson", data = name of data) where y = response and X1, etc. = explanatory variables.

Example: Understand how counts of scallopped hammerhead sharks made at a seamount in the Pacific have changed through time and what relationship they have to sea surface temperature (SST) and a measure of the El Nino (ONI) while controlling for visibility.

```
hammerhead_sharks<-read.csv("c:/Users/gjosg/Dropbox/R_Handbook_2020_UVic/Data/Hammerheads.csv", header=
head(hammerhead_sharks)
```

```
##   Year SiteCode Visibility DiverCode Hammerheads   SST   ONI
## 1 2012      110         12         9           1 29.40 -0.38
## 2 2018      103         15         3          20 29.53 -0.41
## 3 2016      103         12         4          15 27.13 -0.74
## 4 2007      101         30         3          20 29.18  0.32
## 5 2019      112         24        43          40 27.77  0.13
## 6 2007      111          9         3          15 25.53 -1.60
```

```
#Run the GLM:
glm.poisson<-glm(Hammerheads~Year+SST+ONI+Visibility, family = "poisson", data=hammerhead_sharks, na.ac

#Get the coefficients (intercept, slopes) and p-values from a Wald's Z-test:
summary(glm.poisson)
```

```
##
## Call:
## glm(formula = Hammerheads ~ Year + SST + ONI + Visibility, family = "poisson",
##     data = hammerhead_sharks, na.action = "na.fail")
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -14.7753   -5.9476   -3.4142    0.6047   27.3578
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) 77.9619713  4.2585183   18.31   <2e-16 ***
## Year        -0.0374011  0.0021315  -17.55   <2e-16 ***
## SST         -0.0007929  0.0113555   -0.07    0.944
## ONI         -0.4120340  0.0164040  -25.12   <2e-16 ***
## Visibility   0.0184108  0.0014657   12.56   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 13827  on 269  degrees of freedom
## Residual deviance: 12164  on 265  degrees of freedom
## AIC: 13124
##
## Number of Fisher Scoring iterations: 6
```

```
#Get the AIC:
AIC(glm.poisson)
```

```
## [1] 13123.66
```

```
#Compare AIC among suite of candidate models:
#install.packages("MuMIn")
library(MuMIn) #dredge function is in MuMIn

glm.poisson.dredge<-dredge(glm.poisson) #list of all possible sub-models (each differing in one explana
```

```
## Fixed term is "(Intercept)"
```

```
glm.poisson.dredge
```

```
## Global model call: glm(formula = Hammerheads ~ Year + SST + ONI + Visibility, family = "poisson",
##     data = hammerhead_sharks, na.action = "na.fail")
## ---
## Model selection table
##    (Intrc)     ONI        SST   Vsblt     Year df    logLik    AICc   delta
## 14  77.980 -0.4124            0.01840 -0.03742  4 -6556.834 13121.8    0.00
## 16  77.960 -0.4120 -0.0007929 0.01841 -0.03740  5 -6556.832 13123.9    2.07
## 10  96.690 -0.4220                    -0.04656  3 -6629.686 13265.5  143.64
## 12  96.820 -0.4257  0.0085170         -0.04674  4 -6629.398 13266.9  145.13
## 8    3.441 -0.4450 -0.0292000 0.02775           4 -6709.064 13426.3  304.46
## 6    2.628 -0.4586            0.02775           3 -6712.380 13430.8  309.03
## 15  88.280         -0.0893000 0.02120 -0.04130  4 -6888.798 13785.7  663.93
## 4    3.850 -0.4741 -0.0246800                   3 -6899.962 13806.0  684.19
## 2    3.163 -0.4862                              2 -6902.370 13808.8  686.96
## 13  91.350                    0.02082 -0.04406  3 -6920.485 13847.1  725.24
## 11 107.400         -0.0808900         -0.05073  3 -6978.901 13963.9  842.07
## 9  109.700                            -0.05299  2 -7005.542 14015.1  893.31
## 7    6.331         -0.1323000 0.03133           3 -7090.645 14187.4 1065.56
## 5    2.648                    0.03201           2 -7160.911 14325.9 1204.05
## 3    6.975         -0.1333000                   2 -7315.105 14634.3 1512.44
## 1    3.276                                      1 -7388.211 14778.4 1656.62
##    weight
## 14  0.738
## 16  0.262
## 10  0.000
## 12  0.000
## 8   0.000
## 6   0.000
## 15  0.000
## 4   0.000
## 2   0.000
## 13  0.000
## 11  0.000
## 9   0.000
## 7   0.000
## 5   0.000
## 3   0.000
## 1   0.000
## Models ranked by AICc(x)
```

Negative Binomial Models

When the variance of count data increases faster than the mean, the data are "overdispersed."

The negative binomial distribution lets variance increase faster than the mean, and negative binomial models model this variance explicity with a dispersion parameter.

The glm function does not include the negative binomial family, so use the glm.nb() function from the MASS package.

Use: glm.nb(y~X1+X2+. . . , data= name of data) where y = response and X1, etc. = explanatory variables. No need for the "family" argument.

```r
library(MASS)
hammerhead_sharks<-read.csv("c:/Users/gjosg/Dropbox/R_Handbook_2020_UVic/Data/Hammerheads.csv", header=
head(hammerhead_sharks)
```

```
##   Year SiteCode Visibility DiverCode Hammerheads   SST   ONI
## 1 2012      110         12         9           1 29.40 -0.38
## 2 2018      103         15         3          20 29.53 -0.41
## 3 2016      103         12         4          15 27.13 -0.74
## 4 2007      101         30         3          20 29.18  0.32
## 5 2019      112         24        43          40 27.77  0.13
## 6 2007      111          9         3          15 25.53 -1.60
```

```r
#Run the GLM:
glm.negbin<-glm.nb(Hammerheads~Year+SST+ONI+Visibility, data=hammerhead_sharks, na.action = "na.fail")

#Get the coefficients (intercept, slopes) and p-values from a Wald's Z-test:
summary(glm.negbin)
```

```
##
## Call:
## glm.nb(formula = Hammerheads ~ Year + SST + ONI + Visibility,
##     data = hammerhead_sharks, na.action = "na.fail", init.theta = 0.3878566926,
##     link = log)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -1.95444  -1.23959  -0.51512   0.09476   2.00302
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) 60.77788   35.31081   1.721 0.085210 .
## Year        -0.02710    0.01770  -1.531 0.125739
## SST         -0.13958    0.09636  -1.449 0.147455
## ONI         -0.42826    0.12985  -3.298 0.000974 ***
## Visibility   0.03533    0.01490   2.372 0.017707 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for Negative Binomial(0.3879) family taken to be 1)
##
##     Null deviance: 340.97  on 269  degrees of freedom
## Residual deviance: 313.89  on 265  degrees of freedom
## AIC: 2120.4
##
## Number of Fisher Scoring iterations: 1
##
##
##               Theta:  0.3879
##           Std. Err.:  0.0337
##
##  2 x log-likelihood:  -2108.3870
```

```
#Get the AIC and compare to Poisson model (since Poisson model is nested within negative binomial model
AIC(glm.negbin, glm.poisson) # AIC of negative binomial model is much lower, indicating it has a better
```

```
##             df       AIC
## glm.negbin   6  2120.387
## glm.poisson  5 13123.663
```

```
#Compare AIC among suite of candidate models:
#dredge function is in MuMIn package

glm.negbin.dredge<-dredge(glm.negbin) #list of all possible sub-models (each differing in one explanato
```

```
## Fixed term is "(Intercept)"
```

```
glm.negbin.dredge
```

```
## Global model call: glm.nb(formula = Hammerheads ~ Year + SST + ONI + Visibility,
##     data = hammerhead_sharks, na.action = "na.fail", init.theta = 0.3878566926,
##     link = log)
## ---
## Model selection table
##    (Intrc)     ONI      SST   Vsblt     Year df    logLik   AICc delta weight
## 14  61.240 -0.4562          0.03107 -0.02922  5 -1054.962 2120.2  0.00  0.264
## 16  60.780 -0.4283 -0.13960 0.03533 -0.02710  6 -1054.194 2120.7  0.56  0.200
## 6    2.423 -0.5067          0.03735           4 -1056.462 2121.1  0.92  0.166
## 8    6.759 -0.4681 -0.15910 0.04185           5 -1055.465 2121.2  1.01  0.160
## 10  75.720 -0.4640                  -0.03612  4 -1056.932 2122.0  1.87  0.104
## 12  76.990 -0.4481 -0.08953         -0.03552  5 -1056.612 2123.5  3.30  0.051
## 2    3.149 -0.5427                            3 -1059.329 2124.7  4.60  0.026
## 4    5.956 -0.5213 -0.10100                   4 -1058.927 2126.0  5.85  0.014
## 15  83.280         -0.19390 0.03858 -0.03753  5 -1058.687 2127.6  7.45  0.006
## 13  88.120                  0.03209 -0.04256  4 -1060.234 2128.6  8.47  0.004
## 7    8.766         -0.23330 0.04998           4 -1060.969 2130.1  9.94  0.002
## 9  107.500                          -0.05189  3 -1062.199 2130.5 10.34  0.002
## 11 106.800         -0.13910         -0.04960  4 -1061.397 2130.9 10.79  0.001
## 5    2.405                  0.04429           3 -1063.263 2132.6 12.46  0.001
## 3    8.221         -0.17820                   3 -1065.811 2137.7 17.56  0.000
## 1    3.276                                    2 -1067.116 2138.3 18.13  0.000
## Models ranked by AICc(x)
```

Binomial models (ie logistic regression)

Binomial distribution represents data that are counts of "successes" from a total number of trials or yes/no (ie successes from one trial - called the Bernoulli trial).

Bernoulli use: glm(y~X1+X2+. . . , family = "binomial", data= name of data)

Bernoulli example: Relationship between whether or not (1 or 0) dengue was recorded any time between 1961 and 1990 and humidity, temperature, and tree cover in the DAAG package.

Binomial use: glm(y~X1+X2+. . . , weights=w, family = "binomial", data= name of data) where "w" is the number of "trials" (ie the maximum possible count) and y is the proportion (ie between 0 and 1) of those total number of trails that were "successes."

Binomial example: Test if an experimental herbicide is effective in removing invasive species. Look for the species at 10 replicate sites in each of the pesticide and control treatments. Data are number of quadrats that included the invasive species, as well as annual precipitation per site as a potential confounding variable. There were 15 quadrats maximum per site, but some sites received fewer quadrats.

```
#install.packages("DAAG")
library(DAAG)
```

```
## Loading required package: lattice

##
## Attaching package: 'DAAG'

## The following object is masked from 'package:car':
##
##     vif

## The following object is masked from 'package:MASS':
##
##     hills
```

```
#load dengue data
data(dengue)
head(dengue)
```

```
##       humid  humid90      temp    temp90   h10pix h10pix90 trees trees90 NoYes
## 1 0.6713889 4.416667  2.037500  8.470835 17.35653 17.80861     0     1.5     0
## 2 7.6483340 8.167500 12.325000 14.925000 10.98361 11.69167     0     1.0     0
## 3 6.9790556 9.563058  6.925000 14.591660 17.50833 17.62528     0     1.2     0
## 4 1.1104163 1.825361  4.641665  6.046669 17.41763 17.51694     0     0.6     0
## 5 9.0270555 9.742751 18.175000 19.710000 13.84306 13.84306     0     0.0     0
## 6 8.9141113 9.516778 11.900000 16.643341 11.69167 11.69167     0     0.2     0
##   Xmin Xmax Ymin Ymax
## 1 70.5 74.5 38.0 35.5
## 2 62.5 64.5 35.5 34.5
## 3 68.5 69.5 36.0 35.0
## 4 67.0 68.0 35.0 34.0
## 5 61.0 64.5 33.5 32.0
## 6 64.5 65.5 36.5 35.0
```

```
#Bernoulli example:
#Run the GLM for Bernoulli data:
glm.bernoulli<-glm(NoYes~humid+temp+trees, family = "binomial", data=dengue) #model the relationship be

#Get the coefficients (intercept, slopes) and p-values from a Wald's Z-test:
summary(glm.bernoulli)
```

```
##
## Call:
## glm(formula = NoYes ~ humid + temp + trees, family = "binomial",
##     data = dengue)
```

```
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.7495  -0.3907  -0.2034   0.4829   3.5559
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -6.621563   0.309908 -21.366   <2e-16 ***
## humid        0.298371   0.023364  12.770   <2e-16 ***
## temp         0.044742   0.021597   2.072   0.0383 *
## trees        0.001983   0.003645   0.544   0.5865
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2695.4  on 1985  degrees of freedom
## Residual deviance: 1348.1  on 1982  degrees of freedom
##   (14 observations deleted due to missingness)
## AIC: 1356.1
##
## Number of Fisher Scoring iterations: 6
```

```
#Get the AIC:
AIC(glm.bernoulli)
```

```
## [1] 1356.137
```

```
#Binomial example:
invasive_data<-data.frame(Treatment=rep(c("Pesticide", "Control"), rep(10,2)), Number_of_Quadrats=c(15,
invasive_data$ProportionInvasive<-invasive_data$InvasiveSpecies/invasive_data$Number_of_Quadrats #creat

#Run the GLM for Bernoulli data:
glm.binomial<-glm(ProportionInvasive~Treatment+Precip, weights=Number_of_Quadrats, family = "binomial",

#Get the coefficients (intercept, slopes) and p-values from a Wald's Z-test:
summary(glm.binomial)
```

```
##
## Call:
## glm(formula = ProportionInvasive ~ Treatment + Precip, family = "binomial",
##     data = invasive_data, weights = Number_of_Quadrats)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.1495  -1.2007  -0.1489   0.8735   3.2182
##
## Coefficients:
##                      Estimate Std. Error z value Pr(>|z|)
## (Intercept)         0.5185172  0.3700638   1.401    0.161
## TreatmentPesticide -1.2049797  0.2619744  -4.600 4.23e-06 ***
## Precip             -0.0004571  0.0003594  -1.272    0.203
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 60.171  on 19  degrees of freedom
## Residual deviance: 37.590  on 17  degrees of freedom
## AIC: 99.755
##
## Number of Fisher Scoring iterations: 4
```

```
#Get the AIC:
AIC(glm.binomial)
```

```
## [1] 99.75461
```

Polynomial regression

You can model a polynomial relationship between a response and an explanatory variable using the poly() and I() functions.

poly(x, exponent) is for orthogonal polynomials (ensure polynomial terms are independent). I() is for regular polynomials.

Use: For quadratic 1. lm(y~poly(X1,2)+. . . , data=data name) 2. lm(y~X1+I(X1ˆ2)+. . . , data=data name)

For cubic: 1. lm(y~poly(X1,3)+. . . , data=data name) 2. lm(y~X1+I(X1$^{3)+I(X1}$3)+. . . , data=data name)

These functions can be expanded to higher degrees by changing the exponent in poly or adding more X terms in I().

The function can also be used in glm(). The example below expands on the scallopped hammerhead shark example from Poisson generalized linear models, now assuming a quadratic relationship between hammerhead count and sea surface temperature (SST).

```
hammerhead_sharks<-read.csv("c:/Users/gjosg/Dropbox/R_Handbook_2020_UVic/Data/Hammerheads.csv", header='
head(hammerhead_sharks)
```

```
##   Year SiteCode Visibility DiverCode Hammerheads   SST   ONI
## 1 2012      110         12         9           1 29.40 -0.38
## 2 2018      103         15         3          20 29.53 -0.41
## 3 2016      103         12         4          15 27.13 -0.74
## 4 2007      101         30         3          20 29.18  0.32
## 5 2019      112         24        43          40 27.77  0.13
## 6 2007      111          9         3          15 25.53 -1.60
```

```
#Run the GLM (including original without quadratic for comparison):

#Orthogonal polynomial regression:
glm.poisson.quad.orth<-glm(Hammerheads~Year+poly(SST,2)+ONI+Visibility, family = "poisson", data=hammerl

#Regular (not orthagonal) polynomial regression:
glm.poisson.quad<-glm(Hammerheads~Year+SST+I(SST^2)+ONI+Visibility, family = "poisson", data=hammerhead_

glm.poisson<-glm(Hammerheads~Year+SST+ONI+Visibility, family = "poisson", data=hammerhead_sharks, na.ac
```

```r
#Get the coefficients (intercept, slopes) and p-values from a Wald's Z-test:
summary(glm.poisson.quad.orth)
```

```
##
## Call:
## glm(formula = Hammerheads ~ Year + poly(SST, 2) + ONI + Visibility,
##     family = "poisson", data = hammerhead_sharks, na.action = "na.fail")
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -14.373   -5.784   -2.997    1.267   25.262
##
## Coefficients:
##                 Estimate Std. Error z value Pr(>|z|)
## (Intercept)    83.231993   4.334871  19.201  < 2e-16 ***
## Year           -0.040035   0.002153 -18.594  < 2e-16 ***
## poly(SST, 2)1 -0.852274   0.240575  -3.543 0.000396 ***
## poly(SST, 2)2 -5.658683   0.255964 -22.107  < 2e-16 ***
## ONI           -0.504934   0.017155 -29.433  < 2e-16 ***
## Visibility     0.015002   0.001490  10.068  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 13827  on 269  degrees of freedom
## Residual deviance: 11572  on 264  degrees of freedom
## AIC: 12533
##
## Number of Fisher Scoring iterations: 6
```

```r
summary(glm.poisson.quad)
```

```
##
## Call:
## glm(formula = Hammerheads ~ Year + SST + I(SST^2) + ONI + Visibility,
##     family = "poisson", data = hammerhead_sharks, na.action = "na.fail")
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -14.373   -5.784   -2.997    1.267   25.262
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.073e+02  9.502e+00  -11.29   <2e-16 ***
## Year        -4.004e-02  2.153e-03  -18.59   <2e-16 ***
## SST          1.370e+01  6.196e-01   22.10   <2e-16 ***
## I(SST^2)    -2.458e-01  1.112e-02  -22.11   <2e-16 ***
## ONI         -5.049e-01  1.716e-02  -29.43   <2e-16 ***
## Visibility   1.500e-02  1.490e-03   10.07   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 13827  on 269  degrees of freedom
## Residual deviance: 11572  on 264  degrees of freedom
## AIC: 12533
##
## Number of Fisher Scoring iterations: 6
```

```
#Get the AIC:
AIC(glm.poisson, glm.poisson.quad.orth, glm.poisson.quad) #notice the choice of orthogonal or not does
```

```
##                       df      AIC
## glm.poisson            5 13123.66
## glm.poisson.quad.orth  6 12533.30
## glm.poisson.quad       6 12533.30
```

```
#Compare AIC among suite of candidate models:
#install.packages("MuMIn")
library(MuMIn) #dredge function is in MuMIn

glm.poisson.dredge.quad<-dredge(glm.poisson.quad.orth) #list of all possible sub-models (each differing
```

```
## Fixed term is "(Intercept)"
```

```
glm.poisson.dredge.quad
```

```
## Global model call: glm(formula = Hammerheads ~ Year + poly(SST, 2) + ONI + Visibility,
##     family = "poisson", data = hammerhead_sharks, na.action = "na.fail")
## ---
## Model selection table
##      (Int)     ONI ply(SST,2)     Vsb      Yer df    logLik    AICc   delta
## 16  83.230 -0.5049          + 0.01500 -0.04003  6 -6260.652 12533.6    0.00
## 12  99.700 -0.5174          +         -0.04809  5 -6308.077 12626.4   92.76
## 8    2.609 -0.5419          + 0.02551           5 -6432.009 12874.2  340.62
## 14  77.980 -0.4124            0.01840 -0.03742  4 -6556.834 13121.8  588.20
## 4    3.099 -0.5728          +                   4 -6593.873 13195.9  662.27
## 10  96.690 -0.4220                    -0.04656  3 -6629.686 13265.5  731.84
## 6    2.628 -0.4586            0.02775           3 -6712.380 13430.8  897.23
## 15  91.520                  + 0.01858 -0.04414  5 -6723.126 13456.5  922.86
## 11 109.200                  +         -0.05274  4 -6790.485 13589.1 1055.50
## 2    3.163 -0.4862                              2 -6902.370 13808.8 1275.16
## 13  91.350                    0.02082 -0.04406  3 -6920.485 13847.1 1313.44
## 7    2.652                  + 0.02992           4 -6952.472 13913.1 1379.47
## 9  109.700                            -0.05299  2 -7005.542 14015.1 1481.51
## 3    3.237                  +                   3 -7155.443 14317.0 1783.35
## 5    2.648                    0.03201           2 -7160.911 14325.9 1792.24
## 1    3.276                                      1 -7388.211 14778.4 2244.81
##    weight
## 16      1
## 12      0
## 8       0
## 14      0
## 4       0
## 10      0
## 6       0
## 15      0
## 11      0
## 2       0
## 13      0
## 7       0
## 9       0
## 3       0
## 5       0
## 1       0
## Models ranked by AICc(x)
```

Generalized additive models

Generalized additive models (GAMs) are useful when a non-linear relationship is expected between the response and an explanatory variable, but the exact nature of that relationship is unknown. GAMs fit “splines” to the data - linear or polynomial (ie cubic) relationships to sections of the data that are joined at the “knots” (where the data were divided).

Use: gam(y~s(X1)+…, family=“poisson”, data=data name) where s(X1) means we are fitting a model with splines/non-linear relationship to X1. Can change “poisson” to “gaussian” (for normal) or “binomial” or “negbin” or other distributions.

The gam() function is in the mgcv package.

```
#install.packages("mgcv")
library(mgcv)
```

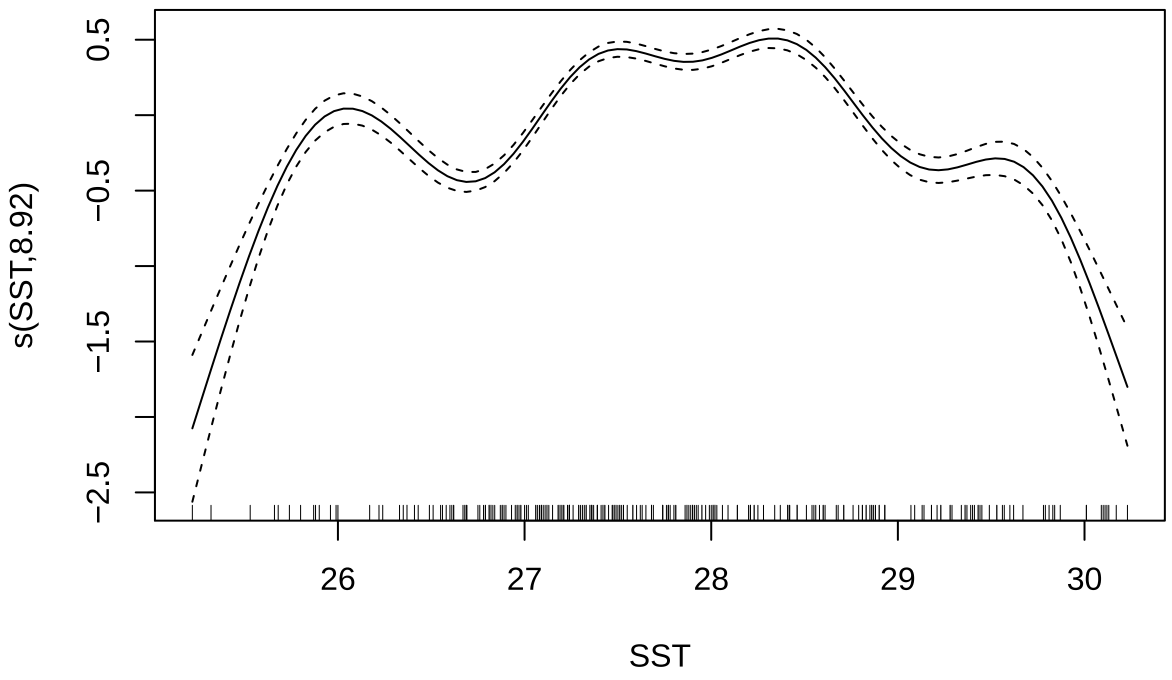
```
## Loading required package: nlme
```

```
## This is mgcv 1.8-31. For overview type 'help("mgcv-package")'.
```

```
hammerhead_sharks<-read.csv("c:/Users/gjosg/Dropbox/R_Handbook_2020_UVic/Data/Hammerheads.csv", header=
head(hammerhead_sharks)
```

```
##   Year SiteCode Visibility DiverCode Hammerheads   SST   ONI
## 1 2012      110         12         9           1 29.40 -0.38
## 2 2018      103         15         3          20 29.53 -0.41
## 3 2016      103         12         4          15 27.13 -0.74
## 4 2007      101         30         3          20 29.18  0.32
## 5 2019      112         24        43          40 27.77  0.13
## 6 2007      111          9         3          15 25.53 -1.60
```

```
#Run the GLM:
gam.out<-gam(Hammerheads~Year+s(SST)+ONI+Visibility, family = "poisson", data=hammerhead_sharks) #fit S

#Visualize the non-linear relationship fit between hammerhead count and SST:
plot(gam.out)
```



```r
#Get the coefficients (intercept, slopes) and p-values from a Wald's Z-test:
summary(gam.out)
```

```
##
## Family: poisson
## Link function: log
##
## Formula:
## Hammerheads ~ Year + s(SST) + ONI + Visibility
##
## Parametric coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) 78.675546   4.396564   17.89   <2e-16 ***
## Year        -0.037819   0.002183  -17.32   <2e-16 ***
## ONI         -0.551856   0.018054  -30.57   <2e-16 ***
## Visibility   0.018219   0.001591   11.45   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##          edf Ref.df Chi.sq p-value
## s(SST) 8.916  8.998  825.9  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## R-sq.(adj) =  0.0953   Deviance explained =   19%
## UBRE = 40.558  Scale est. = 1         n = 270
```

```
#Get the AIC:
AIC(gam.out)
```

```
## [1] 12169.96
```

Non-linear models

Models that are linear in the parameters can be fit with gam or polynomials using lm() if non-linear relationships exist between response and explanatory variables, but models that are non-linear in the parameters need another function - nls().

Use: nls(y~"insert non-linear equation here", start=list("list starting values for parameters"), data=data name). The starting values are rough guesses to get the function started in fitting the model. See the example below. The "y" needs to be the name of your response variable in your data frame. The x value specified in your equation/formula needs to be called the same thing as its column in the data frame.

Example: fit a von-Bertalanffy growth curve to sturgeon length-age data.

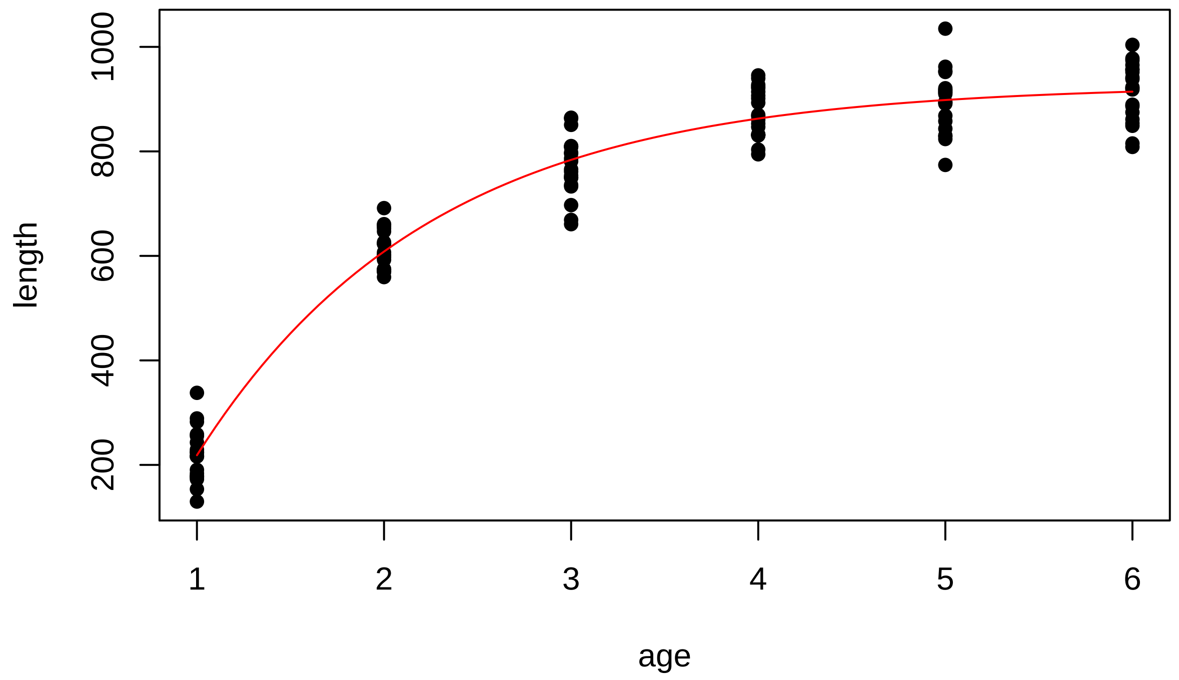
The von-Bertalanffy function is: Length (at age) = L_infinity(1-exp(-k(age-t0))) where t is the age and t0 is the age when length is zero and L_infinity is asymptotic maximum length.

```
#make the von-Bertalanffy function

#Simulate some data for the example:
von_Bert<-function(age, t0, L_infinity, k) {L_infinity*(1-exp(-k*(age-t0)))} #von Bert function for sim
salmon<-data.frame(age=rep(1:6, 20), length=rnorm(120, von_Bert(age=c(1:6), t0=0.63, L_infinity=927, k=

nonlin_mod<-nls(length~L_infinity*(1-exp(-k*(age-t0))),start=list(t0=0.2,L_infinity=1000,k=1), data=sal

summary(nonlin_mod) #get estimates and significance for each parameter
```

```
##
## Formula: length ~ L_infinity * (1 - exp(-k * (age - t0)))
##
## Parameters:
##             Estimate Std. Error t value Pr(>|t|)
## t0           0.66359    0.02936   22.60   <2e-16 ***
## L_infinity 927.36458   10.36607   89.46   <2e-16 ***
## k            0.79846    0.04288   18.62   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 51.51 on 117 degrees of freedom
##
## Number of iterations to convergence: 4
## Achieved convergence tolerance: 9.119e-08
```

```
#Plot the data and the curve:
plot(length~age, data=salmon, pch=16)
#get the model coefficients to plot the curve specified by the nls model:
```

```
t0_estimate<-coef(nonlin_mod)[1] #t0 is the first coefficient in coef(nonlin_mod)
L_infinity_estimate<-coef(nonlin_mod)[2] #L_infinity is the second coefficient in coef(nonlin_mod)
k_estimate<-coef(nonlin_mod)[3] #k is the third coefficient in coef(nonlin_mod)
age_x<-c(1:6) #want to plot for ages 1 to 6
curve(L_infinity_estimate*(1-exp(-k_estimate*(x-t0_estimate))), from=1, to=6, add=TRUE, col="red") #add
```



```
#Get the AIC:
AIC(nonlin_mod)
```

```
## [1] 1291.547
```

AIC and Likelihood Ratio Tests

AIC can be used to compare numerous candidate models based on their fit (likelihood), penalized by number of parameters to account for overfitting. The dredge function in the MuMIn package will calculate the AIC for all possible subsets of your model, and include delta-AICs and model weights.

Likelihood ratio tests are another way to assess significance of a parameter (besides t-tests/Wald z-tests provided by summary function). They will compare the increase in likelihood from a simpler model to a more complex one and give a p-value representing if that increase represents a significant increase in fit (meaning the model explains something more biologically) or if it is just overfitting. Models must be nested.

```
hammerhead_sharks<-read.csv("c:/Users/gjosg/Dropbox/R_Handbook_2020_UVic/Data/Hammerheads.csv", header=
head(hammerhead_sharks)
```

```
##   Year SiteCode Visibility DiverCode Hammerheads   SST   ONI
## 1 2012      110         12         9           1 29.40 -0.38
## 2 2018      103         15         3          20 29.53 -0.41
## 3 2016      103         12         4          15 27.13 -0.74
## 4 2007      101         30         3          20 29.18  0.32
## 5 2019      112         24        43          40 27.77  0.13
## 6 2007      111          9         3          15 25.53 -1.60
```

```
#Run the GLM:
glm.poisson<-glm(Hammerheads~Year+SST+ONI+Visibility, family = "poisson", data=hammerhead_sharks, na.ac

#Use the function AIC() to get the AIC of one or more models:
AIC(glm.poisson, update(glm.poisson, ~.-Year)) #compare AIC of full model to updated model lacking Year
```

```
##                                  df      AIC
## glm.poisson                       5 13123.66
## update(glm.poisson, ~. - Year)   4 13426.13
```

```
#dredge for the best candidate model among all possible subsets of your full model:
library(MuMIn)
dredge(glm.poisson)
```

```
## Fixed term is "(Intercept)"
```

```
## Global model call: glm(formula = Hammerheads ~ Year + SST + ONI + Visibility, family = "poisson",
##     data = hammerhead_sharks, na.action = "na.fail")
## ---
## Model selection table
##    (Intrc)     ONI        SST   Vsblt     Year df    logLik    AICc   delta
## 14  77.980 -0.4124            0.01840 -0.03742  4 -6556.834 13121.8    0.00
## 16  77.960 -0.4120 -0.0007929 0.01841 -0.03740  5 -6556.832 13123.9    2.07
## 10  96.690 -0.4220                    -0.04656  3 -6629.686 13265.5  143.64
## 12  96.820 -0.4257  0.0085170         -0.04674  4 -6629.398 13266.9  145.13
## 8    3.441 -0.4450 -0.0292000 0.02775           4 -6709.064 13426.3  304.46
## 6    2.628 -0.4586            0.02775           3 -6712.380 13430.8  309.03
## 15  88.280         -0.0893000 0.02120 -0.04130  4 -6888.798 13785.7  663.93
## 4    3.850 -0.4741 -0.0246800                   3 -6899.962 13806.0  684.19
## 2    3.163 -0.4862                              2 -6902.370 13808.8  686.96
## 13  91.350                    0.02082 -0.04406  3 -6920.485 13847.1  725.24
## 11 107.400         -0.0808900         -0.05073  3 -6978.901 13963.9  842.07
## 9  109.700                            -0.05299  2 -7005.542 14015.1  893.31
## 7    6.331         -0.1323000 0.03133           3 -7090.645 14187.4 1065.56
## 5    2.648                    0.03201           2 -7160.911 14325.9 1204.05
## 3    6.975         -0.1333000                   2 -7315.105 14634.3 1512.44
## 1    3.276                                      1 -7388.211 14778.4 1656.62
##    weight
## 14  0.738
## 16  0.262
## 10  0.000
## 12  0.000
## 8   0.000
## 6   0.000
## 15  0.000
## 4   0.000
## 2   0.000
## 13  0.000
## 11  0.000
## 9   0.000
## 7   0.000
## 5   0.000
## 3   0.000
## 1   0.000
## Models ranked by AICc(x)
```

Permutation tests

Permutation tests are a type of non-parametric test used when the assumptions of a parametric test (ie t-tes, ANOVA, regression) are not met. Permutation tests resample from the data, ignoring the explanatory variables, to generate a null-distribution of the response variable assuming explanatory variables do not matter. The test compares the data to this null-distribution generated by permutation to see if the data fall within the center of the distribution (high p-value) or in the extremes (small p-value).

Use: adonis(y~x, data=data name) in the vegan package

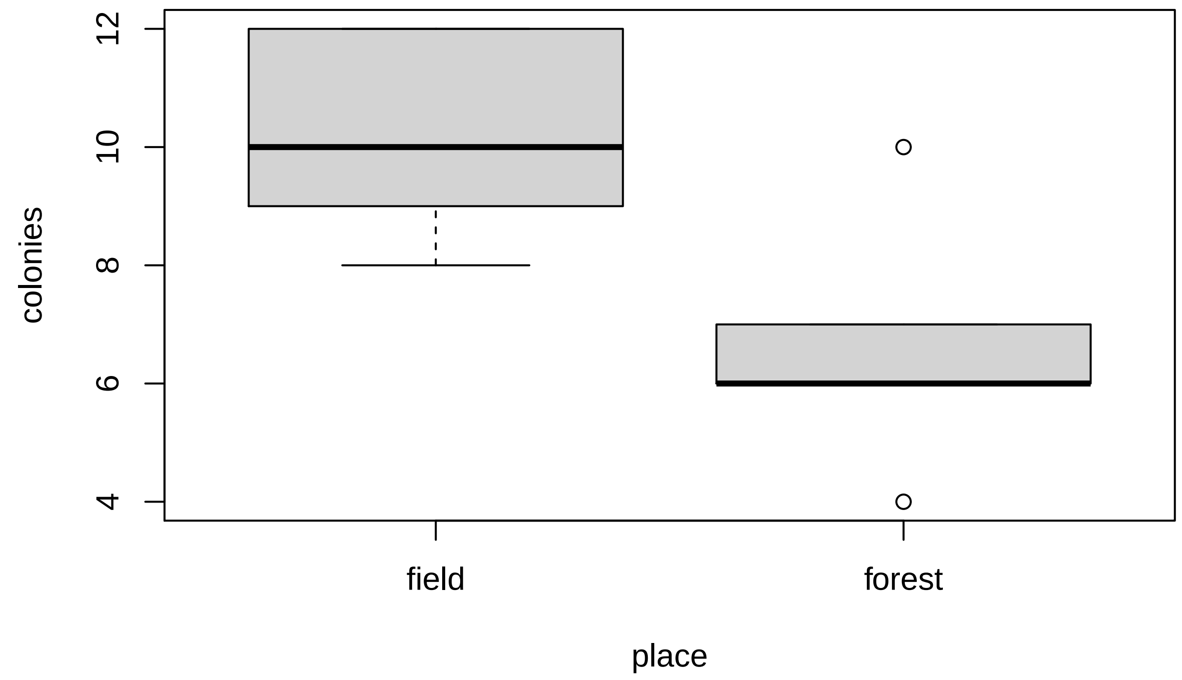
oneway_test(y~x,data=data name) in the coin package

The coin package lets you calculate asymptotic p-values (like nonparametric tests) or approximate p-values (taking many random samples) or exact p-values (from all possible combinations).

Example: compare number of ant colonies in a field versus a forest

Example origin and more code here: https://mac-theobio.github.io/QMEE/permutation_examples.html

```
#the data:
ants<-data.frame(place=rep(c("field", "forest"), rep(5,2)), colonies=c(12,9,12,10,8,6,4,6,7,10))

#plot the data:
boxplot(colonies~place, data=ants)
```



```r
#run a permutation test:
#install.packages("coin")
library(coin)
```

```
## Loading required package: survival

##
## Attaching package: 'survival'

## The following object is masked from 'package:DAAG':
##
##     lung
```

```r
#your categorical variable needs to be a factor:
ants$place<-as.factor(ants$place)

#Asymptotic (the default - similar to rank-based non-parametric tests)
oneway_test(colonies~place,data=ants)
```

```
##
##  Asymptotic Two-Sample Fisher-Pitman Permutation Test
##
## data:  colonies by place (field, forest)
## Z = 2.1279, p-value = 0.03335
## alternative hypothesis: true mu is not equal to 0
```

```
#Exact (all possible permutations):
oneway_test(colonies~place,data=ants,distribution="exact")
```

```
##
##  Exact Two-Sample Fisher-Pitman Permutation Test
##
## data:  colonies by place (field, forest)
## Z = 2.1279, p-value = 0.03968
## alternative hypothesis: true mu is not equal to 0
```

```
#Approximate (random-sampling):
oneway_test(colonies~place,data=ants,distribution=approximate(nresample=9999))
```

```
##
##  Approximative Two-Sample Fisher-Pitman Permutation Test
##
## data:  colonies by place (field, forest)
## Z = 2.1279, p-value = 0.0409
## alternative hypothesis: true mu is not equal to 0
```

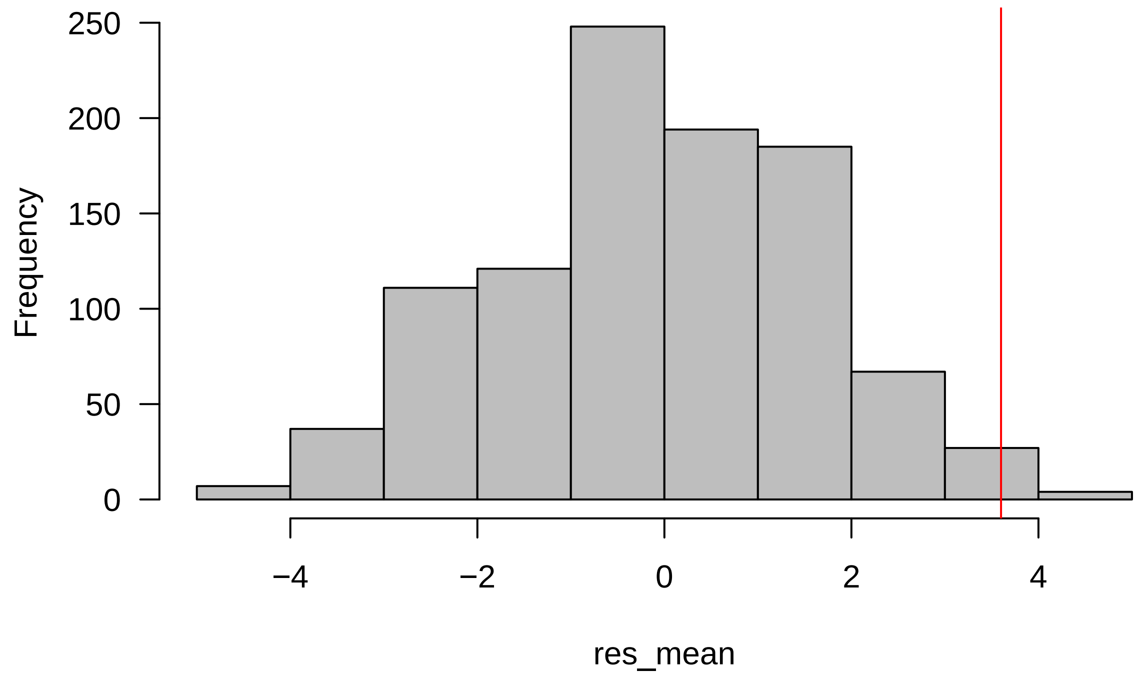
Permutation test - manual approach

You can make your own permutation test, altering the test statistic calculated in each permutation using for loops.

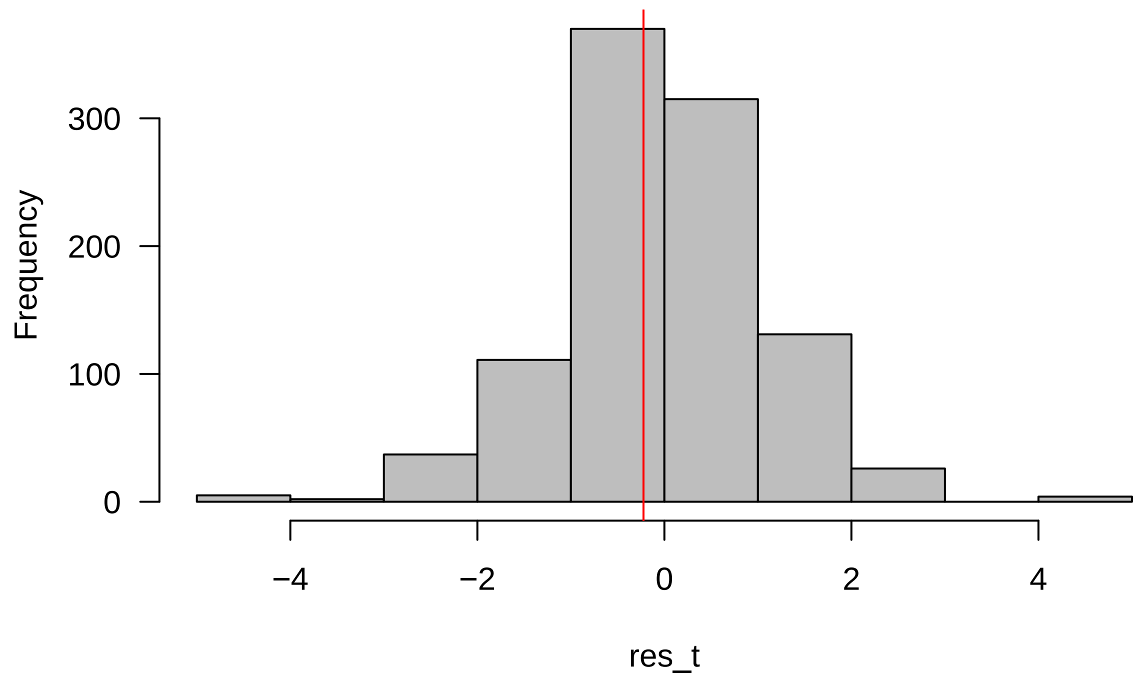
For each permutation of the data, you will calculate some test statistics (difference in means between groups, a t-statistic, an F-statistic). At the end, you compile a distribution of these test statistics calculated from these permutations - a permutation-generated null distribution for your test statistic. Compare the test statistic of your actual data to this null distribution representing the permutations, and see if it falls in the extreme 5% (the tails). For a permutation test to replace an ANOVA, a F-statistic may be a good choice as a test statistic to calculate each permutation.

For more: https://mac-theobio.github.io/QMEE/permutation_examples.html

```
set.seed(66) ##ensures random processes "start" at the same place in the computing, so each time you ru
nsim <- 1000 #number of simulations/permutations
res_mean <- numeric(nsim) ## make a numeric vector (ie stores numbers) to save the differences in means
res_t <- numeric(nsim) ## make a numeric vector to save the t-statistics in each permutation
#this for loop will run the code in the {} nsim times.
for (i in 1:nsim) {
    ## standard approach: scramble response value
    perm <- sample(nrow(ants)) #sample all rows of ants in random order
    bdat <- transform(ants,colonies=colonies[perm]) #combine these randomly sampled rows with the expla

## compute & store test statistic (eg. difference in means, t-statistic); store the value
# change this code for test statistic of your liking:
  #NOTE I give two examples (difference in means and t-statistic, but you need only one)
   #difference in means:
     agg <- aggregate(colonies~place,FUN=mean,data=bdat) #compute means by group (field vs forest)
  res_mean[i] <- agg$colonies[1]-agg$colonies[2] #save the difference in means for this permutation of
  #t-statistic (difference in means divided by standard error):
    tt <- t.test(colonies~place,data=bdat,var.equal=TRUE) #calculate t-statistic
    res_t[i] <- tt$statistic #save this permutation's t-statistic in the numeric vector

}
obs_mean <- mean(ants[ants$place=="field","colonies"])-
    mean(ants[ants$place=="forest","colonies"]) #calculate your observed difference in means (from your
obs_t <- t.test(colonies~place,data=bdat,var.equal=TRUE)$statistic #calculate t-statistic for the obser

## append the observed value to the list of results
res_mean <- c(res_mean,obs_mean) #your observed data count as a permutation, so you add it to the permu
res_t <- c(res_t,obs_t) #add observed t-statistic to permutation-generated t-statistics

#histogram representing your permutation-generated null distribution:
hist(res_mean,col="gray",las=1,main="")
abline(v=obs_mean,col="red") #red line for where the observed data fall (is it in the center or extreme
```



```
#histogram using t-statistic as test statistic:
hist(res_t,col="gray",las=1,main="")
abline(v=obs_t,col="red")
```



```
#Results of the permutation test:
mean(abs(res_mean)>=abs(obs_mean)) #p-value
```

```
## [1] 0.03396603
```

```
mean(abs(res_t)>=abs(obs_t)) #p-value
```

```
## [1] 0.9170829
```

Multivariate statistics

For when you have multiple response variables, and what to visual and investigate how they co-vary (especially used in community ecology with presence/absence or abundance of species as multiple responses across many sites).

Data should be set up as a "species by site" matrix - species (or other response variable) as columns and sites (or other sampling unit) as rows - matrix is filled in by values of each response (eg. species abundance) for each sampling unit (eg site).

The package "vegan" will handle most analyses.

The package "factoextra" helps make visualizations of ordination results.

For information on ordination, check out "Numerical Ecology with R" by Daniel Borcard, Francois Gillet, and Pierre Legendre.

This code loads the package and data for all the multivariate statistics examples.

```r
#install.packages("vegan")
library(vegan)
```

```
## Loading required package: permute

## This is vegan 2.5-6
```

```r
#install.packages("factoextra")
library(factoextra)
```

```
## Welcome! Want to learn more? See two factoextra-related books at https://goo.gl/ve3WBa
```

```r
#load in example "dune" data set:
data(dune) #data on abundance of 30 species at 20 sites
head(dune)
```

```
##   Achimill Agrostol Airaprae Alopgeni Anthodor Bellpere Bromhord Chenalbu
## 1        1        0        0        0        0        0        0        0
## 2        3        0        0        2        0        3        4        0
## 3        0        4        0        7        0        2        0        0
## 4        0        8        0        2        0        2        3        0
## 5        2        0        0        0        4        2        2        0
## 6        2        0        0        0        3        0        0        0
##   Cirsarve Comapalu Eleopalu Elymrepe Empenigr Hyporadi Juncarti Juncbufo
## 1        0        0        0        4        0        0        0        0
## 2        0        0        0        4        0        0        0        0
## 3        0        0        0        4        0        0        0        0
## 4        2        0        0        4        0        0        0        0
## 5        0        0        0        4        0        0        0        0
## 6        0        0        0        0        0        0        0        0
##   Lolipere Planlanc Poaprat Poatriv Ranuflam Rumeacet Sagiproc Salirepe
## 1        7        0       4       2        0        0        0        0
## 2        5        0       4       7        0        0        0        0
## 3        6        0       5       6        0        0        0        0
## 4        5        0       4       5        0        0        5        0
## 5        2        5       2       6        0        5        0        0
## 6        6        5       3       4        0        6        0        0
##   Scorautu Trifprat Trifrepe Vicilath Bracruta Callcusp
## 1        0        0        0        0        0        0
## 2        5        0        5        0        0        0
## 3        2        0        2        0        2        0
## 4        2        0        1        0        2        0
## 5        3        2        2        0        2        0
## 6        3        5        5        0        6        0
```

```r
data(dune.env)#environmental data on those 20 sites
head(dune.env)
```

```
##    A1 Moisture Management      Use Manure
## 1 2.8        1         SF Haypastu      4
## 2 3.5        1         BF Haypastu      2
## 3 4.3        2         SF Haypastu      4
## 4 4.2        2         SF Haypastu      4
## 5 6.3        1         HF Hayfield      2
## 6 4.3        1         HF Haypastu      2
```

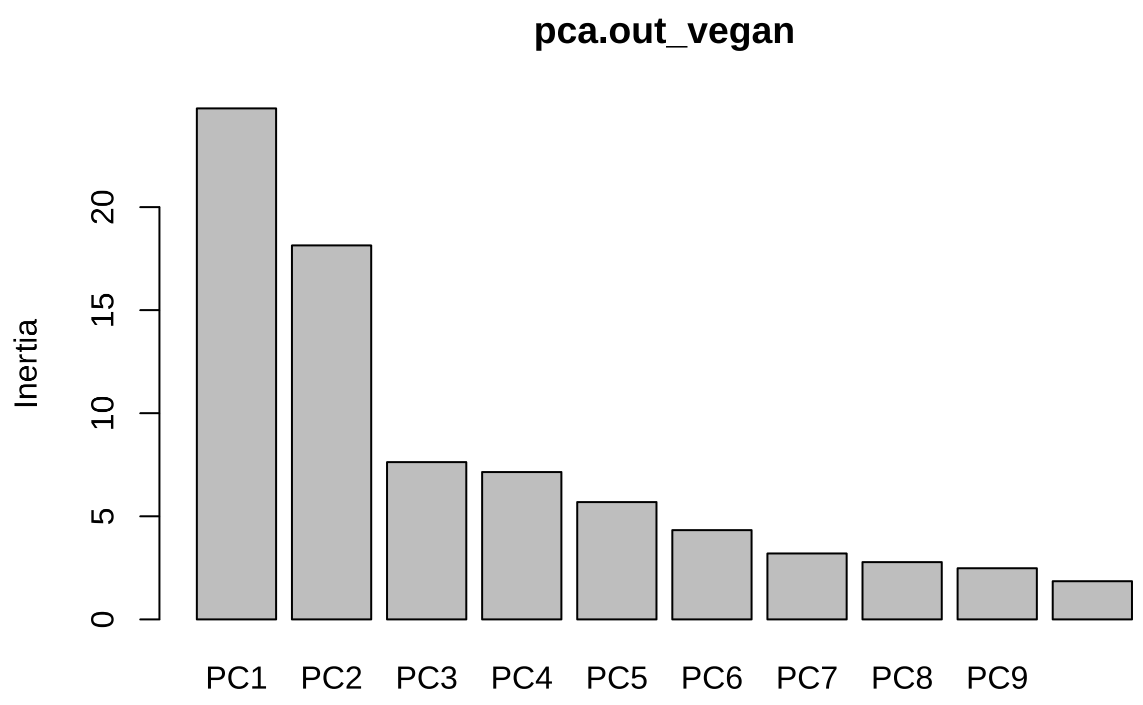
Principal Component Analysis (PCA)

The function prcomp() runs a PCA in base R. The function rda() runs “redundancy analysis” (RDA) - which is a PCA constrained by explanatory variables. Use “~1” to indicate no explanatory variables and run just a PCA. You can include explanatory variables to run an RDA.

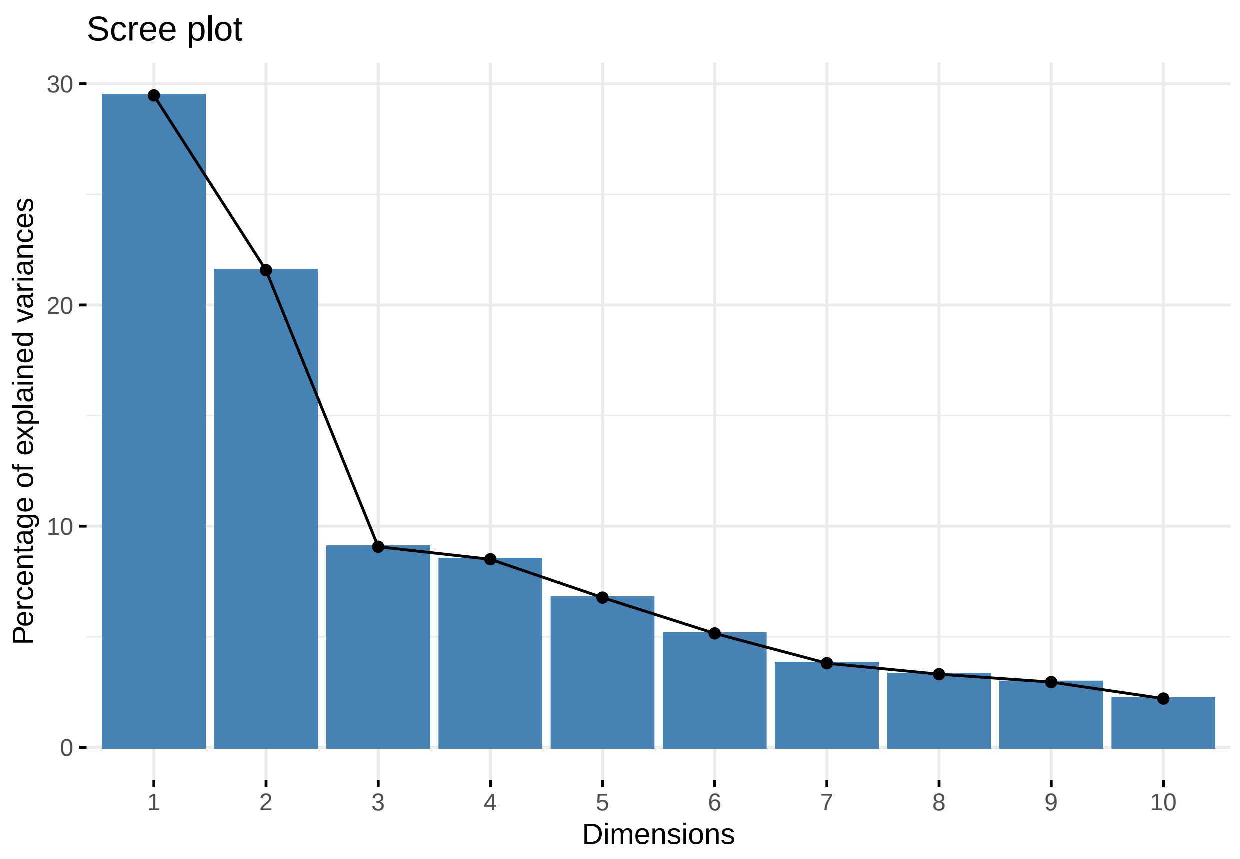
The FactoMineR package is also useful. Check out this video (and related videos) by the package author: https://www.youtube.com/watch?v=pks8m2ka7Pk.

Other plotting options: https://cran.r-project.org/web/packages/ggfortify/vignettes/plot_pca.html

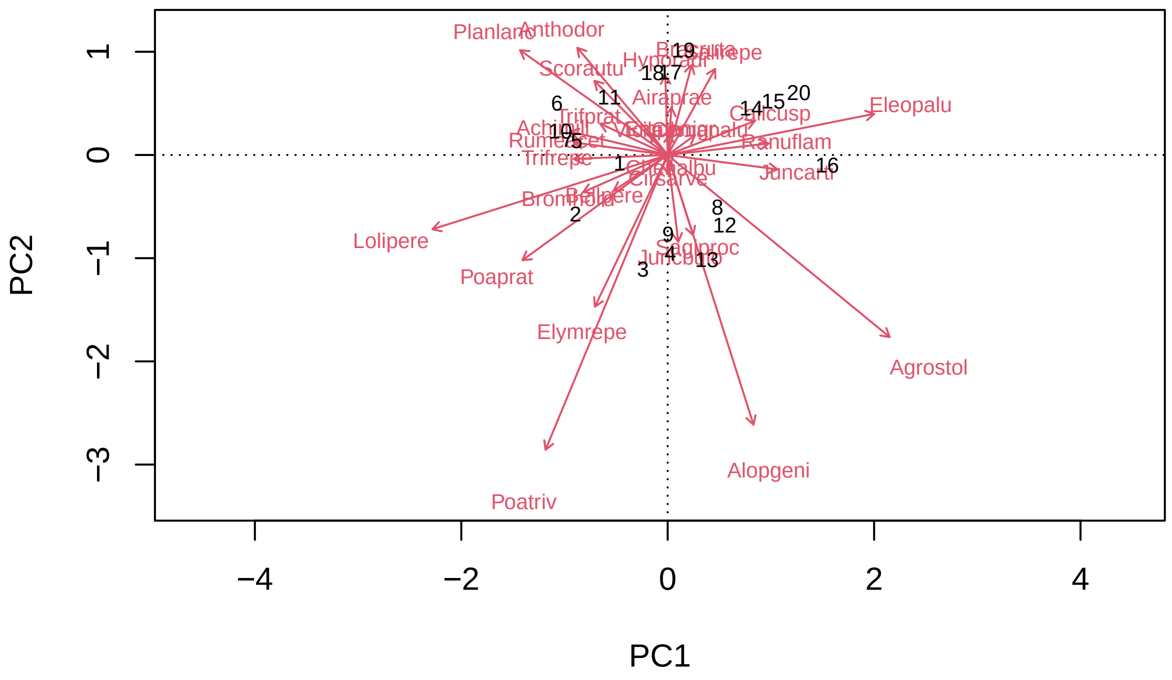
```
#Two functions to run a PCA:

pca.out_base<-prcomp(dune)
pca.out_vegan<-rda(dune~1)

#Scree plots (percentage of variation explained by each principal component axis):
screeplot(pca.out_vegan) #drop-off after second axis, suggesting only first two axes need examining
```



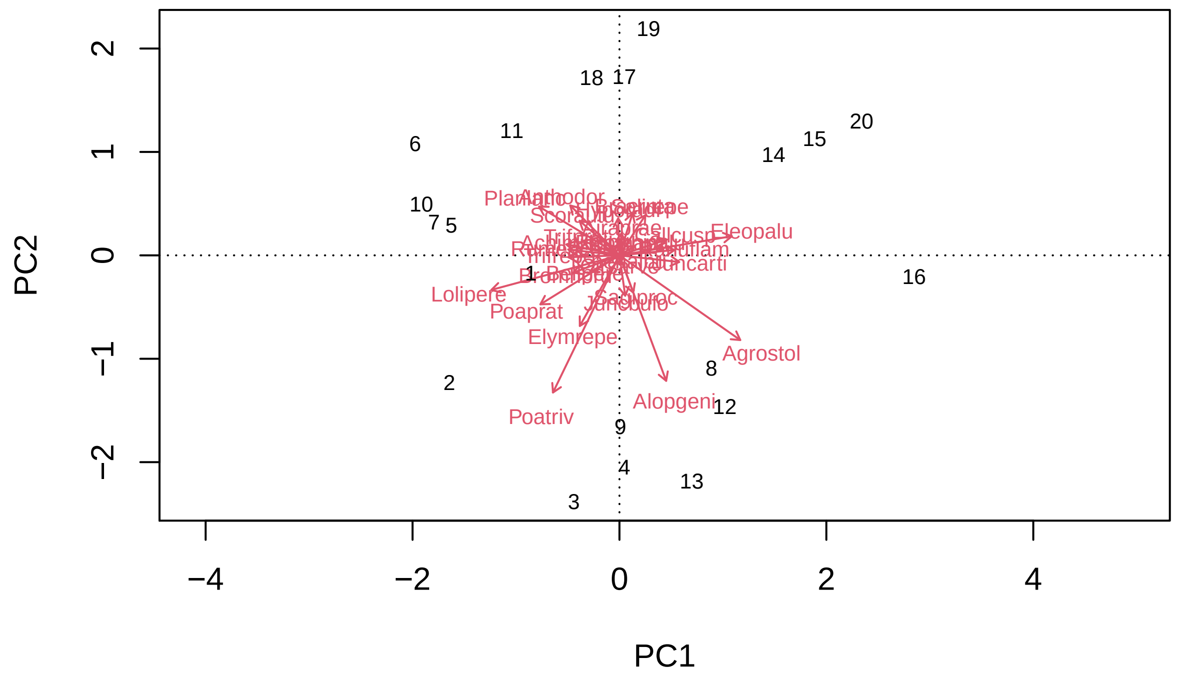
```r
fviz_eig(pca.out_base)
```



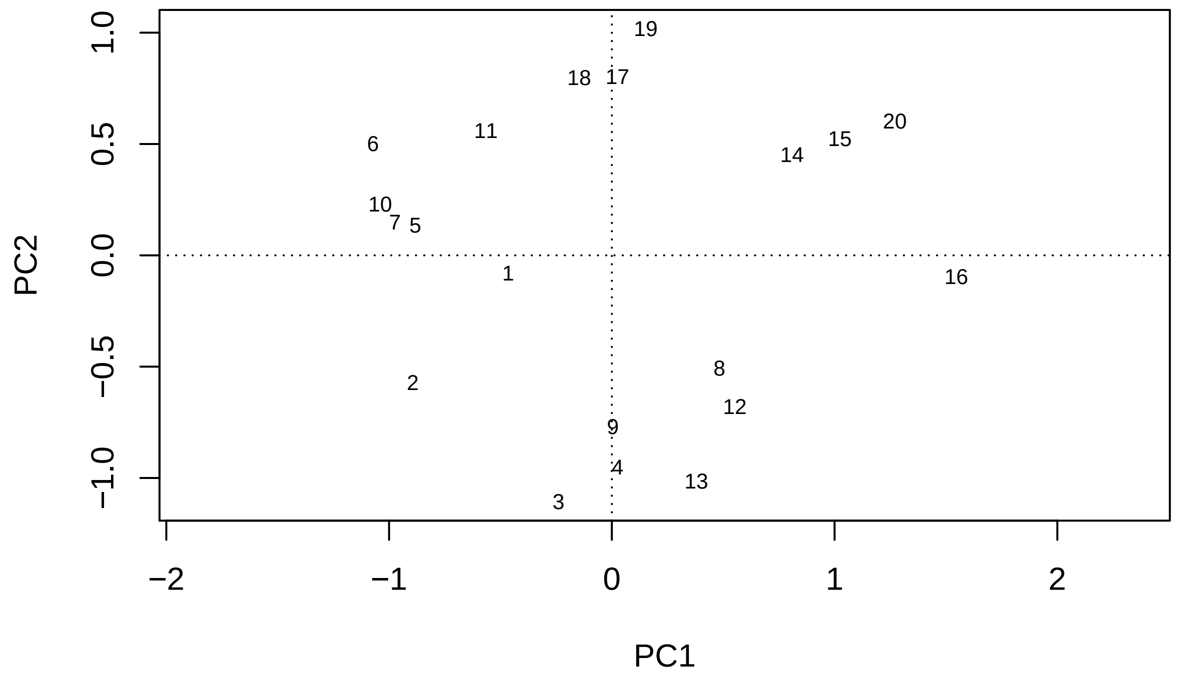
```r
#Examine biplots:
#scaling=1 (for individuals/sites/samples):
biplot(pca.out_vegan, scaling=1)
```



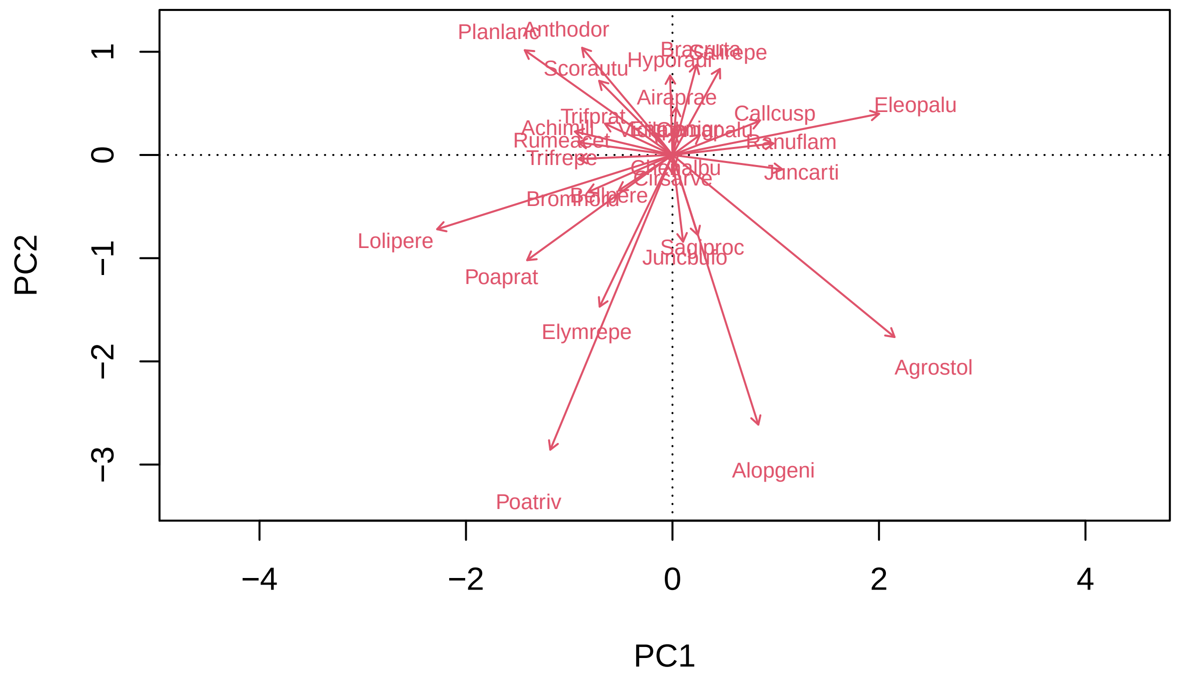
```r
#scaling=2 (for species/response):
biplot(pca.out_vegan, scaling=2)
```



```
#only show the individuals/sites/samples:
biplot(pca.out_vegan, scaling=1, display="sites")
```



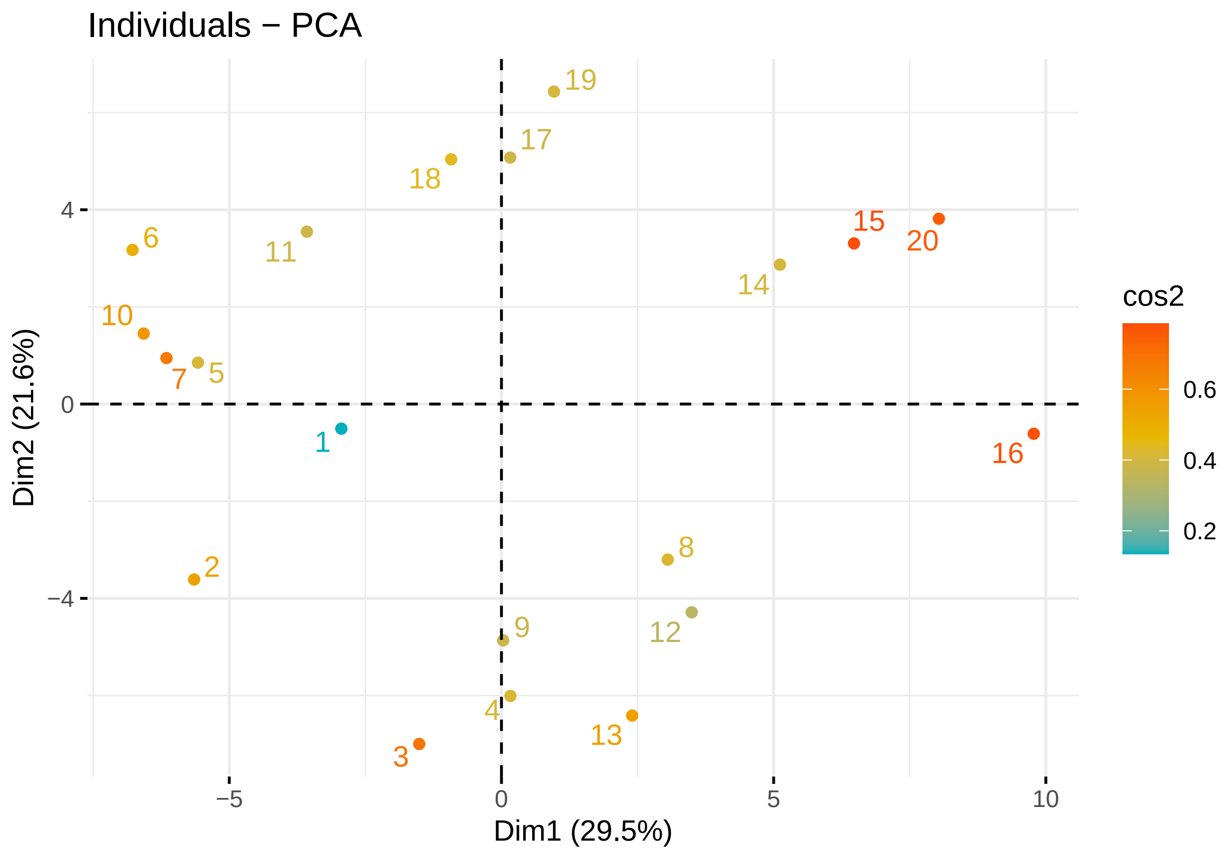
```
#only show the species/response:
biplot(pca.out_vegan, scaling=1, display="species")
```



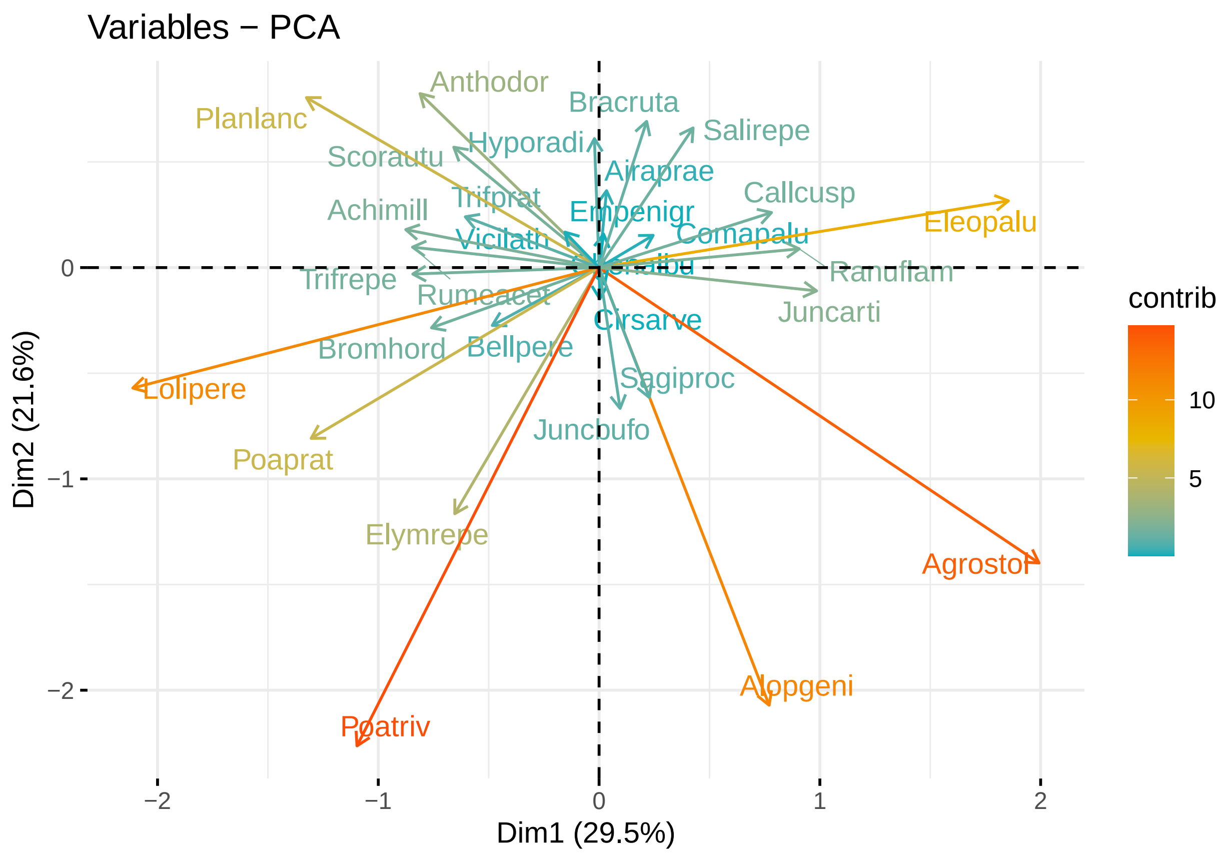
```r
#Summary of results - proportion of variance explained, species scores, PC values for each site.
summary(pca.out_vegan)
```

```
##
## Call:
## rda(formula = dune ~ 1)
##
## Partitioning of variance:
##               Inertia Proportion
## Total           84.12          1
## Unconstrained   84.12          1
##
## Eigenvalues, and their contribution to the variance
##
## Importance of components:
##                          PC1     PC2     PC3     PC4    PC5     PC6     PC7
## Eigenvalue            24.7953 18.1466 7.62913 7.15277 5.6950 4.33331 3.19936
## Proportion Explained   0.2947  0.2157 0.09069 0.08503 0.0677 0.05151 0.03803
## Cumulative Proportion  0.2947  0.5105 0.60115 0.68618 0.7539 0.80539 0.84342
##                          PC8    PC9    PC10    PC11    PC12    PC13     PC14
## Eigenvalue            2.78186 2.4820 1.85377 1.74712 1.31358 0.99051 0.637794
## Proportion Explained  0.03307 0.0295 0.02204 0.02077 0.01561 0.01177 0.007582
## Cumulative Proportion 0.87649 0.9060 0.92803 0.94880 0.96441 0.97619 0.983768
##                           PC15     PC16     PC17     PC18     PC19
## Eigenvalue            0.550827 0.350584 0.199556 0.148798 0.115753
## Proportion Explained  0.006548 0.004167 0.002372 0.001769 0.001376
## Cumulative Proportion 0.990316 0.994483 0.996855 0.998624 1.000000
##
## Scaling 2 for species and site scores
## * Species are scaled proportional to eigenvalues
## * Sites are unscaled: weighted dispersion equal on all dimensions
## * General scaling constant of scores:  6.322924
##
##
## Species scores
##
##                PC1      PC2       PC3       PC4      PC5       PC6
## Achimill -0.603786  0.12392  0.008464  0.159574  0.40871  0.127857
## Agrostol  1.373953 -0.96401  0.166905  0.266466 -0.08765  0.047368
## Airaprae  0.023415  0.25078 -0.194768 -0.326043  0.05574 -0.079619
## Alopgeni  0.531234 -1.42784 -0.505241 -0.042885 -0.44293  0.278566
## Anthodor -0.559138  0.56761 -0.476205  0.015781  0.34408 -0.135783
## Bellpere -0.333560 -0.18881  0.140638 -0.084177  0.12541  0.134771
## Bromhord -0.523468 -0.19656  0.164222  0.005671  0.38612  0.257634
## Chenalbu  0.017494 -0.05462 -0.055349 -0.010582  0.02664  0.016405
## Cirsarve  0.002398 -0.10237  0.063716 -0.048735 -0.03212 -0.036055
## Comapalu  0.168933  0.10522  0.063625  0.052352  0.13056  0.108129
## Eleopalu  1.278257  0.21782  0.469213  0.667986  0.20877  0.189927
## Elymrepe -0.450692 -0.80310  0.340783 -0.243514  0.25145 -0.691715
## Empenigr  0.014054  0.10956 -0.099378 -0.161788 -0.02289 -0.001195
## Hyporadi -0.014612  0.42079 -0.223096 -0.535685 -0.10309 -0.025185
## Juncarti  0.679423 -0.07604  0.243642  0.310903 -0.08877 -0.248736
## Juncbufo  0.065583 -0.45959 -0.548944 -0.018900 -0.10571 -0.087168
## Lolipere -1.455985 -0.39306  1.013109  0.170493 -0.52430  0.114397
## Planlanc -0.913938  0.55455 -0.244341  0.617631 -0.12494 -0.098047
## Poaprat  -0.899147 -0.55712  0.542805 -0.042467 -0.27815 -0.026353
## Poatriv  -0.756003 -1.56056 -0.480385  0.351099  0.36641  0.044066
## Ranuflam  0.625121  0.06099  0.124760  0.233953  0.13645  0.087328
## Rumeacet -0.582581  0.06663 -0.574256  0.775879 -0.08772 -0.361433
## Sagiproc  0.156823 -0.42388 -0.331722 -0.454322 -0.43262  0.037181
## Salirepe  0.293607  0.45555 -0.023780 -0.196209 -0.20176 -0.097569
## Scorautu -0.453771  0.39268 -0.212281 -0.382424 -0.27635  0.395164
## Trifprat -0.417853  0.16572 -0.234524  0.570030 -0.09646 -0.128045
## Trifrepe -0.581801 -0.02115 -0.167299  0.196535  0.18714  0.928758
## Vicilath -0.106710  0.11571  0.092827 -0.055592 -0.15433  0.129733
## Bracruta  0.148626  0.47690 -0.168758  0.509177 -0.96307  0.029481
## Callcusp  0.538513  0.17963  0.175086  0.238876  0.25531  0.169209
##
##
## Site scores (weighted sums of species scores)
##
##         PC1     PC2     PC3     PC4      PC5      PC6
## 1  -0.85678 -0.1724  2.6079 -1.1296  0.45074 -2.49113
## 2  -1.64477 -1.2299  0.8867 -0.9859  2.03463  1.81057
## 3  -0.44010 -2.3827  0.9297 -0.4601 -1.02783 -0.05183
## 4   0.04795 -2.0463  1.2737 -0.9742 -0.64210 -0.72074
## 5  -1.62445  0.2900 -1.5927  1.5398  1.86008 -2.21191
## 6  -1.97427  1.0802 -1.1501  3.3534 -1.52026  0.03127
## 7  -1.79263  0.3220 -0.2200  1.4714  0.01245 -0.42583
## 8   0.88980 -1.0905  0.9250  0.5165 -1.08897  0.94777
## 9   0.00904 -1.6570 -0.4661 -0.2826 -0.10821 -2.16570
## 10 -1.91463  0.4940  0.7058  0.2676  1.36985  2.62386
## 11 -1.04110  1.2081  1.4203 -0.9566 -2.71745  1.11200
## 12  1.01822 -1.4598 -3.2509 -0.3247 -1.75331  1.01550
## 13  0.69939 -2.1837 -2.2128 -0.4231  1.06502  0.65585
## 14  1.49047  0.9772  0.5447  0.2733  2.38875  2.47896
## 15  1.88644  1.1261  0.7271  0.7732  0.22113 -0.31750
## 16  2.84848 -0.2081  0.7041  2.1012  0.29311 -0.08124
## 17  0.04666  1.7279 -0.9135 -1.6663  1.80070 -1.55572
## 18 -0.26936  1.7157  0.1648 -0.5770 -2.10498  0.33880
## 19  0.28094  2.1901 -1.9865 -3.2341 -0.45760 -0.02389
## 20  2.34069  1.2991  0.9029  0.7178 -0.07573 -0.96909
```

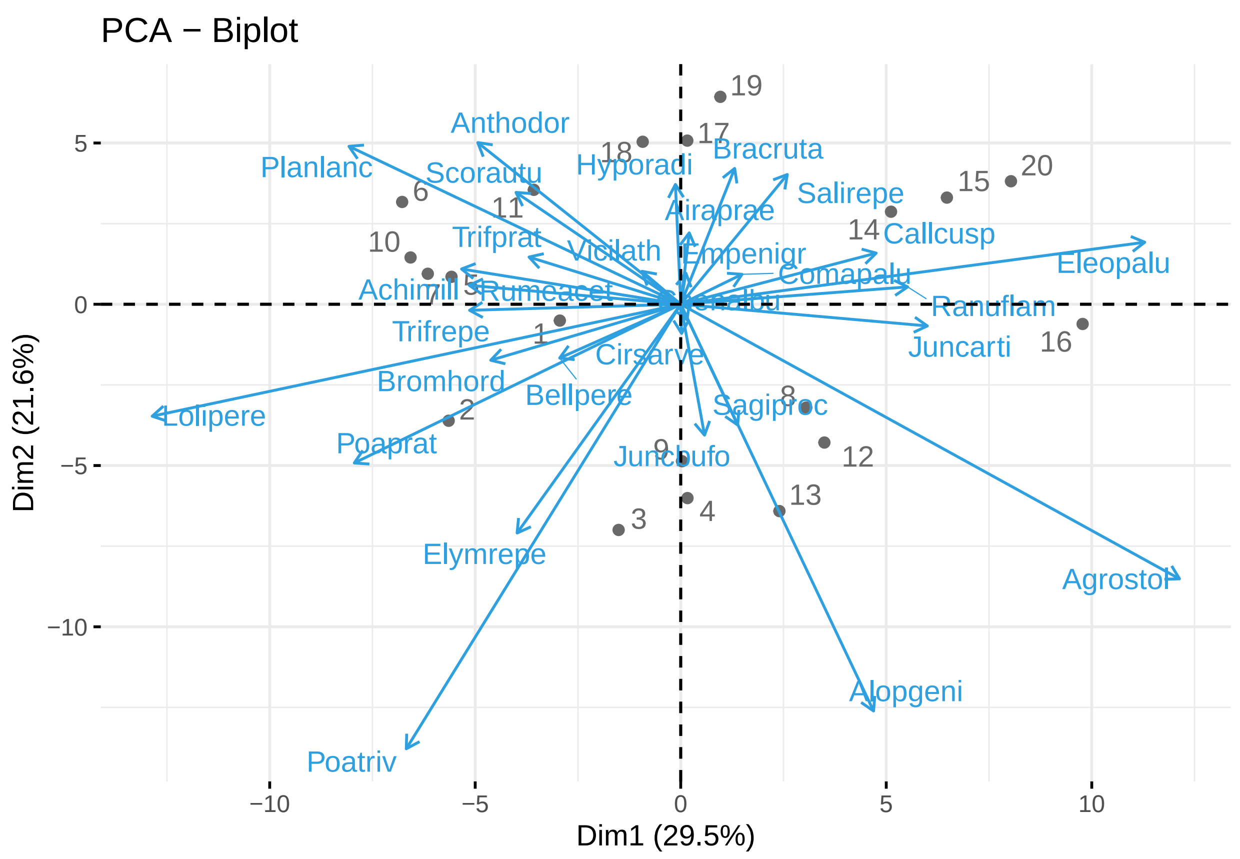
```
#####Using factoextra on prcomp results:######
#check out: http://www.sthda.com/english/articles/31-principal-component-methods-in-r-practical-guide/1

#Coloured plot of sites:
fviz_pca_ind(pca.out_base,
             col.ind = "cos2", # Color by the quality of representation (ie how well the site is repres
             gradient.cols = c("#00AFBB", "#E7B800", "#FC4E07"),
             repel = TRUE     # Avoid text overlapping
             )
```



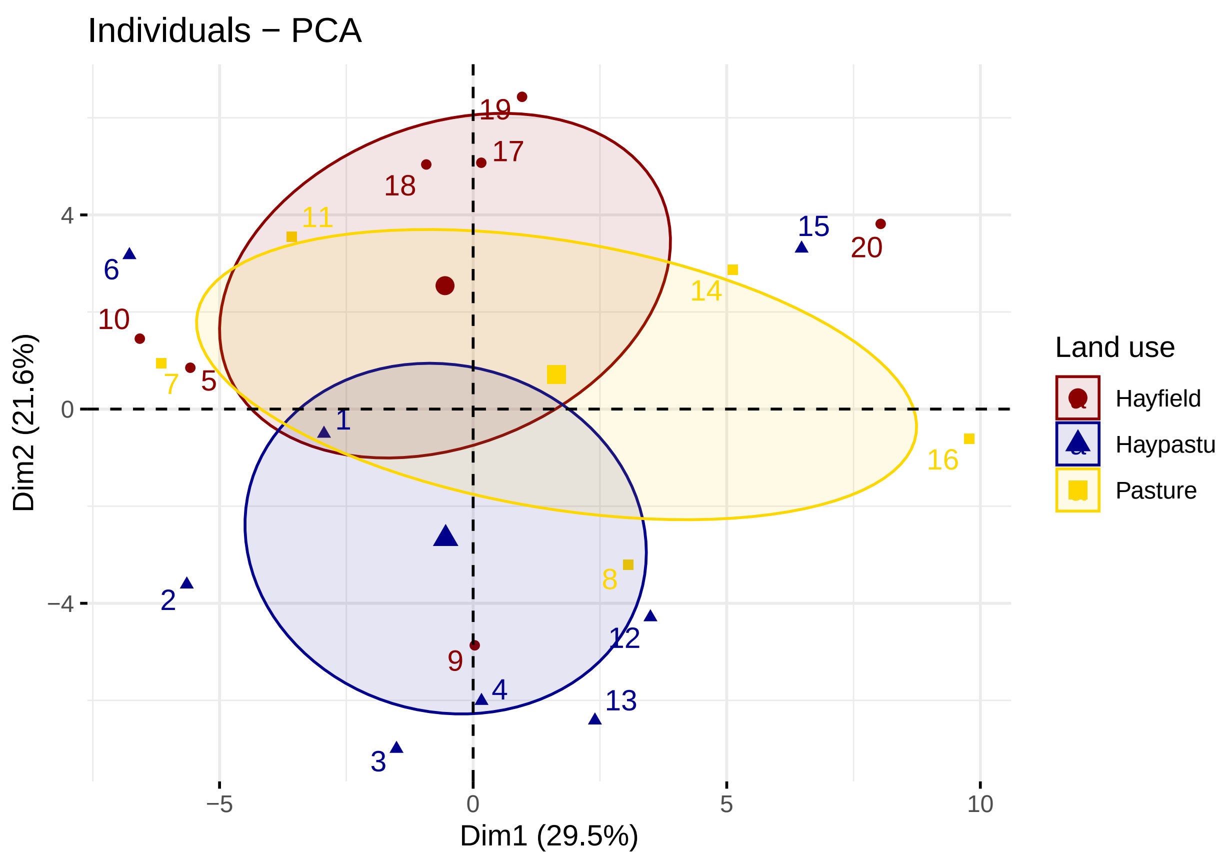
```
#Plot of contributions of each species/response to each axis for base PCA:
fviz_pca_var(pca.out_base,
             col.var = "contrib", # Color by contributions to the PC
             gradient.cols = c("#00AFBB", "#E7B800", "#FC4E07"),
             repel = TRUE     # Avoid text overlapping
             )
```



```
#Coloured biplot of sites and species/responses
fviz_pca_biplot(pca.out_base, repel = TRUE,
                col.var = "#2E9FDF", # Response variables color
                col.ind = "#696969"  # Individuals color
                )
```



```
#Visualize categorical grouping variable:
#Add ellipses representing the average and 95% confidence interval for sites by land use type (or other
fviz_pca_ind(pca.out_base,
             col.ind = as.character(dune.env$Use), # color by groups (in this case "Use")
             palette = c("darkred",  "darkblue", "gold"),
             addEllipses = TRUE, # Concentration ellipses
             ellipse.type = "confidence",
             legend.title = "Land use",
             repel = TRUE
             )
```



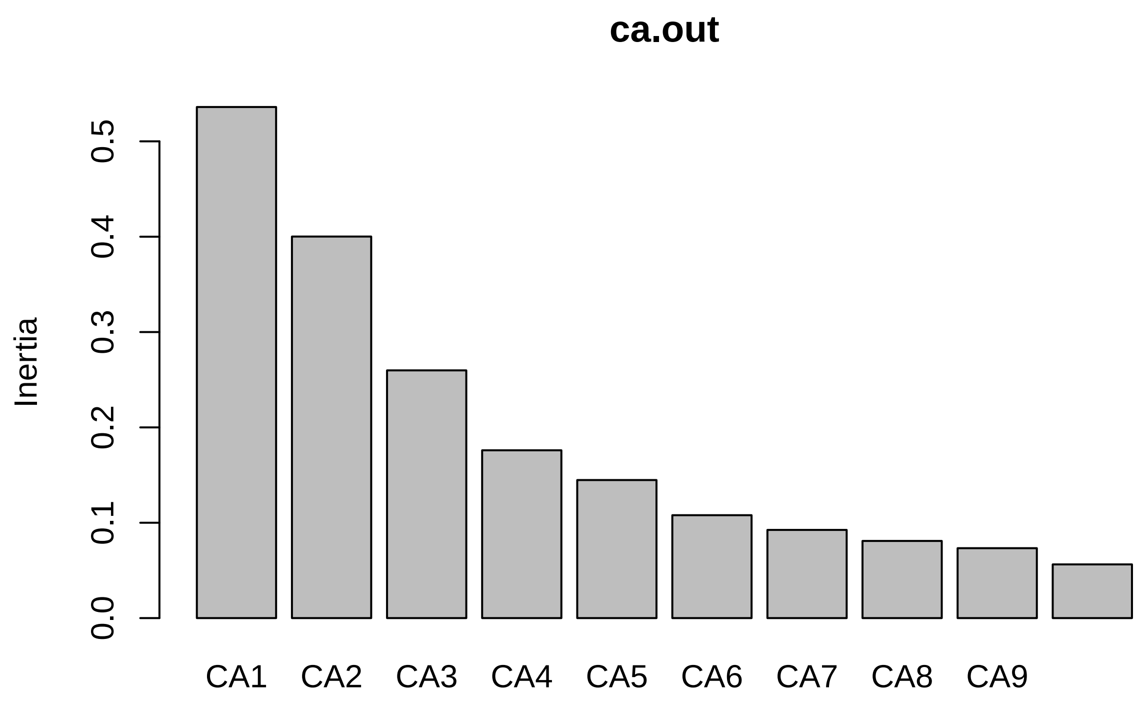
Correspondence analysis

Correspondence analysis (CA) investigates relative relationships between groups of variables based on contigency table - ie interpret your “species by site” matrix as a contigency table of frequencies. For instance, unlike PCA, CA investigates variance in relative abundance or values among species/responses rather than using the absolute abundance or values in the column. Are some species or groups of species associated with some sites?

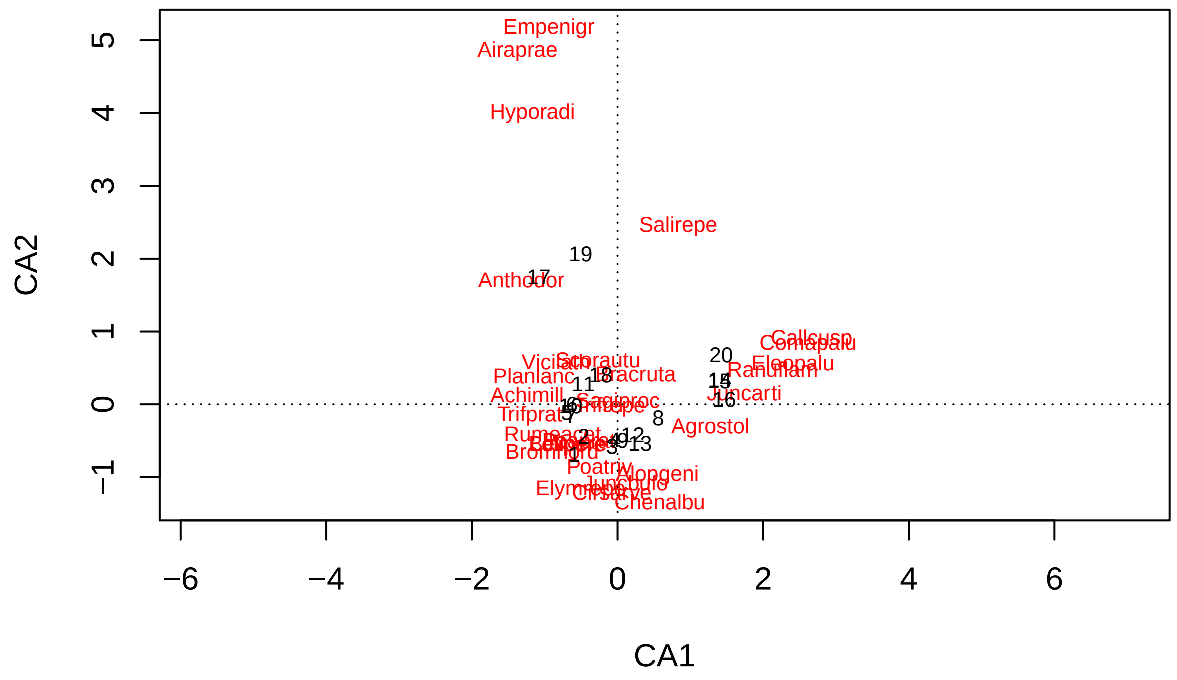
The function cca() in “vegan” runs a canonical correspondance analysis (CCA) - see below. But if you use “~1” it will run a normal CA.

Also check out ca() in the “ca” package, CA() in the FactoMineR package, and corresp() in the “MASS” package. Check out: https://www.gastonsanchez.com/visually-enforced/how-to/2012/07/19/Correspondence-Analysis/

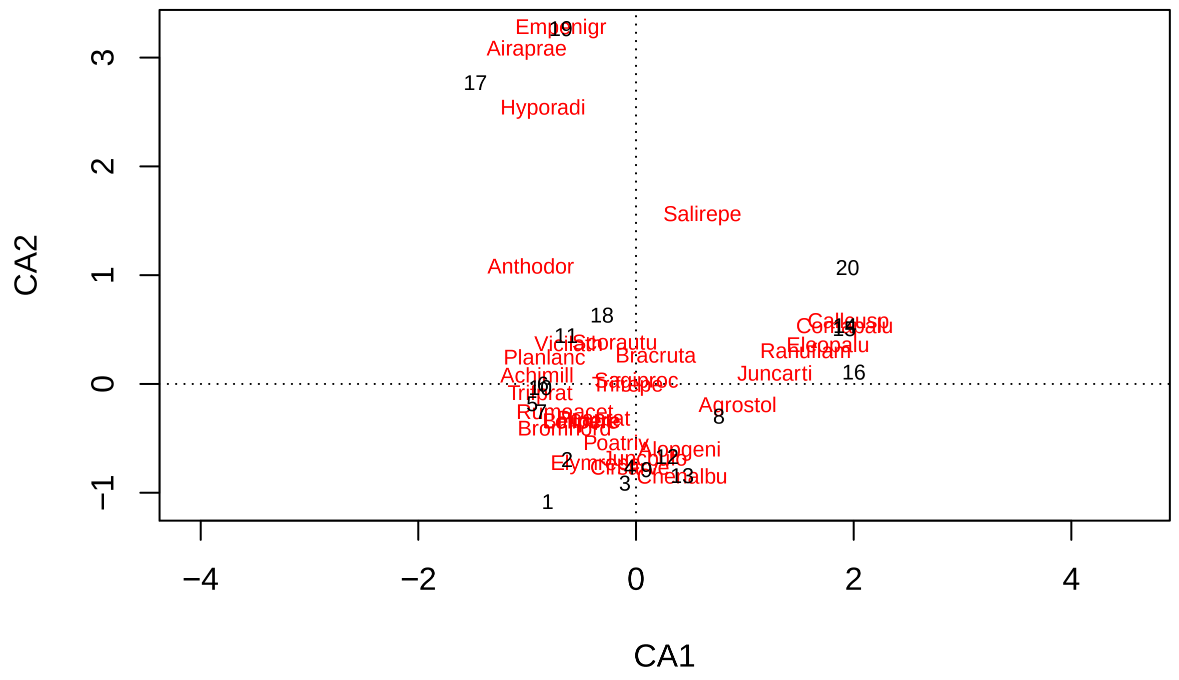
```
#Run the CA:
ca.out<-cca(dune~1)

#Scree plots (percentage of variation explained by each principal component axis):
screeplot(ca.out) #drop-off after second axis, suggesting only first two axes need examining
```



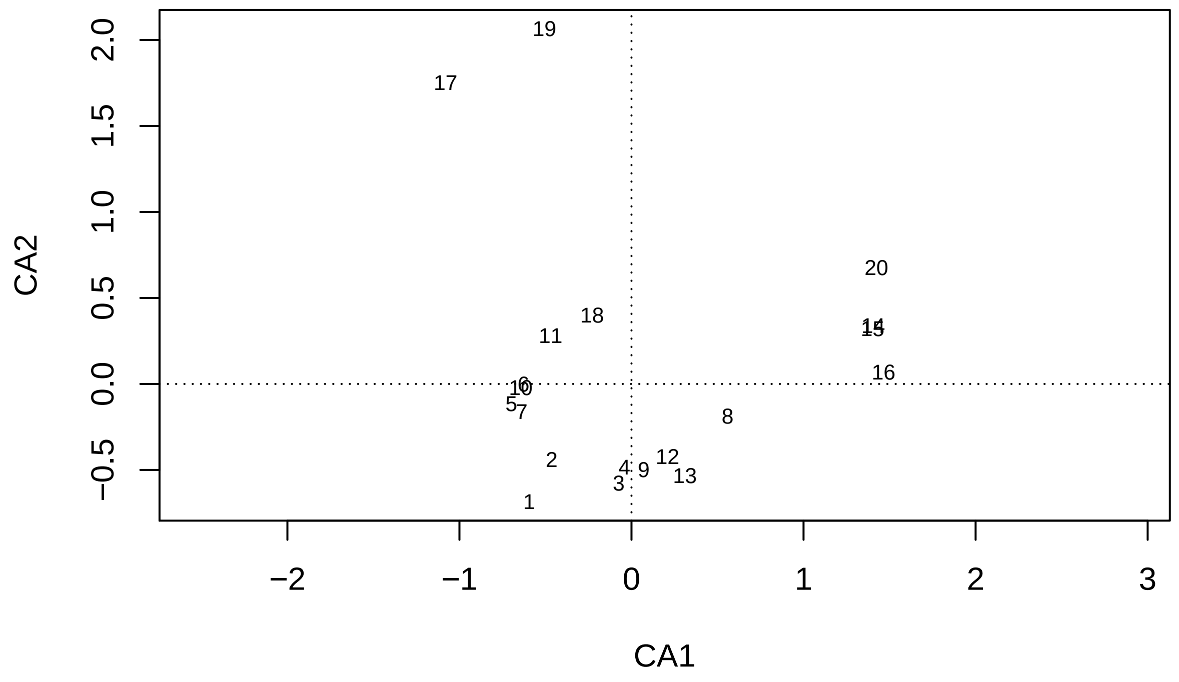
```
#Examine biplots - NOTE use plot() here:
#scaling=1 (for individuals/sites/samples):
plot(ca.out, scaling=1)
```



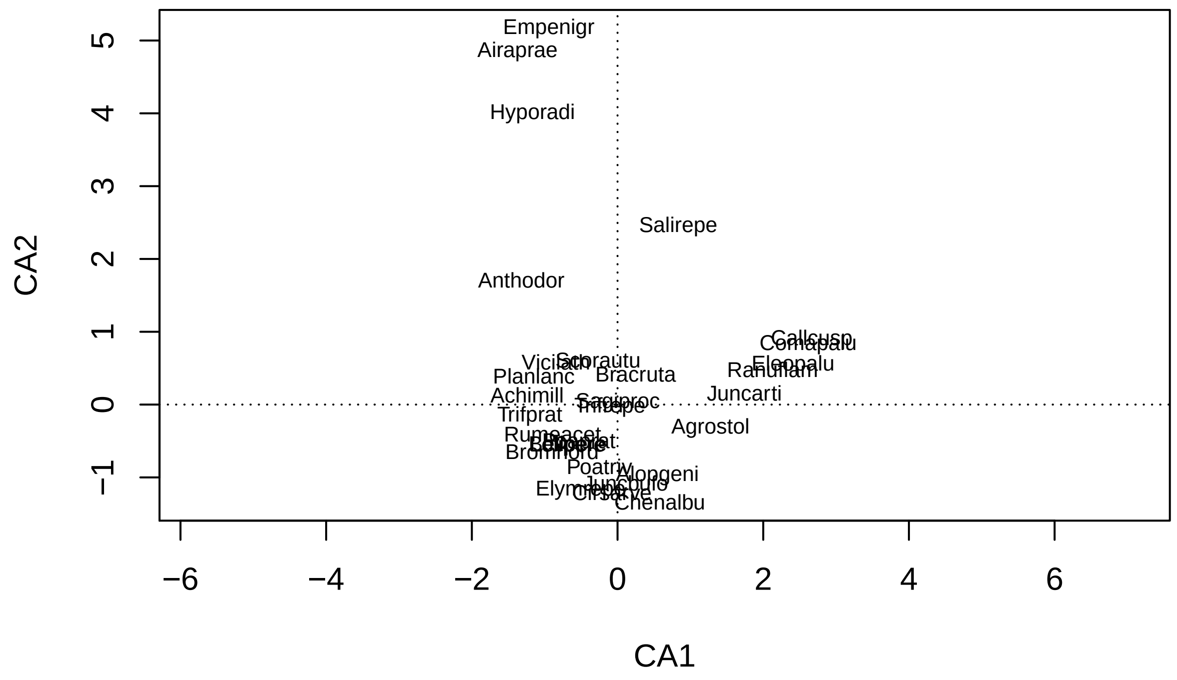
```r
#scaling=2 (for species/response):
plot(ca.out, scaling=2)
```



```r
#only show the individuals/sites/samples:
plot(ca.out, scaling=1, display="sites")
```



```
#only show the species/response:
plot(ca.out, scaling=1, display="species")
```



```r
#Summary of results - proportion of variance explained, species scores, PC values for each site.
summary(ca.out)
```

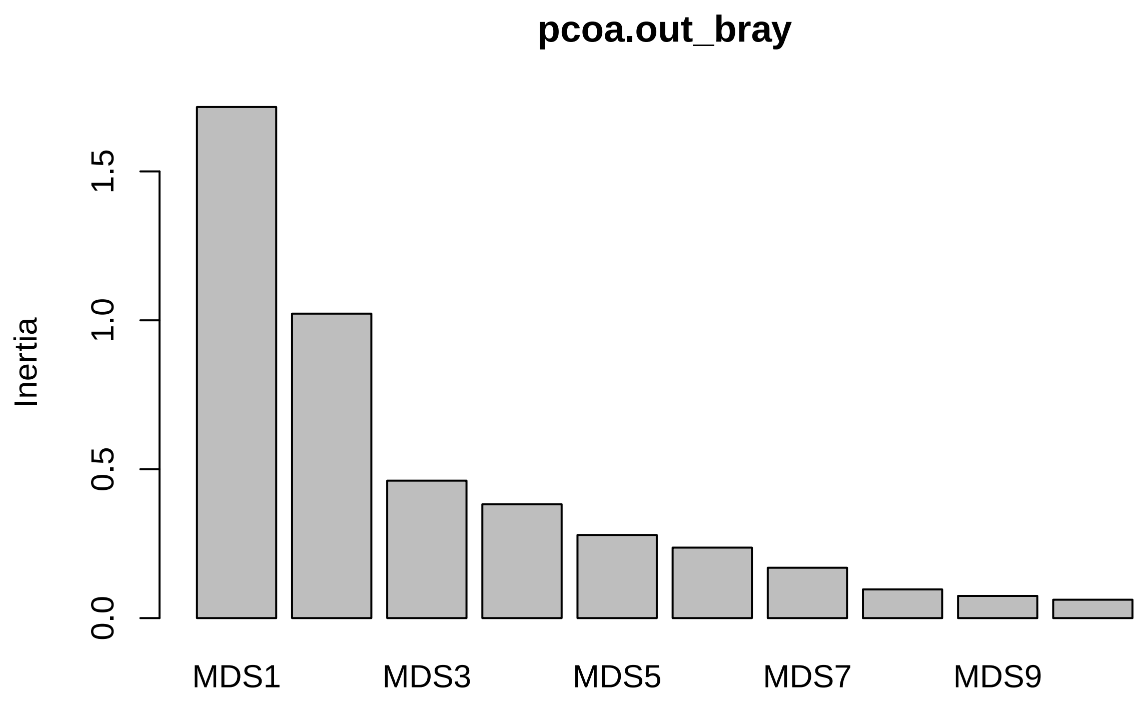
```
##
## Call:
## cca(formula = dune ~ 1)
##
## Partitioning of scaled Chi-square:
##               Inertia Proportion
## Total           2.115          1
## Unconstrained   2.115          1
##
## Eigenvalues, and their contribution to the scaled Chi-square
##
## Importance of components:
##                          CA1    CA2    CA3     CA4     CA5     CA6     CA7
## Eigenvalue            0.5360 0.4001 0.2598 0.17598 0.14476 0.10791 0.09247
## Proportion Explained  0.2534 0.1892 0.1228 0.08319 0.06844 0.05102 0.04372
## Cumulative Proportion 0.2534 0.4426 0.5654 0.64858 0.71702 0.76804 0.81175
##                           CA8     CA9    CA10    CA11    CA12    CA13     CA14
## Eigenvalue            0.08091 0.07332 0.05630 0.04826 0.04125 0.03523 0.020529
## Proportion Explained  0.03825 0.03466 0.02661 0.02282 0.01950 0.01665 0.009705
## Cumulative Proportion 0.85000 0.88467 0.91128 0.93410 0.95360 0.97025 0.979955
##                           CA15     CA16     CA17     CA18     CA19
## Eigenvalue            0.014911 0.009074 0.007938 0.007002 0.003477
## Proportion Explained  0.007049 0.004290 0.003753 0.003310 0.001644
## Cumulative Proportion 0.987004 0.991293 0.995046 0.998356 1.000000
##
## Scaling 2 for species and site scores
## * Species are scaled proportional to eigenvalues
## * Sites are unscaled: weighted dispersion equal on all dimensions
##
##
## Species scores
##
##               CA1      CA2      CA3       CA4       CA5      CA6
## Achimill -0.90859  0.08461 -0.58636 -0.008919 -0.660183 -0.18877
## Agrostol  0.93378 -0.20651  0.28165  0.024293 -0.139326 -0.02256
## Airaprae -1.00434  3.06749  1.33773  0.194305 -1.081813 -0.53699
## Alopgeni  0.40088 -0.61839  0.85013  0.346740  0.016509  0.10169
## Anthodor -0.96676  1.08361 -0.17188  0.459788 -0.607533 -0.30425
## Bellpere -0.50018 -0.35503 -0.15239 -0.704153 -0.058546  0.07308
## Bromhord -0.65762 -0.40634 -0.30685 -0.496751 -0.561358  0.07004
## Chenalbu  0.42445 -0.84402  1.59029  1.248755 -0.207480  0.87566
## Cirsarve -0.05647 -0.76398  0.91793 -1.175919 -0.384024 -0.13985
## Comapalu  1.91690  0.52150 -1.18215 -0.021738 -1.359988  1.31207
## Eleopalu  1.76383  0.34562 -0.57336 -0.002976 -0.332396 -0.14688
## Elymrepe -0.37074 -0.74148  0.26238 -0.566308 -0.270122 -0.72624
## Empenigr -0.69027  3.26420  1.95716 -0.176936 -0.073518 -0.16083
## Hyporadi -0.85408  2.52821  1.13951 -0.175115 -0.311874  0.11177
## Juncarti  1.27580  0.09963 -0.09320  0.005536  0.289410 -0.78247
## Juncbufo  0.08157 -0.68074  1.00545  1.078390  0.268360  0.24168
## Lolipere -0.50272 -0.35955 -0.21821 -0.474727  0.101494 -0.01594
## Planlanc -0.84058  0.24886 -0.78066  0.371149  0.271377  0.11989
## Poaprat  -0.38919 -0.32999 -0.02015 -0.358371  0.079296 -0.05165
## Poatriv  -0.18185 -0.53997  0.23388  0.178834 -0.155342 -0.07584
## Ranuflam  1.55886  0.30700 -0.29765  0.046974 -0.008747 -0.14744
## Rumeacet -0.65289 -0.25525 -0.59728  1.160164  0.255849 -0.32730
## Sagiproc  0.00364  0.01719  1.11570  0.066981  0.186654  0.32463
## Salirepe  0.61035  1.54868  0.04970 -0.607136  1.429729 -0.55183
## Scorautu -0.19566  0.38884  0.03975 -0.130392  0.141232  0.23717
## Trifprat -0.88116 -0.09792 -1.18172  1.282429  0.325706 -0.33388
## Trifrepe -0.07666 -0.02032 -0.20594  0.026462 -0.186748  0.53957
## Vicilath -0.61893  0.37140 -0.46057 -1.000375  1.162652  1.44971
## Bracruta  0.18222  0.26477 -0.16606  0.064009  0.576334  0.07741
## Callcusp  1.95199  0.56743 -0.85948 -0.098969 -0.556737  0.23282
##
##
## Site scores (weighted averages of species scores)
##
##         CA1        CA2      CA3      CA4      CA5      CA6
## 1  -0.81167 -1.0826714 -0.14479 -2.10665 -0.39287 -1.83462
## 2  -0.63268 -0.6958357 -0.09708 -1.18695 -0.97686  0.06575
## 3  -0.10148 -0.9128732  0.68815 -0.68137 -0.08709 -0.28678
## 4  -0.05647 -0.7639784  0.91793 -1.17592 -0.38402 -0.13985
## 5  -0.95293 -0.1846015 -0.95609  0.86853 -0.34552 -0.98333
## 6  -0.85633 -0.0005408 -1.39735  1.59909  0.65494 -0.19386
## 7  -0.87149 -0.2547040 -0.86830  0.90468  0.17385 -0.03446
## 8   0.76268 -0.2968459  0.35648 -0.10772  0.17507 -0.36444
## 9   0.09693 -0.7864314  0.86492  0.40090  0.28704 -1.02783
## 10 -0.87885 -0.0353136 -0.82987 -0.68053 -0.75438  0.81070
## 11 -0.64223  0.4440332 -0.17371 -1.09684  1.37462  2.00626
## 12  0.28557 -0.6656161  1.64423  1.71496  0.65381  1.17376
## 13  0.42445 -0.8440195  1.59029  1.24876 -0.20748  0.87566
## 14  1.91996  0.5351062 -1.39863 -0.08575 -2.21317  2.43044
## 15  1.91384  0.5079036 -0.96567  0.04227 -0.50681  0.19370
## 16  2.00229  0.1090627 -0.33414  0.33760 -0.50097 -0.76159
## 17 -1.47545  2.7724102  0.40859  0.75117 -2.59425 -1.10122
## 18 -0.31241  0.6328355 -0.66501 -1.12728  2.65575  0.97565
## 19 -0.69027  3.2642026  1.95716 -0.17694 -0.07352 -0.16083
## 20  1.94438  1.0688809 -0.66595 -0.55317  1.59606 -1.70292
```

Principal coordinates analysis (PCoA)

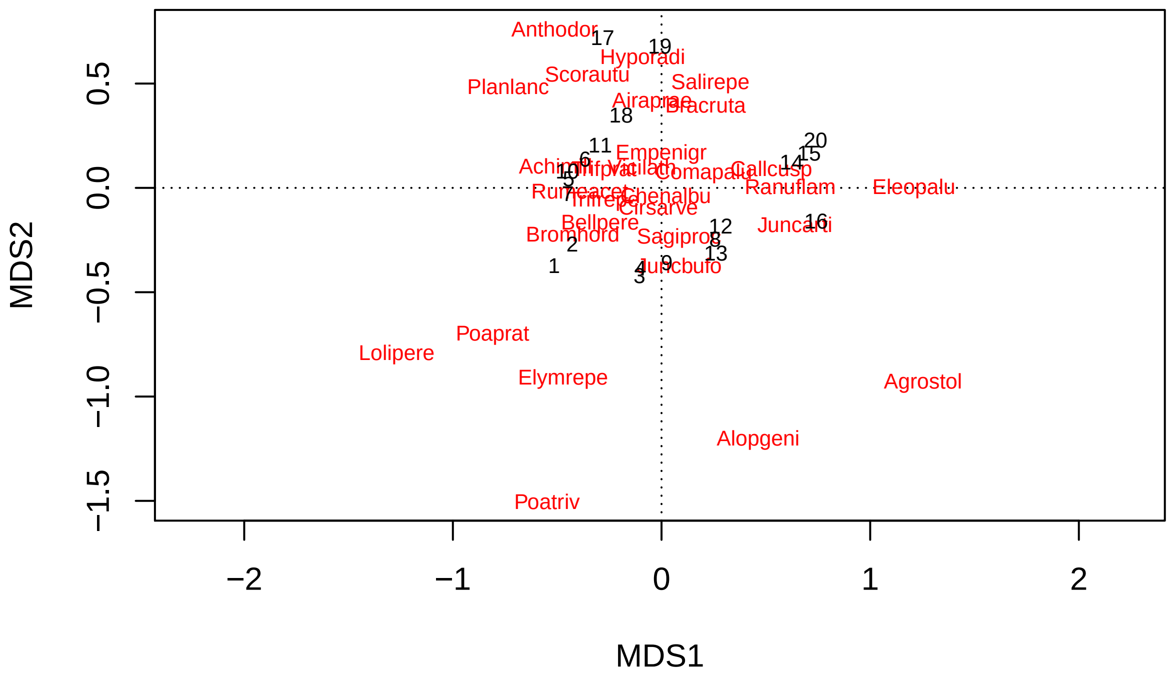
PCA and CA represent the Euclidean and Chi-square distance between sites/samples, respectively, in the ordination plots. In ecology, other ecological distance metrics are more appropriate to preserve and represent on the ordination (ie Bray-Curtis, Jaccard). Principal coordinates analysis (PCoA) analyses a matrix of dissimilarities (ie distances between pairs of sites/samples) computed from these other distance metrics.

The function capscale() in "vegan" runs PCoA. You include "~1" after your species by site matrix to run a PCoA and NOT a canonical analysis with explanatory variables (db-RDA - see below).

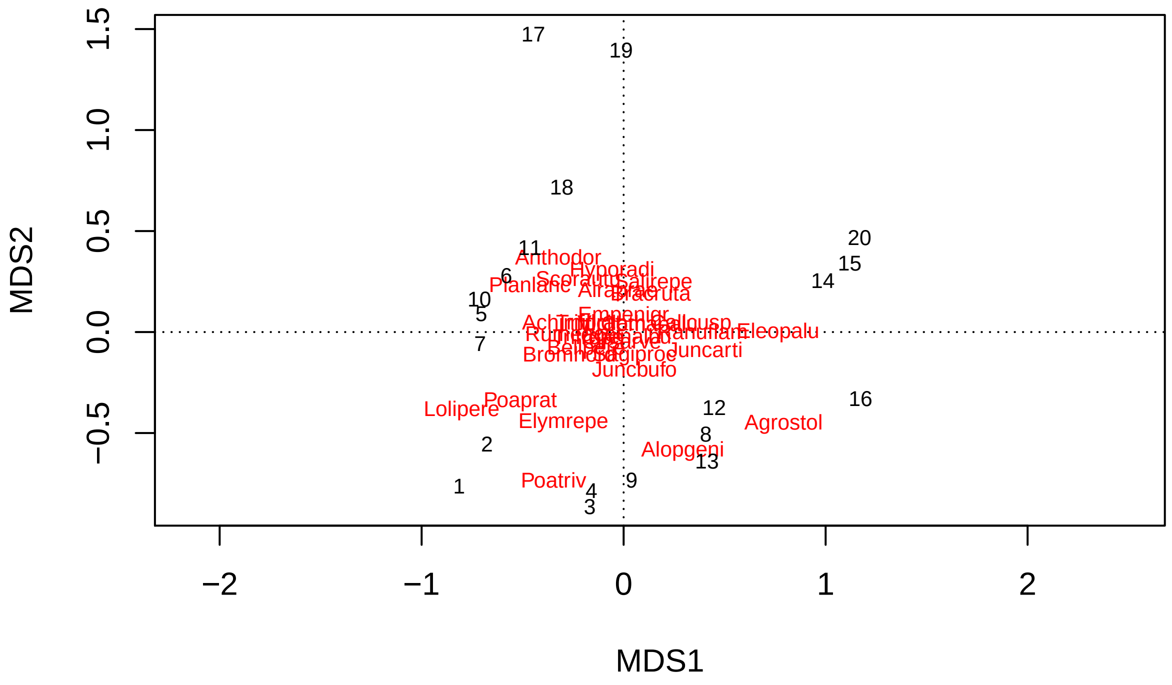
```r
#Run the PCoA (also called metric multidimensional scaline - MDS):

pcoa.out_bray<-capscale(dune~1, distance="bray") #Using Bray-Curtis distance
pcoa.out_jaccard<-capscale(dune~1, distance="jaccard") #Using Jaccard distance

#Distances can also be calculated first using vegdist from "vegan"
bray_dist<-vegdist(dune, distance="bray")
jaccard_dist<-vegdist(dune, distance="jaccard", binary=TRUE) #binary=TRUE caculates distance based on p


#Scree plots (percentage of variation explained by each principal component axis):
screeplot(pcoa.out_bray) #drop-off after second axis, suggesting only first two axes need examining
```



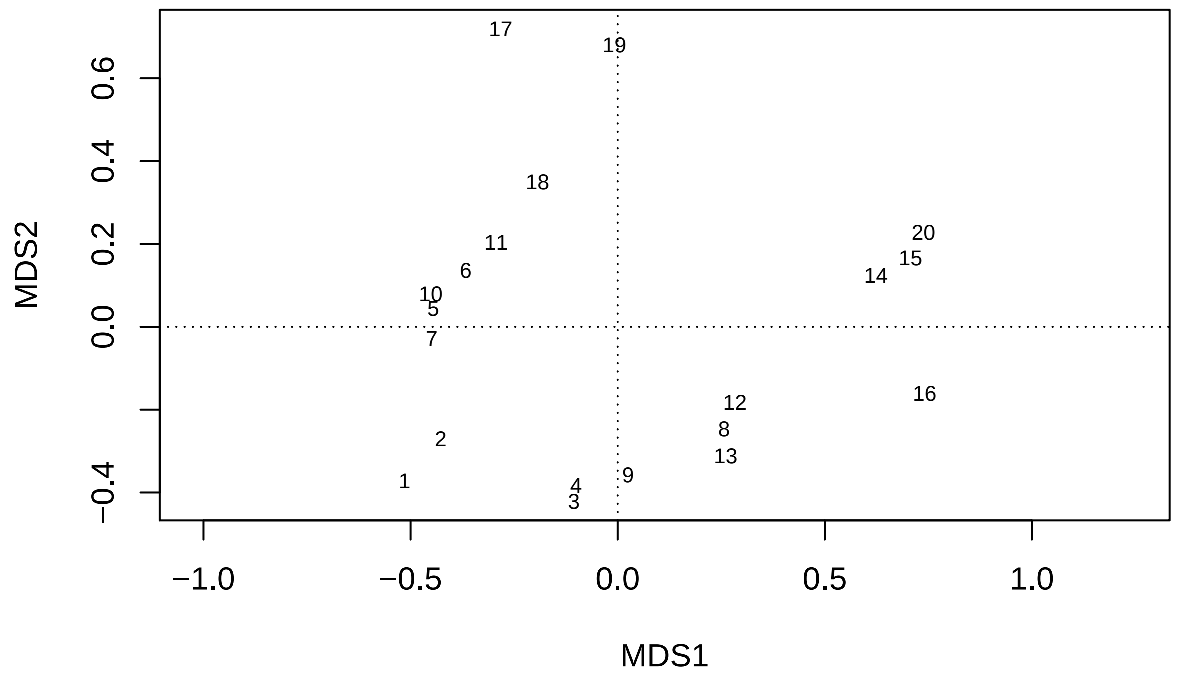
```r
#Examine biplots - NOTE use plot() here:
#scaling=1 (for individuals/sites/samples):
plot(pcoa.out_bray, scaling=1)
```



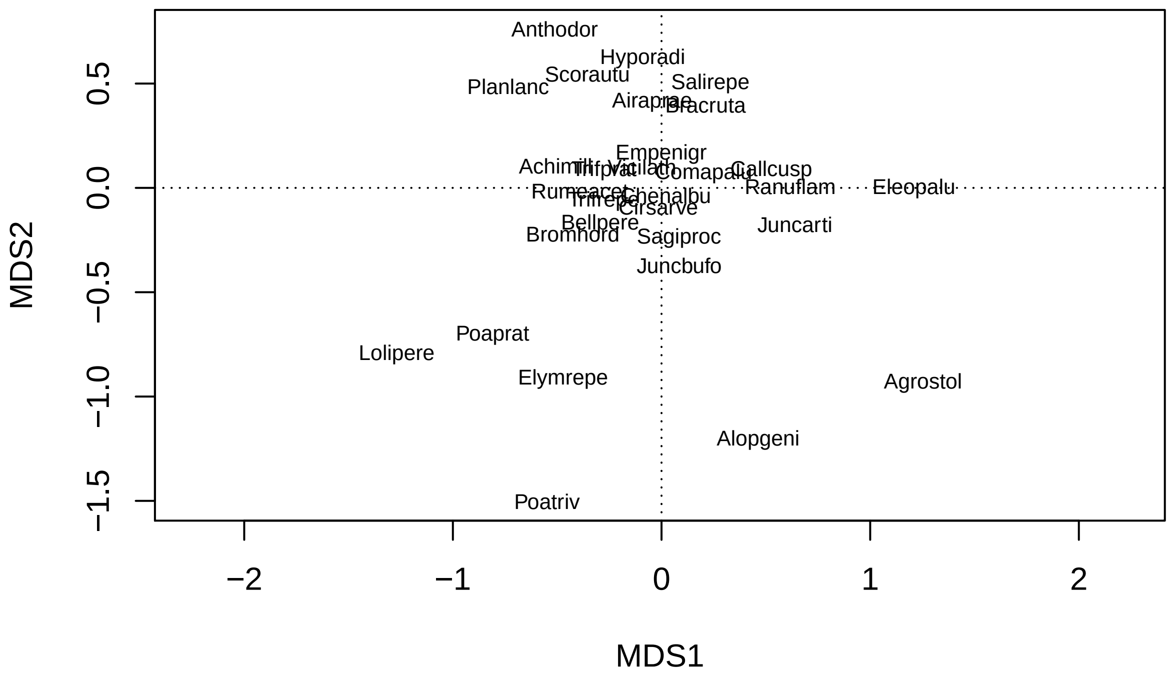
```r
#scaling=2 (for species/response):
plot(pcoa.out_bray, scaling=2)
```



```r
#only show the individuals/sites/samples:
plot(pcoa.out_bray, scaling=1, display="sites")
```



```r
#only show the species/response:
plot(pcoa.out_bray, scaling=1, display="species")
```



```r
#Summary of results - proportion of variance explained, species scores, PC values for each site.
summary(pcoa.out_bray)
```

```
##
## Call:
## capscale(formula = dune ~ 1, distance = "bray")
##
## Partitioning of squared Bray distance:
##               Inertia Proportion
## Total           4.594          1
## Unconstrained   4.594          1
##
## Eigenvalues, and their contribution to the squared Bray distance
##
## Importance of components:
##                         MDS1   MDS2   MDS3    MDS4    MDS5    MDS6    MDS7
## Eigenvalue            1.7163 1.0224 0.4615 0.38225 0.27913 0.23663 0.16912
## Proportion Explained  0.3736 0.2226 0.1005 0.08321 0.06076 0.05151 0.03681
## Cumulative Proportion 0.3736 0.5961 0.6966 0.77980 0.84057 0.89207 0.92889
##                          MDS8    MDS9   MDS10   MDS11    MDS12    MDS13
## Eigenvalue            0.09625 0.07449 0.06171 0.05494 0.019174 0.016119
## Proportion Explained  0.02095 0.01622 0.01343 0.01196 0.004174 0.003509
## Cumulative Proportion 0.94984 0.96605 0.97949 0.99145 0.995620 0.999129
##                           MDS14
## Eigenvalue            0.0040009
## Proportion Explained  0.0008709
## Cumulative Proportion 1.0000000
##
## Scaling 2 for species and site scores
## * Species are scaled proportional to eigenvalues
## * Sites are unscaled: weighted dispersion equal on all dimensions
## * General scaling constant of scores:  3.00629
##
##
## Species scores
##
##               MDS1      MDS2      MDS3     MDS4      MDS5      MDS6
## Achimill -0.320652  0.050785 -0.045297  0.21985  0.111932 -0.005217
## Agrostol  0.792011 -0.456459  0.081159  0.03248 -0.117220  0.014240
## Airaprae -0.028212  0.200637  0.085968  0.02860 -0.112276 -0.029322
## Alopgeni  0.292625 -0.589600  0.454199 -0.07724 -0.072491 -0.025047
## Anthodor -0.323418  0.371692  0.130578  0.23158  0.047438  0.100978
## Bellpere -0.185697 -0.084954 -0.023038 -0.02997  0.062929 -0.109871
## Bromhord -0.268651 -0.108319 -0.012514  0.10262  0.097815 -0.120634
## Chenalbu  0.012504 -0.017985  0.031430  0.01763 -0.007031 -0.013428
## Cirsarve -0.009629 -0.044165  0.003751 -0.01317 -0.026107 -0.012486
## Comapalu  0.127606  0.033620 -0.057412  0.06009  0.060453 -0.047584
## Eleopalu  0.764294 -0.001495 -0.306801  0.10603  0.027716  0.114095
## Elymrepe -0.298166 -0.447175 -0.016911  0.03260 -0.153774  0.006590
## Empenigr -0.000750  0.078421  0.049868 -0.04466 -0.041865 -0.027993
## Hyporadi -0.057090  0.302651  0.112572 -0.08732 -0.166492 -0.080693
## Juncarti  0.404049 -0.085638 -0.100966 -0.09028 -0.051398  0.167292
## Juncbufo  0.053663 -0.181714  0.323283 -0.02073  0.010898  0.060984
## Lolipere -0.801915 -0.389317 -0.361712 -0.22002  0.083093 -0.021242
## Planlanc -0.464304  0.236609 -0.016853  0.01177  0.294222  0.238473
## Poaprat  -0.511688 -0.343515 -0.116610 -0.14507  0.014470 -0.048151
## Poatriv  -0.346395 -0.733533  0.434512  0.24340  0.132518 -0.017296
## Ranuflam  0.389923  0.003347 -0.094346  0.05431  0.006067  0.009029
## Rumeacet -0.247239 -0.007153  0.159051  0.07498  0.282619  0.328080
## Sagiproc  0.053106 -0.116931  0.302545 -0.22042 -0.136081 -0.083687
## Salirepe  0.148042  0.243911 -0.040437 -0.25124 -0.053459  0.003265
## Scorautu -0.224483  0.266618  0.115519 -0.30361  0.101635 -0.200960
## Trifprat -0.173688  0.041251  0.016379  0.05763  0.200367  0.193945
## Trifrepe -0.174976 -0.030751  0.048933  0.05677  0.400922 -0.293581
## Vicilath -0.059037  0.048307 -0.042693 -0.10541  0.015758 -0.041113
## Bracruta  0.133463  0.193444 -0.018002 -0.52302  0.199899  0.250421
## Callcusp  0.332400  0.040490 -0.149837  0.12354  0.043917 -0.031710
##
##
## Site scores (weighted sums of species scores)
##
##        MDS1     MDS2     MDS3     MDS4     MDS5     MDS6
## 1  -0.81403 -0.76313 -1.37762  0.18151 -1.31784  0.31486
## 2  -0.67609 -0.55329 -0.14852  0.28126  0.22457 -1.34598
## 3  -0.16698 -0.86481  0.05174 -0.14445 -0.15041 -0.19105
## 4  -0.15892 -0.78551  0.07235 -0.21454 -0.57466 -0.26946
## 5  -0.70464  0.09013  0.40380  0.56131  0.53594  0.87376
## 6  -0.58064  0.28010 -0.12455  0.08067  1.30909  0.98289
## 7  -0.71055 -0.05671  0.22347  0.17585  0.60184  0.85465
## 8   0.40677 -0.50573 -0.18837 -0.26789  0.02922 -0.01364
## 9   0.03999 -0.73336  0.55679 -0.26637 -0.27337  0.45933
## 10 -0.71422  0.16245 -0.34006  0.34694  0.71410 -0.61786
## 11 -0.46426  0.41960 -0.44870 -1.16090 -0.27186 -0.50454
## 12  0.44857 -0.37442  1.53975 -0.42119  0.32454  0.20611
## 13  0.41276 -0.63974  1.21236  0.57438 -0.30954 -0.57961
## 14  0.98699  0.25606 -0.62525  1.06297  0.99713 -1.34406
## 15  1.11920  0.34190 -0.48204 -0.08416  0.33357  0.31712
## 16  1.17312 -0.32968 -0.44952  0.24935 -0.45323  0.85663
## 17 -0.44708  1.47632  0.21535  1.55708 -1.08913  0.27338
## 18 -0.30611  0.71670 -0.40938 -1.45936  0.52338 -0.14760
## 19 -0.01238  1.39477  0.96179 -0.72748 -0.92154 -0.60413
## 20  1.16849  0.46836 -0.64341 -0.32499 -0.23180  0.47921
```

Non-metric multidimensional scaling

Like PCA, CA, and PCoA, non-metric multidimensional scaling (NMDS) represents multi-dimensional response data in reduced dimensions (typically 2). Unlike PCA, CA, and PCoA, which preserve certain distances, NMDS uses rank orders (ie it preserves the order of data points), making it flexible for wide variety of data types. The program keeps moving the data points in the reduced dimensions until the stress (a representation of how well the representation represents the original data) is low (best <0.1, okay <0.2). The process starts with a matrix of distances (ie Bray-Curtis) and compares the rank-based ordination to those distances to compute the stress using regression.

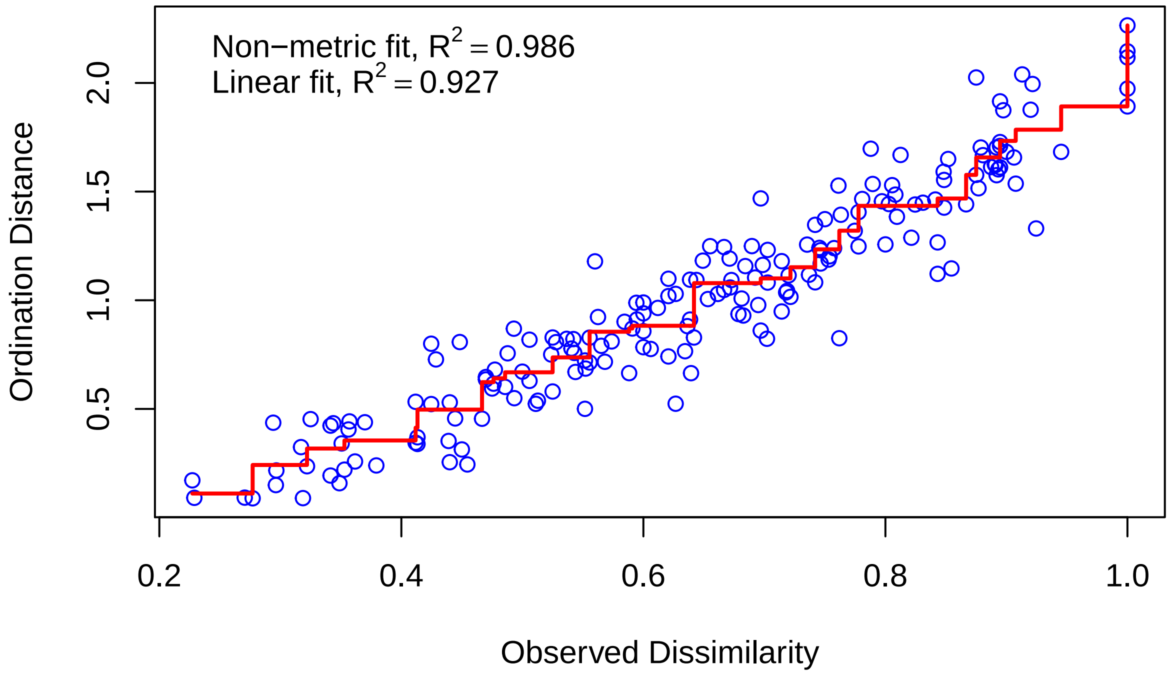
The package is “vegan” and the function metaMDS().

For more info: https://jonlefcheck.net/2012/10/24/nmds-tutorial-in-r/

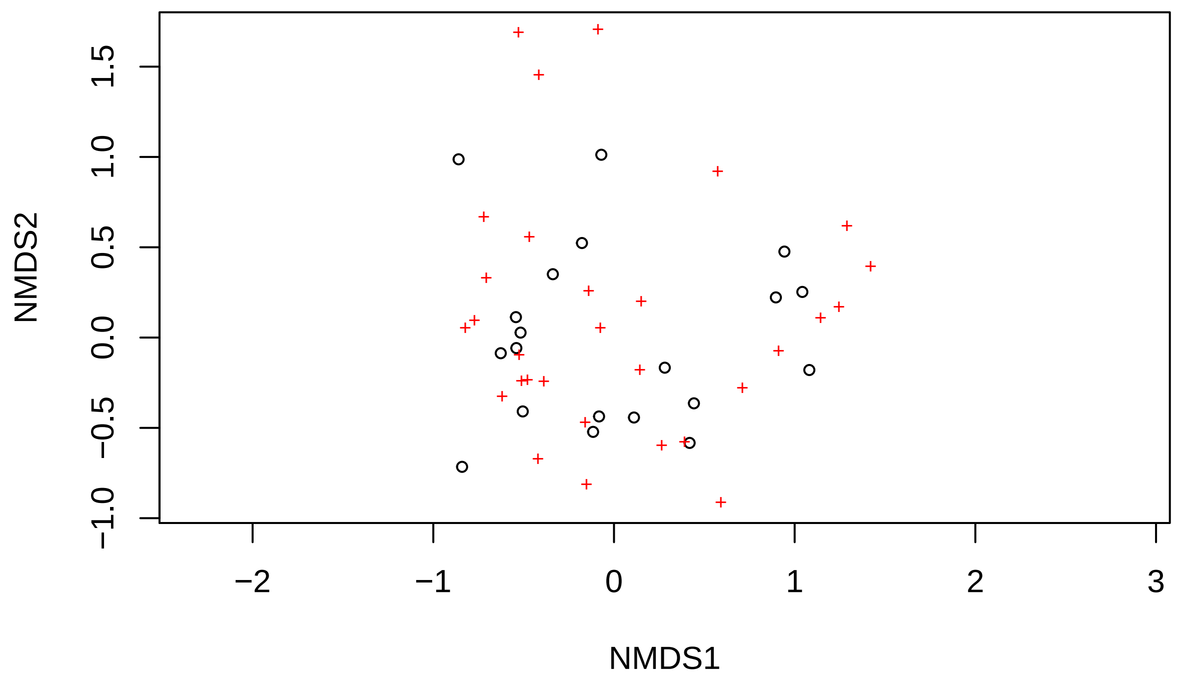
```r
#Run the NMDS


set.seed(2) #NMDS has some randomness, so set the seed to be consistent each time you run.

NMDS.out<-metaMDS(dune, # Our community-by-species matrix
                  distance = "bray", #compute Bray-Curtis distance matrix for use in NMDS
                     k=2, # The number of reduced dimensions
                  trymax=100) #maximum number of iterations tried
```

```
## Run 0 stress 0.1192678
## Run 1 stress 0.1192688
## ... Procrustes: rmse 0.0005664198  max resid 0.001734033
## ... Similar to previous best
## Run 2 stress 0.1183186
## ... New best solution
## ... Procrustes: rmse 0.02027079  max resid 0.06495653
## Run 3 stress 0.1183186
## ... Procrustes: rmse 2.067539e-05  max resid 6.809411e-05
## ... Similar to previous best
## Run 4 stress 0.1192679
## Run 5 stress 0.1889641
## Run 6 stress 0.3443384
## Run 7 stress 0.1192678
## Run 8 stress 0.1192678
## Run 9 stress 0.1192679
## Run 10 stress 0.1808917
## Run 11 stress 0.2980682
## Run 12 stress 0.1183186
## ... Procrustes: rmse 7.943263e-05  max resid 0.0002402825
## ... Similar to previous best
## Run 13 stress 0.1192678
## Run 14 stress 0.1183186
## ... Procrustes: rmse 3.207489e-05  max resid 9.040647e-05
## ... Similar to previous best
## Run 15 stress 0.1886534
## Run 16 stress 0.1183186
## ... Procrustes: rmse 8.447779e-05  max resid 0.0002355591
## ... Similar to previous best
## Run 17 stress 0.1192678
## Run 18 stress 0.1192679
## Run 19 stress 0.1183186
## ... Procrustes: rmse 8.669157e-06  max resid 2.482825e-05
## ... Similar to previous best
## Run 20 stress 0.1192679
## *** Solution reached
```

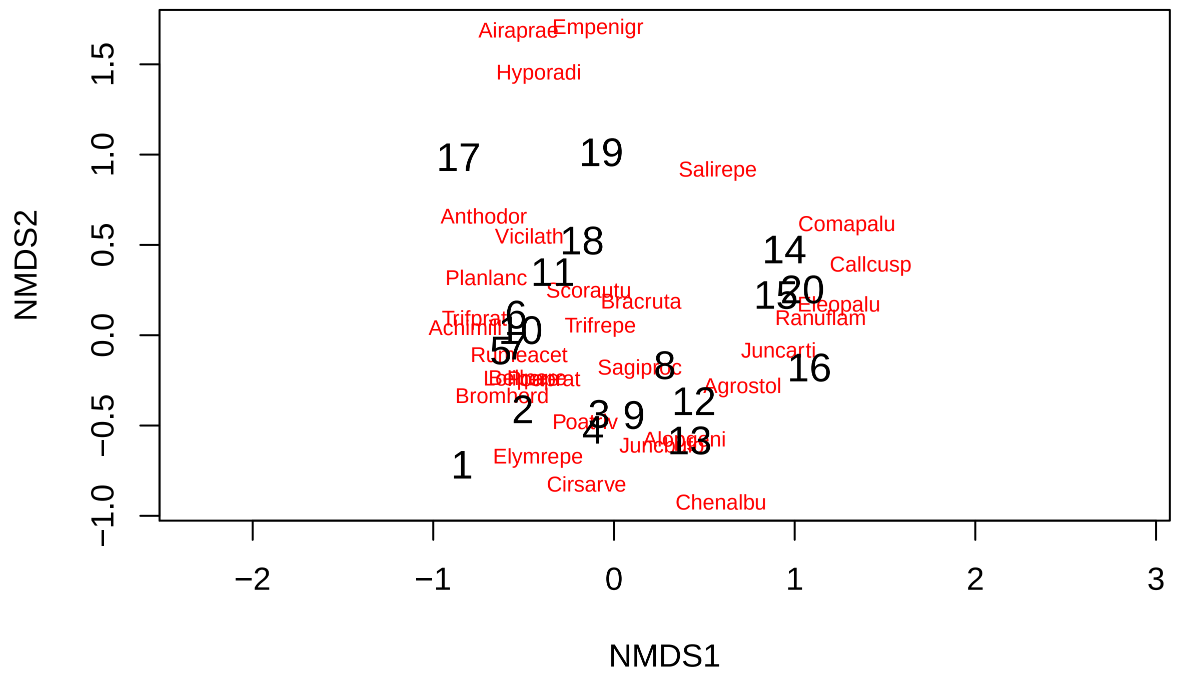
```
#if it fails to converge, try increasing the number of iterations (trymax). If the stress is too high,

#Examine Shepard (stress) plot - large scatter around the line indicates lots of stress (NMDS is a poor
stressplot(NMDS.out)
```



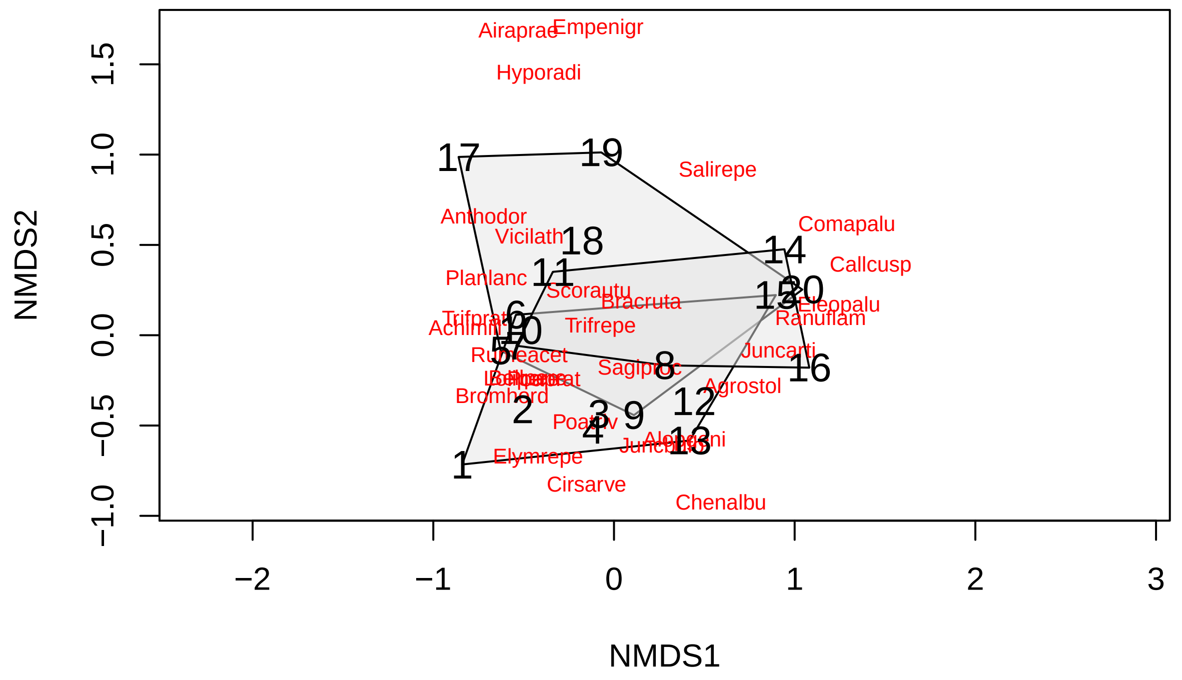
```r
#Examine plots
#can use plot: open circles are sites, red crosses are species:
plot(NMDS.out)
```



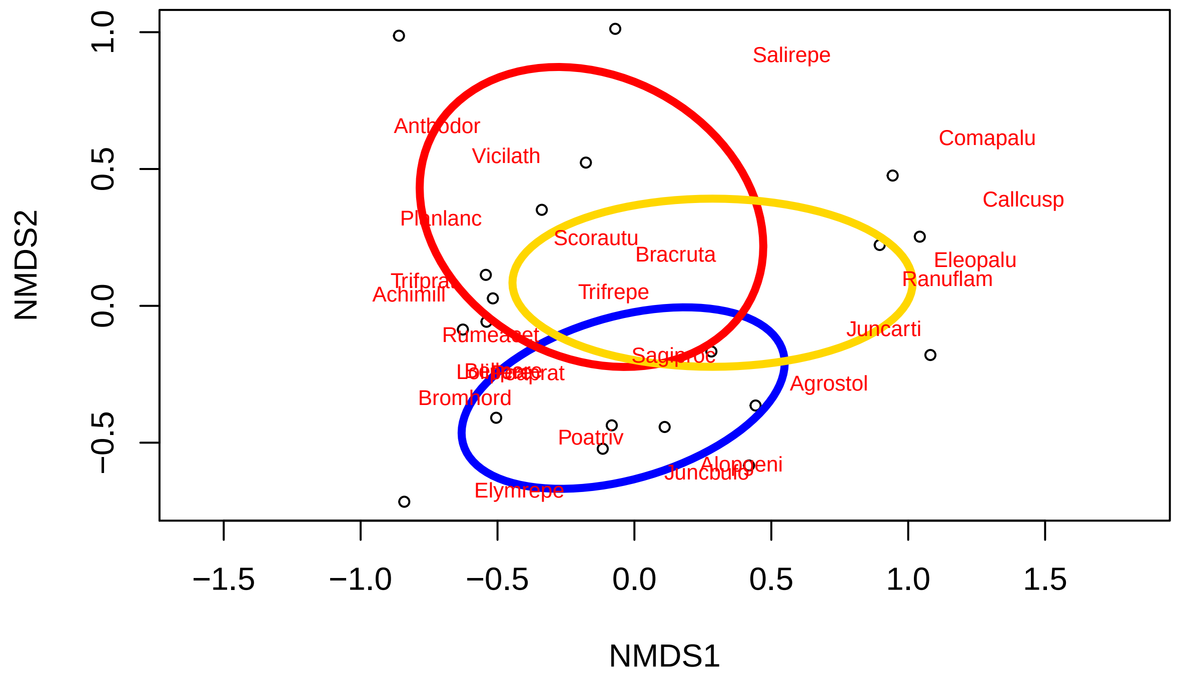
```r
#Plots with labels: first make empty plot with ordiplot() (type="n" specifies the plot is empty) then a
ordiplot(NMDS.out,type="n")
orditorp(NMDS.out,display="species",col="red",air=0.01)
orditorp(NMDS.out,display="sites",cex=1.25,air=0.01)
```



```r
#Is there clustering associated with a factor/treatment/categorical variable:
#example: visualizing effects of land use on ecological community composition:

#look at convex hulls containing all the points for each group:
ordiplot(NMDS.out,type="n")
ordihull(NMDS.out,groups=dune.env$Use,draw="polygon",col="grey90",label=F)
orditorp(NMDS.out,display="species",col="red",air=0.01)
orditorp(NMDS.out,display="sites",air=0.01,cex=1.25)
```



```
#95% confidence ellipse based on a grouping/factor/treatment/categorical variable:

colors<-c("red","blue", "gold") #the colours for the ellipses. Should be as many as you have groups. Go
ordiplot(NMDS.out,display="sites") #special function for making pretty ordination plots (works for PCAs
#Plot convex hulls with colors baesd on treatment
for (i in unique (as.numeric(dune.env$Use))) { #for loop going through and making an ellipse for each l
  ordiellipse(NMDS.out, #function for the ellipse
  groups = as.numeric(dune.env$Use), show.group = i, col = colors[i],
    kind="sd", label = F, scaling=1, lwd=4)}
orditorp(NMDS.out,display="species",col="red",air=0.01) #puts species names on the plot
```



```r
#Summary of results - proportion of variance explained, species scores, PC values for each site.
summary(pcoa.out_bray)
```

```
##
## Call:
## capscale(formula = dune ~ 1, distance = "bray")
##
## Partitioning of squared Bray distance:
##               Inertia Proportion
## Total           4.594          1
## Unconstrained   4.594          1
##
## Eigenvalues, and their contribution to the squared Bray distance
##
## Importance of components:
##                         MDS1   MDS2   MDS3    MDS4    MDS5    MDS6    MDS7
## Eigenvalue            1.7163 1.0224 0.4615 0.38225 0.27913 0.23663 0.16912
## Proportion Explained  0.3736 0.2226 0.1005 0.08321 0.06076 0.05151 0.03681
## Cumulative Proportion 0.3736 0.5961 0.6966 0.77980 0.84057 0.89207 0.92889
##                          MDS8    MDS9   MDS10   MDS11    MDS12    MDS13
## Eigenvalue            0.09625 0.07449 0.06171 0.05494 0.019174 0.016119
## Proportion Explained  0.02095 0.01622 0.01343 0.01196 0.004174 0.003509
## Cumulative Proportion 0.94984 0.96605 0.97949 0.99145 0.995620 0.999129
##                           MDS14
## Eigenvalue            0.0040009
## Proportion Explained  0.0008709
## Cumulative Proportion 1.0000000
##
## Scaling 2 for species and site scores
## * Species are scaled proportional to eigenvalues
## * Sites are unscaled: weighted dispersion equal on all dimensions
## * General scaling constant of scores:  3.00629
##
##
## Species scores
##
##               MDS1      MDS2      MDS3     MDS4      MDS5      MDS6
## Achimill -0.320652  0.050785 -0.045297  0.21985  0.111932 -0.005217
## Agrostol  0.792011 -0.456459  0.081159  0.03248 -0.117220  0.014240
## Airaprae -0.028212  0.200637  0.085968  0.02860 -0.112276 -0.029322
## Alopgeni  0.292625 -0.589600  0.454199 -0.07724 -0.072491 -0.025047
## Anthodor -0.323418  0.371692  0.130578  0.23158  0.047438  0.100978
## Bellpere -0.185697 -0.084954 -0.023038 -0.02997  0.062929 -0.109871
## Bromhord -0.268651 -0.108319 -0.012514  0.10262  0.097815 -0.120634
## Chenalbu  0.012504 -0.017985  0.031430  0.01763 -0.007031 -0.013428
## Cirsarve -0.009629 -0.044165  0.003751 -0.01317 -0.026107 -0.012486
## Comapalu  0.127606  0.033620 -0.057412  0.06009  0.060453 -0.047584
## Eleopalu  0.764294 -0.001495 -0.306801  0.10603  0.027716  0.114095
## Elymrepe -0.298166 -0.447175 -0.016911  0.03260 -0.153774  0.006590
## Empenigr -0.000750  0.078421  0.049868 -0.04466 -0.041865 -0.027993
## Hyporadi -0.057090  0.302651  0.112572 -0.08732 -0.166492 -0.080693
## Juncarti  0.404049 -0.085638 -0.100966 -0.09028 -0.051398  0.167292
## Juncbufo  0.053663 -0.181714  0.323283 -0.02073  0.010898  0.060984
## Lolipere -0.801915 -0.389317 -0.361712 -0.22002  0.083093 -0.021242
## Planlanc -0.464304  0.236609 -0.016853  0.01177  0.294222  0.238473
## Poaprat  -0.511688 -0.343515 -0.116610 -0.14507  0.014470 -0.048151
## Poatriv  -0.346395 -0.733533  0.434512  0.24340  0.132518 -0.017296
## Ranuflam  0.389923  0.003347 -0.094346  0.05431  0.006067  0.009029
## Rumeacet -0.247239 -0.007153  0.159051  0.07498  0.282619  0.328080
## Sagiproc  0.053106 -0.116931  0.302545 -0.22042 -0.136081 -0.083687
## Salirepe  0.148042  0.243911 -0.040437 -0.25124 -0.053459  0.003265
## Scorautu -0.224483  0.266618  0.115519 -0.30361  0.101635 -0.200960
## Trifprat -0.173688  0.041251  0.016379  0.05763  0.200367  0.193945
## Trifrepe -0.174976 -0.030751  0.048933  0.05677  0.400922 -0.293581
## Vicilath -0.059037  0.048307 -0.042693 -0.10541  0.015758 -0.041113
## Bracruta  0.133463  0.193444 -0.018002 -0.52302  0.199899  0.250421
## Callcusp  0.332400  0.040490 -0.149837  0.12354  0.043917 -0.031710
##
##
## Site scores (weighted sums of species scores)
##
##        MDS1     MDS2     MDS3     MDS4     MDS5     MDS6
## 1  -0.81403 -0.76313 -1.37762  0.18151 -1.31784  0.31486
## 2  -0.67609 -0.55329 -0.14852  0.28126  0.22457 -1.34598
## 3  -0.16698 -0.86481  0.05174 -0.14445 -0.15041 -0.19105
## 4  -0.15892 -0.78551  0.07235 -0.21454 -0.57466 -0.26946
## 5  -0.70464  0.09013  0.40380  0.56131  0.53594  0.87376
## 6  -0.58064  0.28010 -0.12455  0.08067  1.30909  0.98289
## 7  -0.71055 -0.05671  0.22347  0.17585  0.60184  0.85465
## 8   0.40677 -0.50573 -0.18837 -0.26789  0.02922 -0.01364
## 9   0.03999 -0.73336  0.55679 -0.26637 -0.27337  0.45933
## 10 -0.71422  0.16245 -0.34006  0.34694  0.71410 -0.61786
## 11 -0.46426  0.41960 -0.44870 -1.16090 -0.27186 -0.50454
## 12  0.44857 -0.37442  1.53975 -0.42119  0.32454  0.20611
## 13  0.41276 -0.63974  1.21236  0.57438 -0.30954 -0.57961
## 14  0.98699  0.25606 -0.62525  1.06297  0.99713 -1.34406
## 15  1.11920  0.34190 -0.48204 -0.08416  0.33357  0.31712
## 16  1.17312 -0.32968 -0.44952  0.24935 -0.45323  0.85663
## 17 -0.44708  1.47632  0.21535  1.55708 -1.08913  0.27338
## 18 -0.30611  0.71670 -0.40938 -1.45936  0.52338 -0.14760
## 19 -0.01238  1.39477  0.96179 -0.72748 -0.92154 -0.60413
## 20  1.16849  0.46836 -0.64341 -0.32499 -0.23180  0.47921
```

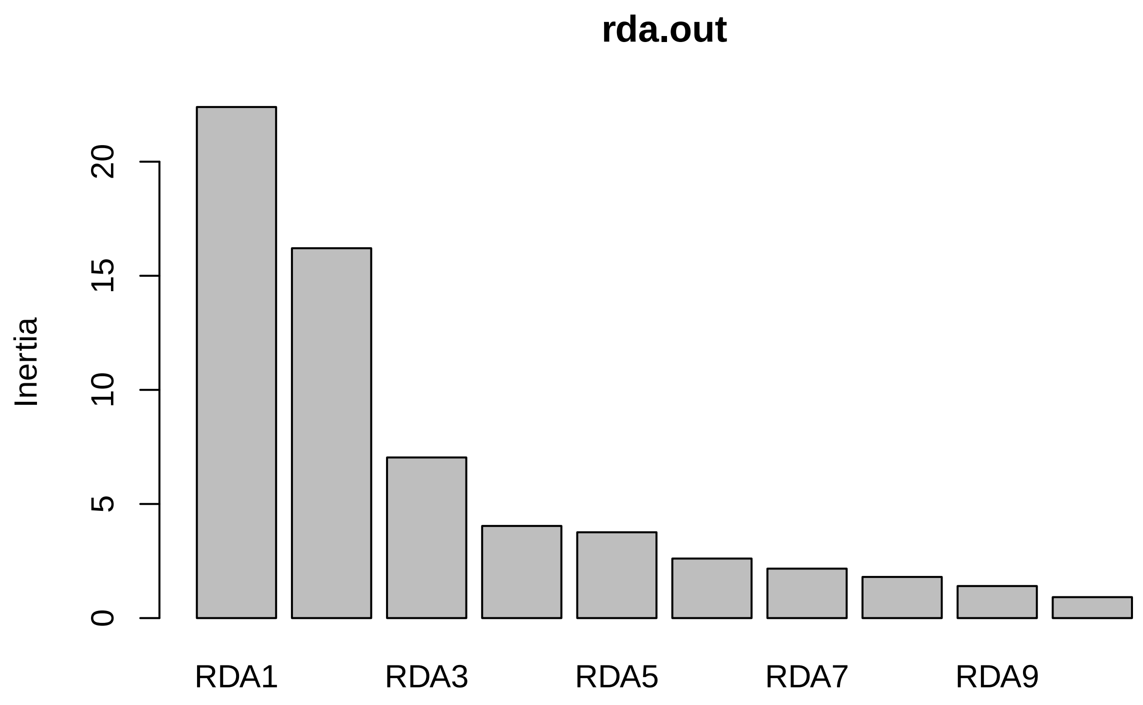
Canonical analyses - RDA, CCA, db-RDA

Canonical or constrained ordinations show variation explained by explanatory variables. Each canonical axis is a linear combination (regression model) of all explanatory variabes. Unconstrained axes are also computed to represent remaining residual variation unaccounted for by explanatory variables.

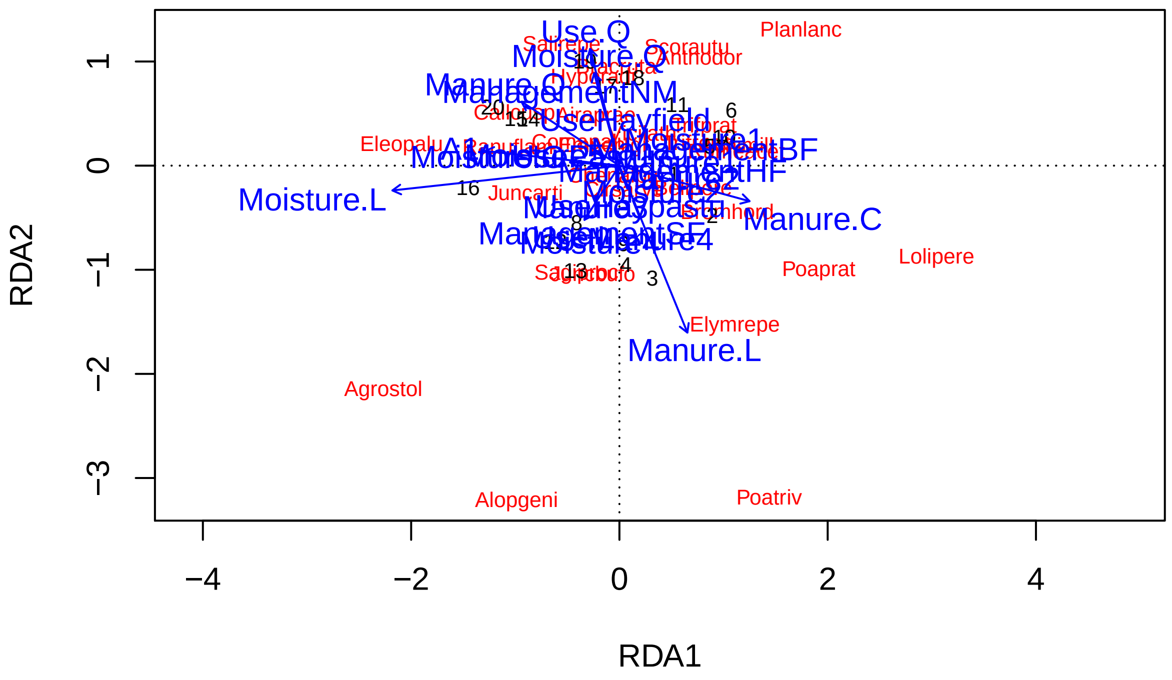
Canonical analysis examines relationship between a response matrix and exlanatory matrix, using both ordination techniques (ie PCA) and linear regression.

Use the same functions in vegan as for unconstraintted ordination above, but include explanatory variables in place of the "~1" ie CommunityData~EnvData

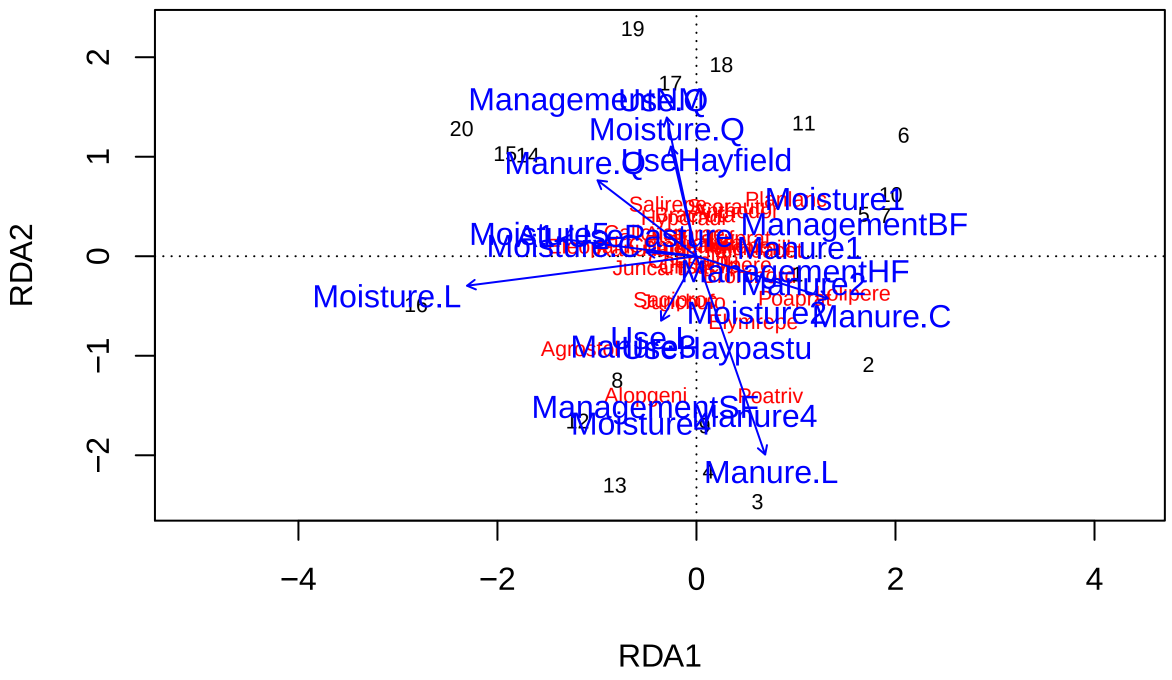
```
#Run redundancy (RDA) - constrained PCA
rda.out<-rda(dune~Use+Management+A1+Moisture+Manure, data=dune.env) #Run the RDA on the dune community

#Run constrained correspondance analysis (CCA)
cca.out<-cca(dune~Use+Management+A1+Moisture+Manure, data=dune.env) #Run the CCA on the dune community

#Run distanced-based redundancy analysis - constrained PCoA
dbRDA.out<-capscale(dune~Use+Management+A1+Moisture+Manure, distance="bray", data=dune.env) #Run the CC

#Code below is for all constrained analyses, although example is for RDA:

#Scree plots (percentage of variation explained by each principal component axis):
screeplot(rda.out) #drop-off after second canonical axis, suggesting only first two axes need examining
```



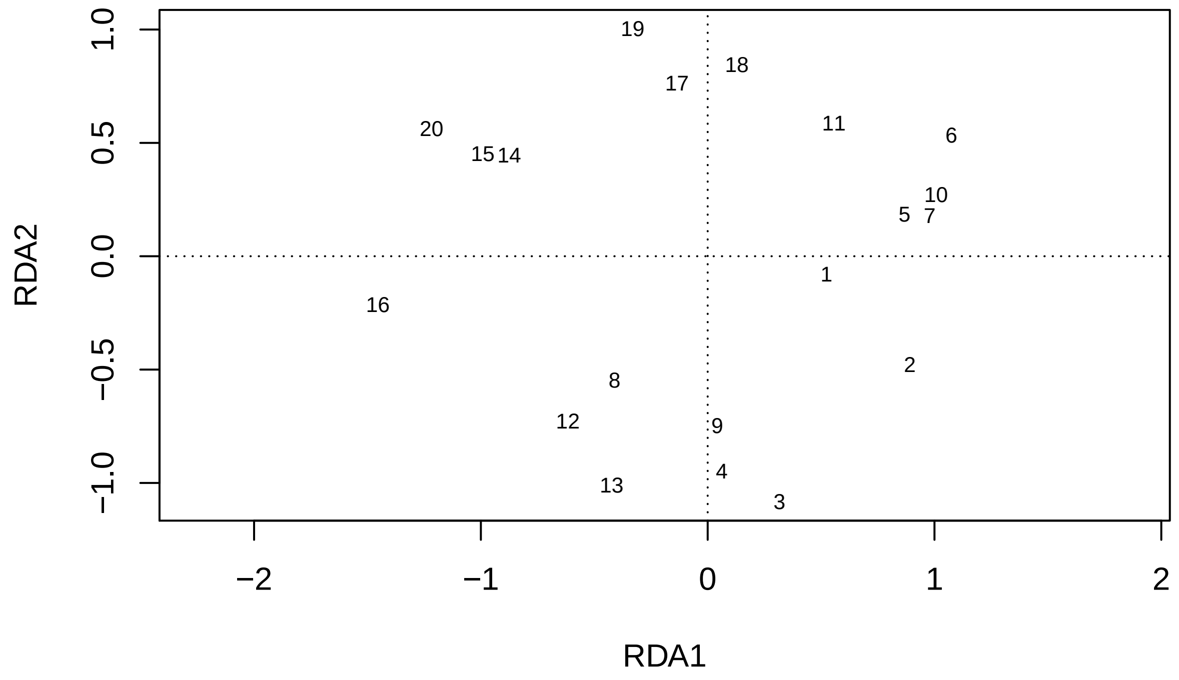
```r
#Examine biplots - NOTE use plot() here:
#scaling=1 (for individuals/sites/samples):
plot(rda.out, scaling=1)
```



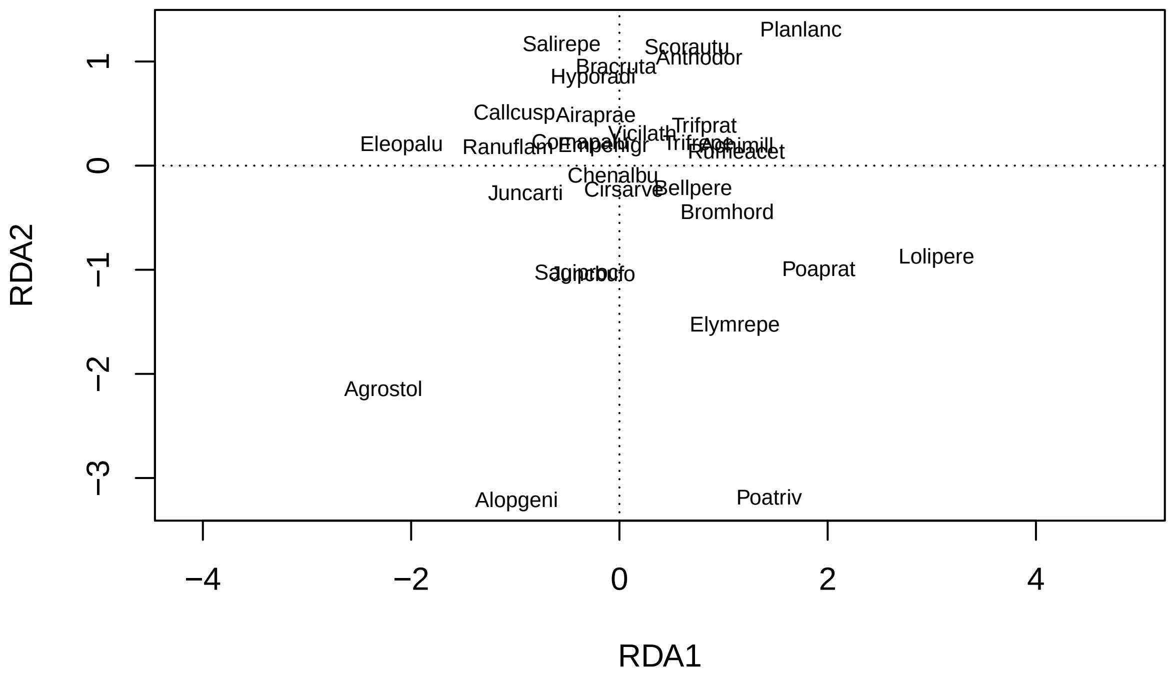
```r
#scaling=2 (for species/response):
plot(rda.out, scaling=2)
```



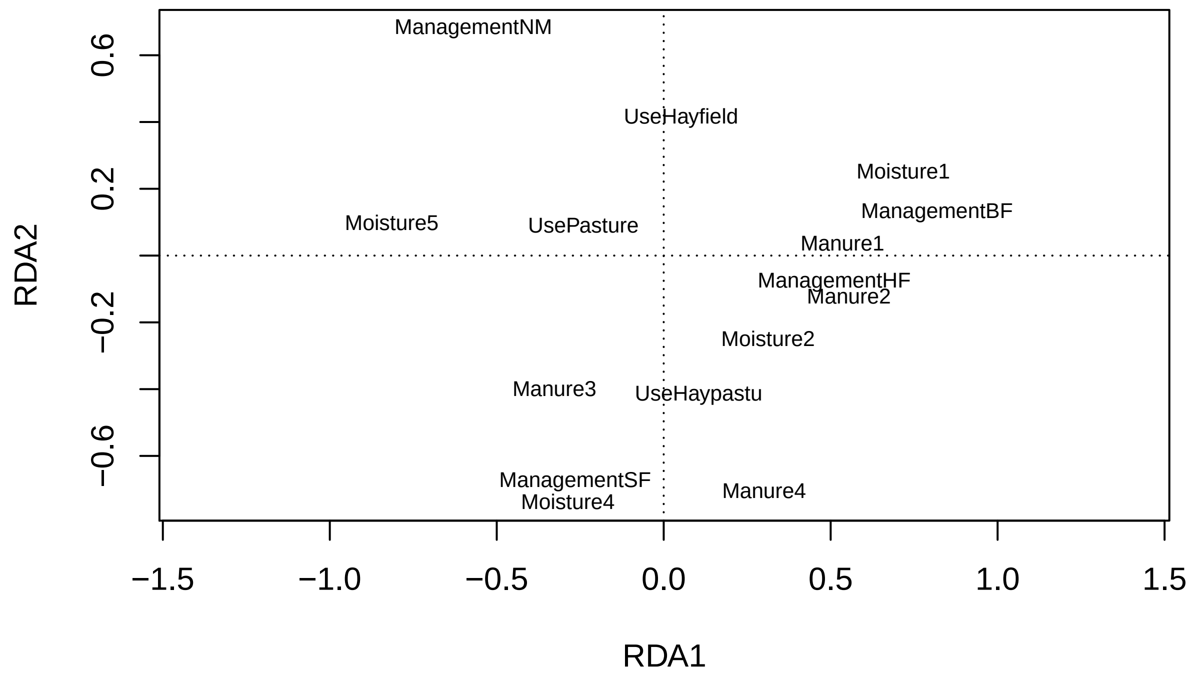
```r
#only show the individuals/sites/samples:
plot(rda.out, scaling=1, display="sites")
```



```r
#only show the species/response:
plot(rda.out, scaling=1, display="species")
```



```r
#only show the constraints/explanatory variables:
plot(rda.out, scaling=1, display="cn")
```



```r
#Summary of results - proportion of variance explained, species scores, PC values for each site.
summary(rda.out)
```

```
##
## Call:
## rda(formula = dune ~ Use + Management + A1 + Moisture + Manure,      data = dune.env)
##
## Partitioning of variance:
##               Inertia Proportion
## Total           84.12     1.0000
## Constrained     63.21     0.7513
## Unconstrained   20.92     0.2487
##
## Eigenvalues, and their contribution to the variance
##
## Importance of components:
##                          RDA1    RDA2    RDA3   RDA4   RDA5    RDA6    RDA7
## Eigenvalue            22.3955 16.2076 7.03891 4.0380 3.7602 2.60874 2.16693
## Proportion Explained   0.2662  0.1927 0.08367 0.0480 0.0447 0.03101 0.02576
## Cumulative Proportion  0.2662  0.4589 0.54256 0.5906 0.6353 0.66627 0.69203
##                          RDA8    RDA9   RDA10    RDA11    RDA12     PC1     PC2
## Eigenvalue            1.80327 1.40421 0.91739 0.581545 0.283927 6.62687 4.30914
## Proportion Explained  0.02144 0.01669 0.01091 0.006913 0.003375 0.07878 0.05122
## Cumulative Proportion 0.71346 0.73015 0.74106 0.747973 0.751348 0.83012 0.88135
##                           PC3     PC4     PC5    PC6      PC7
## Eigenvalue            3.54913 2.54648 2.34027 0.9335 0.612096
## Proportion Explained  0.04219 0.03027 0.02782 0.0111 0.007276
## Cumulative Proportion 0.92354 0.95381 0.98163 0.9927 1.000000
##
## Accumulated constrained eigenvalues
## Importance of components:
##                          RDA1    RDA2   RDA3    RDA4    RDA5    RDA6    RDA7
## Eigenvalue            22.3955 16.2076 7.0389 4.03795 3.76017 2.60874 2.16693
## Proportion Explained   0.3543  0.2564 0.1114 0.06389 0.05949 0.04127 0.03428
## Cumulative Proportion  0.3543  0.6107 0.7221 0.78600 0.84549 0.88676 0.92105
##                          RDA8    RDA9   RDA10    RDA11    RDA12
## Eigenvalue            1.80327 1.40421 0.91739 0.581545 0.283927
## Proportion Explained  0.02853 0.02222 0.01451 0.009201 0.004492
## Cumulative Proportion 0.94958 0.97179 0.98631 0.995508 1.000000
##
## Scaling 2 for species and site scores
## * Species are scaled proportional to eigenvalues
## * Sites are unscaled: weighted dispersion equal on all dimensions
## * General scaling constant of scores:  6.322924
##
##
## Species scores
##
##              RDA1     RDA2      RDA3      RDA4      RDA5      RDA6
## Achimill  0.58181  0.08704 -0.105553 -0.043694 -0.422290 -0.062221
## Agrostol -1.16975 -0.94792  0.060309  0.334142  0.030721  0.222829
## Airaprae -0.11716  0.20786  0.018154 -0.180185 -0.050376  0.041993
## Alopgeni -0.50957 -1.41677 -0.322262 -0.116416 -0.013318 -0.300469
## Anthodor  0.39549  0.45832 -0.411832 -0.136384 -0.336365  0.263272
## Bellpere  0.36549 -0.10133  0.162631 -0.160330 -0.370665  0.137716
## Bromhord  0.53405 -0.19418  0.161217 -0.048422 -0.585999 -0.030597
## Chenalbu -0.03146 -0.03881 -0.034281  0.004236 -0.017102  0.003806
## Cirsarve  0.02198 -0.09707  0.075983  0.009750  0.017309  0.069692
## Comapalu -0.19434  0.09464  0.045000  0.157251 -0.021772 -0.021112
## Eleopalu -1.07946  0.08439  0.006165  0.698662 -0.258796  0.127090
## Elymrepe  0.57194 -0.67735  0.299352 -0.407119  0.233601  0.408690
## Empenigr -0.07889  0.08081  0.032442 -0.076400 -0.005971  0.030773
## Hyporadi -0.12994  0.36832  0.135169 -0.148462  0.009242 -0.069277
## Juncarti -0.46506 -0.11361 -0.043146  0.023106  0.051743  0.032617
## Juncbufo -0.13148 -0.45491 -0.399997 -0.320717  0.288602 -0.369510
## Lolipere  1.57085 -0.39016  0.727211  0.554451  0.340886 -0.082511
## Planlanc  0.89986  0.57656 -0.564648  0.273286  0.038960  0.120162
## Poaprat   0.98798 -0.44183  0.418705  0.121187  0.153703  0.036591
## Poatriv   0.74253 -1.39782 -0.559135  0.008607 -0.428312  0.162203
## Ranuflam -0.55236  0.08108  0.004173  0.185963 -0.131421  0.083262
## Rumeacet  0.57938  0.06076 -0.940669  0.137799  0.309740  0.163586
## Sagiproc -0.21299 -0.45646  0.011857 -0.174011  0.208064 -0.216092
## Salirepe -0.28603  0.50726  0.148922 -0.456522  0.030091  0.164904
## Scorautu  0.33487  0.50344  0.086424 -0.148075 -0.258282 -0.288324
## Trifprat  0.42084  0.16346 -0.528815  0.260068  0.196986  0.120890
## Trifrepe  0.39232  0.08889 -0.146355  0.260759 -0.280784 -0.613465
## Vicilath  0.11379  0.13700  0.111709  0.049534  0.011107 -0.146626
## Bracruta -0.01603  0.41968 -0.395909  0.008729  0.367890 -0.142926
## Callcusp -0.52015  0.21794  0.061946  0.231494 -0.019057  0.025981
##
```

```
##
## Site scores (weighted sums of species scores)
##
##        RDA1    RDA2     RDA3     RDA4     RDA5    RDA6
## 1   1.01596 -0.1799  2.59463 -0.11443  2.05964  2.1377
## 2   1.72853 -1.0870  1.31739 -0.89469 -3.45007 -1.0625
## 3   0.61315 -2.4671  1.08784 -0.06821  0.64265  0.3666
## 4   0.12048 -2.1589  1.71771 -0.07977  0.48648  1.8652
## 5   1.68284  0.4191 -2.55443 -0.80052 -1.22481  3.3283
## 6   2.08298  1.2151 -3.01397  2.45080  1.69576 -0.3949
## 7   1.89757  0.4084 -1.15310  1.15696  0.19469  0.1648
## 8  -0.79508 -1.2469  0.59088  1.85859  0.03913 -0.7791
## 9   0.08193 -1.7029 -0.41962 -2.00241  1.90642  0.0328
## 10  1.95184  0.6210  0.49322  0.87955 -3.21475 -1.7550
## 11  1.07928  1.3401  1.77536  0.93077  2.20291 -2.9068
## 12 -1.19429 -1.6563 -2.36651 -1.91407  1.14492 -3.7773
## 13 -0.82017 -2.3013 -1.46058 -1.16242 -1.40595 -0.6812
## 14 -1.69462  1.0179  0.55226  2.12686 -1.38338 -1.1655
## 15 -1.92197  1.0316  0.18669  1.38332  0.21043  0.9451
## 16 -2.81781 -0.4853 -0.45260  2.98066 -0.24307  2.1409
## 17 -0.26124  1.7408 -0.04449 -2.00579 -0.85795  1.4187
## 18  0.25044  1.9245  0.47979 -1.27429  1.11822 -1.1395
## 19 -0.64055  2.2851  0.18205 -3.72278 -0.40927 -1.1779
## 20 -2.35927  1.2820  0.48747  0.27188  0.48801  2.4397
##
##
## Site constraints (linear combinations of constraining variables)
##
##        RDA1     RDA2     RDA3     RDA4     RDA5     RDA6
## 1   0.87487 -0.90608  2.36103 -0.66159  2.53531  1.55448
## 2   1.91194 -0.62145  1.07037 -0.88335 -3.00393 -2.04867
## 3   0.43540 -1.95933  1.52026  0.20428  0.30748  1.42187
## 4   0.43933 -1.94048  1.51888  0.19490  0.34600  1.39312
## 5   1.65691  0.51332 -2.47861  0.06329 -1.37887  3.19410
## 6   2.10890  1.12091 -3.08978  1.58700  1.84982 -0.26074
## 7   1.48330 -0.04811 -0.36778  1.16790  0.69199 -0.12568
## 8  -0.56422 -1.25593  0.05259  1.83632 -0.90431  0.49765
## 9   0.26534 -1.23731 -0.66664 -1.99108  2.35256 -0.95341
## 10  1.52623 -0.09659  0.82501 -0.36238 -2.76909 -0.83620
## 11  1.32148  1.59213  1.69059  2.16136  1.31111 -2.83941
## 12 -1.37770 -2.12186 -2.11948 -1.92541  0.69878 -2.79106
## 13 -1.25782 -1.55141 -1.37052  0.16936 -0.68373  0.15215
## 14 -2.37447  1.51577  0.72709  2.22806  0.23064 -0.39953
## 15 -1.51025  0.37596  0.17244  0.91534 -0.66586 -0.02248
## 16 -2.19675 -0.76968 -0.78968  1.66022 -0.51916  0.32129
## 17  0.02328  1.73218 -0.60987 -1.31102 -0.82795 -0.08329
## 18  0.38018  2.38966  0.25985 -1.97999  0.59091  0.65301
## 19 -1.57691  1.61530  0.64851 -1.52722 -0.11937  0.61515
## 20 -1.56904  1.65300  0.64575 -1.54597 -0.04232  0.55766
##
##
## Biplot scores for constraining variables
##
##                 RDA1     RDA2      RDA3     RDA4      RDA5     RDA6
## Use.L        -0.1398 -0.25485  0.123795  0.81530  0.138325 -0.26210
## Use.Q        -0.1173  0.54889 -0.004563  0.02884 -0.099898  0.04341
## ManagementHF  0.4043 -0.07409 -0.534962  0.21752  0.213258  0.19208
## ManagementNM -0.5114  0.71630  0.142286 -0.24855 -0.064357  0.10191
## ManagementSF -0.2379 -0.71375  0.086470 -0.02765  0.207180  0.15834
## A1           -0.5361  0.06882 -0.218452  0.31811 -0.146216  0.05232
## Moisture.L   -0.9078 -0.11664 -0.056742  0.05789 -0.136471 -0.01179
## Moisture.Q   -0.1012  0.43397 -0.036128  0.40058 -0.008317  0.14269
## Moisture.C   -0.1978  0.01609  0.364691  0.19843 -0.452618  0.35989
## Manure.L      0.2718 -0.78470  0.068253  0.26784  0.140391  0.28325
## Manure.Q     -0.3913  0.30047  0.585951 -0.19204  0.183926  0.39222
## Manure.C      0.5204 -0.16720  0.322839 -0.18763  0.228598 -0.20896
##
##
## Centroids for factor constraints
##
##                 RDA1     RDA2      RDA3     RDA4    RDA5     RDA6
## UseHayfield   0.1009  0.93851 -0.196574 -1.23634 -0.3134  0.44957
## UseHaypastu   0.2031 -0.95047  0.007901 -0.04993  0.1730 -0.07517
## UsePasture   -0.4661  0.20684  0.262562  1.81077  0.1621 -0.50914
## ManagementBF  1.5865  0.29136  1.195323  0.30521 -1.4873 -1.90809
## ManagementHF  0.9900 -0.18143 -1.310046  0.53268  0.5222  0.47038
## ManagementNM -1.1045  1.54698  0.307294 -0.53680 -0.1390  0.22008
## ManagementSF -0.5138 -1.54147  0.186749 -0.05971  0.4474  0.34198
## Moisture1     1.3911  0.57720 -0.079190  0.20780  0.3709  0.01816
## Moisture2     0.6061 -0.56605  0.813569 -0.31856 -0.7359  0.47388
## Moisture4    -0.5562 -1.67959 -1.393063 -1.95824  1.5257 -1.87223
## Moisture5    -1.5785  0.22614  0.012311  0.53373 -0.3863  0.24598
## Manure1       1.0377  0.08608  0.616318 -0.06403  0.2982 -1.54300
## Manure2       1.0750 -0.27727 -1.654376 -0.28962 -0.4585 -0.47659
## Manure3      -0.6339 -0.90628 -0.618849  1.20845 -0.3538  0.21135
## Manure4       0.5832 -1.60196  1.800060 -0.08747  1.0629  1.45649
```

If else statements

Resource: https://www.datamentor.io/r-programming/if-else-statement/

If else statements are used to run code based on conditions. A Boolean statement is supplied (something that can be evaluated as TRUE or FALSE), and if TRUE, the code following the if statement is run. An optional else statement runs if the Boolean is FALSE.

There are two ways to run if else statements.

Simple if else statements can be run using the ifelse() function. The ifelse function takes three arguments: the first is the Boolean (TRUE/FALSE) statement to evaluate, the second is what to do if it returns TRUE, the third what to do if it returns FALSE.

ifelse() is often used on vectors to run functions to parts of the vector, but a different function (or nothing) on the rest.

Example: if the mammal is big (greater than 100 kgs), log its brain mass, otherwise take the square root of brain mass, and save results to a new column in the data frame.

```r
data(mammals)

mammals$log_sqrt <- ifelse(mammals$body>100, log(mammals$brain), sqrt(mammals$brain))
```

You can also construct if else statements for more complicated computations like this:

if (test that returns TRUE/FALSE) { code/function to run }

You can specify what to do when FALSE using else:

if (test that returns TRUE/FALSE) { code/function to run when TRUE } else { code/function to run when FALSE }

You can chain multiple if else statements:

if (test that returns TRUE/FALSE) {

code/function to run

} else {

```
if (second condition) {

code/function to run

} else {

  code/function to run

}
```

}

```r
size <- 855
type <- "bird"

if (size > 1000) {
  print("That's a big animal!")
}

if (size < 1000) {
  print("That's a small animal!")
}
```

```
## [1] "That's a small animal!"
```

```r
if (size >= 1000 & type == "bird") {
  print("That's a big bird")
} else {
  if (size >= 1000 & type == "mammal") {
    print("that's a big mammal")
  } else {
    if (size < 1000 & type!="bird") {
      print("That's a small animal, can't tell what, but I know it's not a bird")
    } else {
      print("Wow that's a small bird!")
    }
  }
}
```

```
## [1] "Wow that's a small bird!"
```

```
#Try changing size and type and see what happens
```

Make your own function in R

If you find yourself repeating (copying and pasting) code many times with only minor tweaks, it might be useful to make that code a function - then you only need to repeat the function name, with arguments representing those minor tweaks or the data used.

You give the function a name and assign (using <- or =) the actual function to that name.

Here's the function for calculating standard error

```
se_func <- function(x) { #put the arguments for your function in the (). This function takes a vector "
  return(sd(x)/sqrt(length(x))) #use return to specify what the function should output. Using return is
} #enclose the body of your function in brackets
```

This function takes a data frame and another function as argument and calculates that function over each column of the data frame. It also includes an example of a for loop (see For loops section below).

```
col_summary <- function(df, fun) {
  result <- vector("numeric", length(df)) #initialize an empty vector to store results of for loop
  for (i in seq_along(df)) {
    result[i] <- fun(df[[i]])
  }
  return(result)
}

#use the function to calculate the mean, median of every column of the dune data set
#NOTE this re-creates functionality from summary() and apply()

col_summary(dune, mean)
```

```
##  [1] 0.80 2.40 0.25 1.80 1.05 0.65 0.75 0.05 0.10 0.20 1.25 1.30 0.10 0.45 0.90
## [16] 0.65 2.90 1.30 2.40 3.15 0.70 0.90 1.00 0.55 2.70 0.45 2.35 0.20 2.45 0.50
```

```
col_summary(dune, median)
```

```
##  [1] 0.0 1.5 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 2.0 0.0 3.0
## [20] 4.0 0.0 0.0 0.0 0.0 2.0 0.0 2.0 0.0 2.0 0.0
```

If else statements in functions

If else statements can be used in functions. A TRUE/FALSE condition can be coded as an argument if you have different variations to run for different conditions.

When a value is supplied when the function is made,

Example: calculate SE for continuous numbers that can vary from negative infinity to infinity versus SE for proportions that vary between 0 and 1. Uses a nested if/else statement – one statement within another

```r
se_func_complete <- function(x, prop=FALSE) {

  if (prop==TRUE) {

    #check if numbers are between 0 and 1:

    if (min(x) < 0 | max(x) > 1) {
      print("Error: Data should be bounded by 0 and 1")
    }
    else {
      return(sqrt(x*(1-x)/length(x))) #SE of proportion data (bounded by 0 and 1)
    }

  } else {

      return(sd(x)/sqrt(length(x))) # SE of count/continuous data

  }

}

se_func_complete(c(2,3.3,4.5,99.2), prop=FALSE)
```

```
## [1] 23.98876
```

```r
se_func_complete(c(0.5,0.77,0.6,0.2,0.99), prop=TRUE)
```

```
## [1] 0.22360680 0.18820202 0.21908902 0.17888544 0.04449719
```

```r
se_func_complete(c(2,3.3,4.5,99.2), prop=TRUE)
```

```
## [1] "Error: Data should be bounded by 0 and 1"
```

Repetitive tasks in R

Check out this resource: https://r4ds.had.co.nz/iteration.html

If you need to run some function or code repeatedly, there are a few options to save you time and coding effort.

1. Make use of vectorization in R.

R is vectorized, meaning running a function on a vector applies that function to each element of that vector. Remember, columns of data frames are vectors! You can make use of your own functions this way too!

2. Use the apply family.

The apply family of functions applies a function to every row, column, or element in a list. You can use these with your own custom functions. You can make use of your own functions this way too!

3. Use aggregate
4. For loops

For example:

```
mammals$brain+1
```

```
##  [1]   45.50   16.50    9.10  424.00  120.50  116.00   99.20    6.50   59.00
## [10]    7.40    5.00    6.70    7.60    1.14    2.00   11.80   13.30    7.30
## [19] 4604.00    1.30  420.00  656.00    4.50  116.00   26.60    6.00   18.50
## [28]  681.00  407.00  326.00   13.30 1321.00 5713.00    4.90  180.00   57.00
## [37]   18.00    2.00    1.40    1.25   13.50  491.00   13.10  176.00  158.00
## [46]  441.00  180.50    3.40   82.00   22.00   40.20    2.90    2.20    4.00
## [55]    1.33  181.00   26.00  170.00    3.60   12.40    3.50   51.40
```

Adds 1 to every element of mammals$brain.

Another example: Calculate the Shannon diversity for every row of the dune community data set:

```
library(vegan)
data(dune)
data(dune.env)

diversity(dune, index="shannon")
```

```
##        1        2        3        4        5        6        7        8
## 1.440482 2.252516 2.193749 2.426779 2.544421 2.345946 2.471733 2.434898
##        9       10       11       12       13       14       15       16
## 2.493568 2.398613 2.106065 2.114495 2.099638 1.863680 1.979309 1.959795
##       17       18       19       20
## 1.876274 2.079387 2.134024 2.048270
```

The apply family

The "apply" family of functions applies a function to every element of a row or column (apply) or for every element in a list (lapply, sapply). There are other functions in the family (vapply, tapply) but these are the ones you are most likely to use. NOTE: aggregate() (see below) is a good function to use instead of tapply().

```
#Apply a function to every column of a data frame:
apply(dune,2,function(x) sd(x)/sqrt(length(x))) #find the standard error of each species - 2 means appl
```

```
##  Achimill  Agrostol  Airaprae  Alopgeni  Anthodor  Bellpere  Bromhord  Chenalbu
## 0.2772041 0.6000000 0.1758438 0.5875910 0.3802700 0.2325488 0.3151858 0.0500000
##  Cirsarve  Comapalu  Eleopalu  Elymrepe  Empenigr  Hyporadi  Juncarti  Juncbufo
## 0.1000000 0.1376494 0.5275315 0.4650976 0.1000000 0.2760149 0.3620119 0.3101358
##  Lolipere  Planlanc   Poaprat   Poatriv  Ranuflam  Rumeacet  Sagiproc  Salirepe
## 0.6320393 0.4358899 0.4129483 0.6294317 0.2625783 0.4032761 0.3479262 0.3118282
##  Scorautu  Trifprat  Trifrepe  Vicilath  Bracruta  Callcusp
## 0.3486817 0.2760149 0.4247290 0.1169795 0.4259664 0.2762531
```

```
#Apply a function to every row of a data frame:
apply(dune,1,function(x) sd(x)/sqrt(length(x))) #find the standard error of species counts at each site
```

```
##         1         2         3         4         5         6         7         8
## 0.2940013 0.3942110 0.4021018 0.3917247 0.3345381 0.4196331 0.3298670 0.3263639
##         9        10        11        12        13        14        15        16
## 0.3342516 0.3672931 0.3421814 0.3716485 0.3785939 0.2971125 0.2655869 0.3934814
##        17        18        19        20
## 0.1841414 0.2969190 0.3197820 0.3303893
```

```
#Apply a function to every element of a list
#lapply will return a list
#sapply will return a "prettier" array, matrix, or data frame to manipulate

#first I'm going to make a list using the lapply function
#remember data frames are lists of columns, so can also use lapply to apply function to every column an
#I want to run a regression of abundance against A1 (a measure of soil thickness at the A1 horizon) for
regressions.out<-lapply(dune, function(x) lm(x~dune.env$A1)) #remember a data frame is a list of column

#Now I have a list of the regressions, one for each species. Remember - use $ or [[]] to retrieve eleme
regressions.out[[1]] #gives me first regression in the list
```

```
##
## Call:
## lm(formula = x ~ dune.env$A1)
##
## Coefficients:
## (Intercept)  dune.env$A1
##      1.6761      -0.1806
```

```
regressions.out$Anthodor #gives me the regression for Anthodor
```

```
##
## Call:
## lm(formula = x ~ dune.env$A1)
##
## Coefficients:
## (Intercept)  dune.env$A1
##      1.8106      -0.1568
```

```
#I can use this list of regressions with lapply or sapply to retrieve just the coefficients for each sp
intercept_slopes<-lapply(regressions.out, coef) #get intercept and slope as a list accessible with $
intercept_slopes_df<-sapply(regressions.out, coef) #get intercept and slope as a matrix/array
```

```r
#coef gets the coefficients from a regression object


#NOTE: use colSums(), rowSums(), colMeans(), or rowMeans() for sums and means for columns and rows.
colSums(dune)
```

```
## Achimill Agrostol Airaprae Alopgeni Anthodor Bellpere Bromhord Chenalbu
##       16       48        5       36       21       13       15        1
## Cirsarve Comapalu Eleopalu Elymrepe Empenigr Hyporadi Juncarti Juncbufo
##        2        4       25       26        2        9       18       13
## Lolipere Planlanc  Poaprat  Poatriv Ranuflam Rumeacet Sagiproc Salirepe
##       58       26       48       63       14       18       20       11
## Scorautu Trifprat Trifrepe Vicilath Bracruta Callcusp
##       54        9       47        4       49       10
```

```r
rowMeans(dune)
```

```
##         1         2         3         4         5         6         7         8
## 0.6000000 1.4000000 1.3333333 1.5000000 1.4333333 1.6000000 1.3333333 1.3333333
##         9        10        11        12        13        14        15        16
## 1.4000000 1.4333333 1.0666667 1.1666667 1.1000000 0.8000000 0.7666667 1.1000000
##        17        18        19        20
## 0.5000000 0.9000000 1.0333333 1.0333333
```

Aggregate

Use aggregate to apply a function over groups/levels of a factor (eg mean weight in each treatment).

```r
#Find mean abundance of Anthodor by pasture type
aggregate(dune$Anthodor, by=list(dune.env$Use), mean)
```

```
##    Group.1        x
## 1 Hayfield 2.285714
## 2 Haypastu 0.375000
## 3  Pasture 0.400000
```

```r
#if the response and factor are in the same data frame:
data.example<-data.frame(Anthodor=dune$Anthodor, Use=dune.env$Use)
aggregate(Anthodor~Use, data=data.example, mean) #mean
```

```
##        Use Anthodor
## 1 Hayfield 2.285714
## 2 Haypastu 0.375000
## 3  Pasture 0.400000
```

```r
aggregate(Anthodor~Use, data=data.example, min) #minimum
```

```
##        Use Anthodor
## 1 Hayfield        0
## 2 Haypastu        0
## 3  Pasture        0
```

For loops

The simplest way to achieve a repetitive task is to have R run the code for you in a loop. However, for loops are not very efficient, and most things can be achieved using R's vectorization, the apply family, and making your own functions.

A for loop designates a variable that changes each time the looped code is run.

Here's a for loop basic example:

Take a list of numbers and find their square:

```
#make the data
numberList <- c(22, 33,12,13,5,7,9)

#Initialize an empty vector to store the results of your for loop
squaredNumbers <- vector(mode = "numeric", length = length(numberList))

#as the for loop runs, "i" in the code below will change, one after the other, to every element specifi

for (i in seq_along(numberList)) {
  squaredNumbers[i] <- numberList[i]^2
}

squaredNumbers
```

```
## [1]  484 1089  144  169   25   49   81
```

NOTE this could be better achieved with simple vectorization:

```
squaredNumbersVectorization <- numberList^2

squaredNumbers
```

```
## [1]  484 1089  144  169   25   49   81
```

```
squaredNumbersVectorization
```

```
## [1]  484 1089  144  169   25   49   81
```

Here's another example that prints out various statements. I used two for loops, one nested in the other:

NOTE for the second for loop, I use "j" rather than "i" so they do not get confused. You can pick any letter, but the standard is "i" for the outer loop, then "j" for the inner loop. I also make use of an if else statement.

```
nouns <- c("roses", "violets", "daisies", "sugar cubes", "onions", "oranges")

colours <- c("red", "blue", "white")

tastes <- c("sweet", "gross", "juicy")

for (i in seq_along(nouns)) {
  if (i <= 3) {
    for (j in seq_along(colours)) {
      print(paste(nouns[i], "are", colours[j]))
    }
  } else {
    for (j in seq_along(tastes)) {
      print(paste(nouns[i], "are", tastes[j]))
      ifelse((tastes[j]=="gross" & nouns[i]=="sugar cubes") | (tastes[j]=="sweet" & nouns[i]=="onions")
    }

  }
}
```

```
## [1] "roses are red"
## [1] "roses are blue"
## [1] "roses are white"
## [1] "violets are red"
## [1] "violets are blue"
## [1] "violets are white"
## [1] "daisies are red"
## [1] "daisies are blue"
## [1] "daisies are white"
## [1] "sugar cubes are sweet"
## [1] "That's so true!"
## [1] "sugar cubes are gross"
## [1] "That's a lie"
## [1] "sugar cubes are juicy"
## [1] "That's so true!"
## [1] "onions are sweet"
## [1] "That's a lie"
## [1] "onions are gross"
## [1] "That's so true!"
## [1] "onions are juicy"
## [1] "That's so true!"
## [1] "oranges are sweet"
## [1] "That's so true!"
## [1] "oranges are gross"
## [1] "That's so true!"
## [1] "oranges are juicy"
## [1] "That's so true!"
```

NOTE something similar can be achieved with a function:

```
printNouns <- function(noun, adjective, truth=TRUE) {
  print(paste(noun, adjective, "Stop!", ifelse(truth, "That's so true!", "That's a lie!")))
}

#Notice each "noun" is getting its corresponding adjective in "c(colours, taste)" BUT every colour or t
printNouns(nouns, c(colours, tastes), truth=c(FALSE, FALSE, TRUE, FALSE, FALSE, TRUE))
```

```
## [1] "roses red Stop! That's a lie!"
## [2] "violets blue Stop! That's a lie!"
## [3] "daisies white Stop! That's so true!"
## [4] "sugar cubes sweet Stop! That’s a lie!"
## [5] "onions gross Stop! That’s a lie!"
## [6] "oranges juicy Stop! That’s so true!"
```

```
#can also use just one FALSE. The number of TRUE/FALSE's should be 1 or the same number as the length o
printNouns(nouns, c(colours, tastes), truth=FALSE)
```

```
## [1] "roses red Stop! That’s a lie!"
## [2] "violets blue Stop! That’s a lie!"
## [3] "daisies white Stop! That’s a lie!"
## [4] "sugar cubes sweet Stop! That’s a lie!"
## [5] "onions gross Stop! That’s a lie!"
## [6] "oranges juicy Stop! That’s a lie!"
```

For loop to make multiple plots:

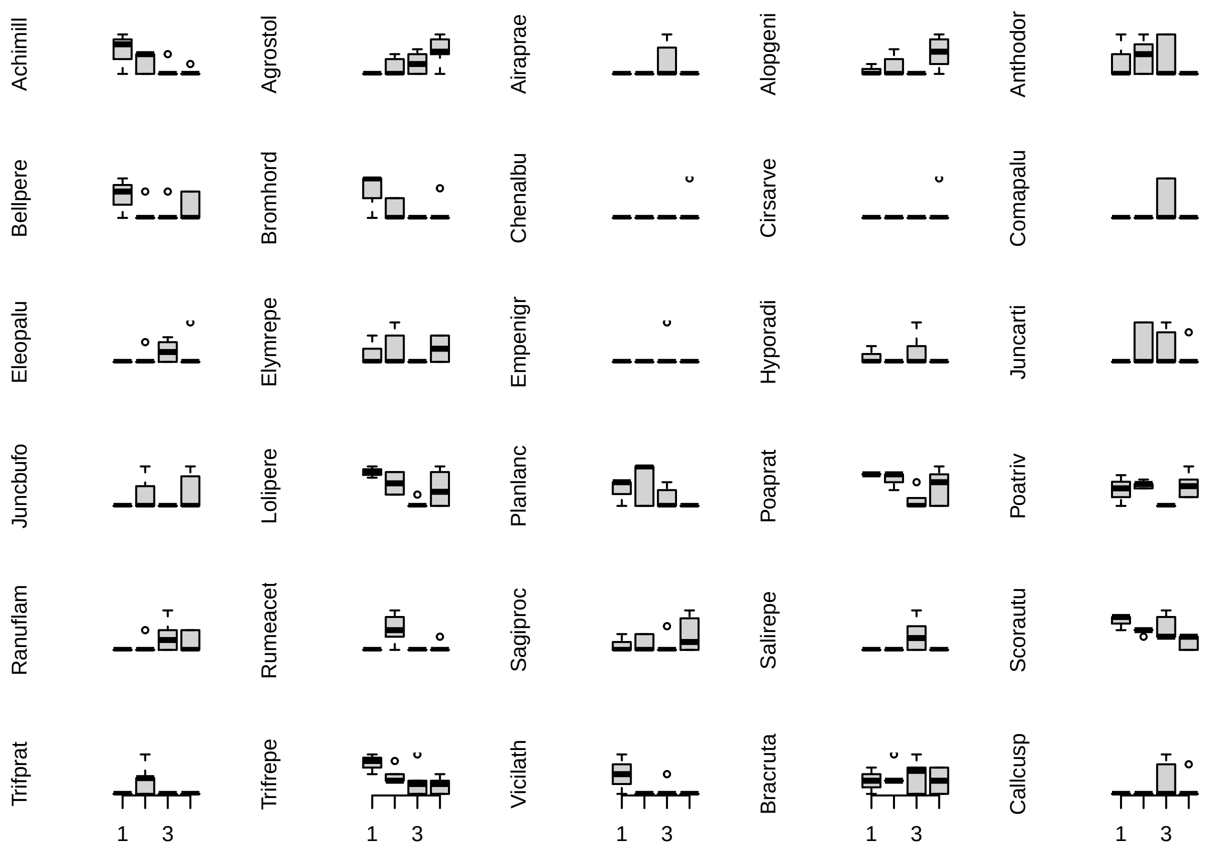
Sometimes you want to make the same plot multiple times for different data.

Example: plot the relationship of each dune species to Moisture

```
#This will produce a plot for each dune species, and give a y-axis based on the name of that species

dim(dune)
```

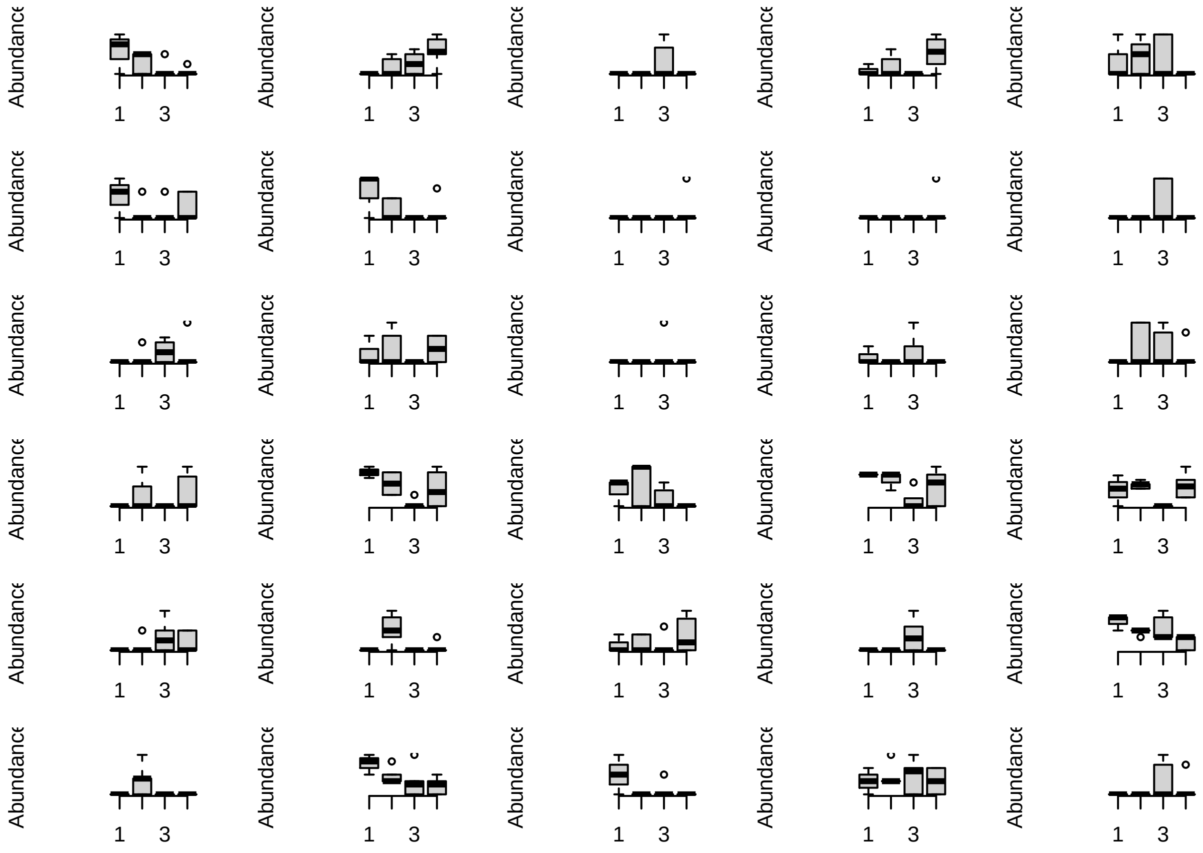
```
## [1] 20 30
```

```
par(mfrow=c(6,5), mar=c(3,5,1,1)) #set up plotting window - six rows, five columns for all thirty speci

#I generate a sequence from 1 to the number of columns in dune (ie number of species in dune; there is
#I use if else statements to ensure that the x-axis is drawn only on the bottom plots - the last five p
for (i in 1:ncol(dune)) {
    plot(dune[,i]~dune.env$Management, ylab=names(dune)[i], xlab="Management level", axes=F)

    if (i > 25) {
      axis(1) # add x-axis
    }

}
```



NOTE: this can be better achieved by making your own function:

```
#This will make a function to make a box plot of a variable against Management

plotDune <- function(x, x_axis=FALSE) {
  plot(x~dune.env$Management, ylab="Abundance", xlab="Management level", axes=F)
   if (x_axis) {
      axis(1) # add x-axis
    }
}


#apply the function to every column:

par(mfrow=c(6,5), mar=c(3,5,1,1)) #set up plotting window - six rows, five columns for all thirty speci

apply(dune, 2, plotDune, x_axis=TRUE)
```

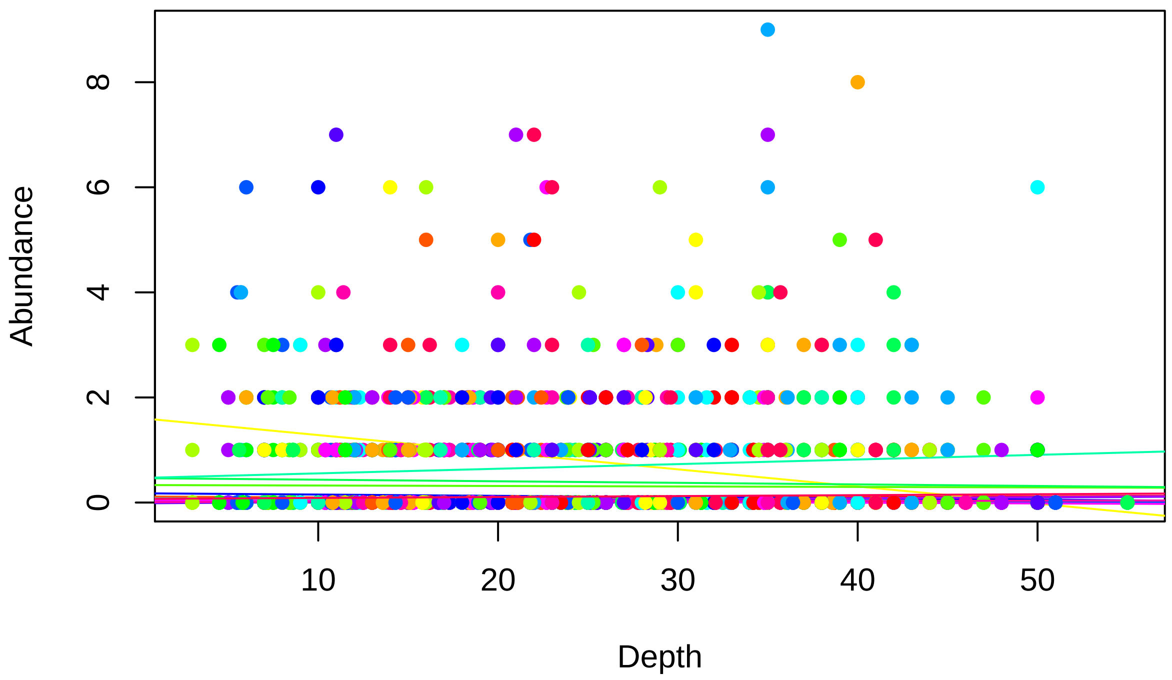


```
##      Achimill Agrostol Airaprae Alopgeni Anthodor Bellpere Bromhord Chenalbu
## [1,]        1        1        1        1        1        1        1        1
## [2,]        2        2        2        2        2        2        2        2
## [3,]        3        3        3        3        3        3        3        3
## [4,]        4        4        4        4        4        4        4        4
##      Cirsarve Comapalu Eleopalu Elymrepe Empenigr Hyporadi Juncarti Juncbufo
## [1,]        1        1        1        1        1        1        1        1
## [2,]        2        2        2        2        2        2        2        2
## [3,]        3        3        3        3        3        3        3        3
## [4,]        4        4        4        4        4        4        4        4
##      Lolipere Planlanc Poaprat Poatriv Ranuflam Rumeacet Sagiproc Salirepe
## [1,]        1        1       1       1        1        1        1        1
## [2,]        2        2       2       2        2        2        2        2
## [3,]        3        3       3       3        3        3        3        3
## [4,]        4        4       4       4        4        4        4        4
##      Scorautu Trifprat Trifrepe Vicilath Bracruta Callcusp
## [1,]        1        1        1        1        1        1
## [2,]        2        2        2        2        2        2
## [3,]        3        3        3        3        3        3
## [4,]        4        4        4        4        4        4
```

For loop to add lines or points to a plot:

Example: shark abundance relative to depth at sites around two marine protected areas in South Africa.

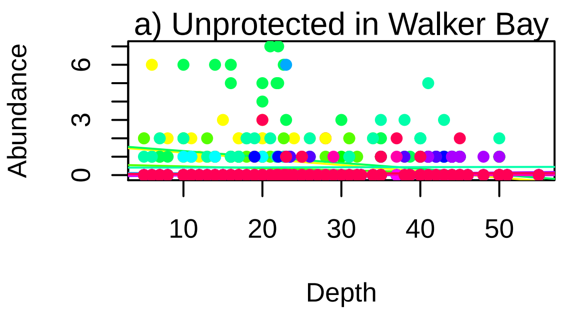
```
#Read in the data:

env <- read.csv("c:/Users/gjosg/Dropbox/R_Handbook_2020_UVic/Data/BRUV_environmentals.csv") #environmen
abundance <- read.csv("c:/Users/gjosg/Dropbox/R_Handbook_2020_UVic/Data/BRUV_abundance.csv") # shark an

sharkCounts <- abundance[,-c(1:4)] #remove extra columns from abundance data so it is only counts

#establish colour palette to pull from, one colour for each shark:
colours <- rainbow(ncol(sharkCounts)) #look up different colour palettes and try different ones

#find the max abundance over all species to set plot's y limit:

max <- max(sharkCounts)

#for loop to plot relationship between each species and depth, as well as the regression line

plot(sharkCounts[,1]~env$Depth, col=colours[1], pch=16, ylab="Abundance", xlab="Depth", ylim=c(0, max))
abline(lm(sharkCounts[,1]~env$Depth), col=colours[1],)

#use for loop for remaining species
for (i in 2:ncol(sharkCounts)) {
  points(sharkCounts[,i]~env$Depth, col=colours[], pch=16, ylab="Abundance", xlab="Depth")#initialize p
  abline(lm(sharkCounts[,i]~env$Depth), col=colours[i],)
}
```

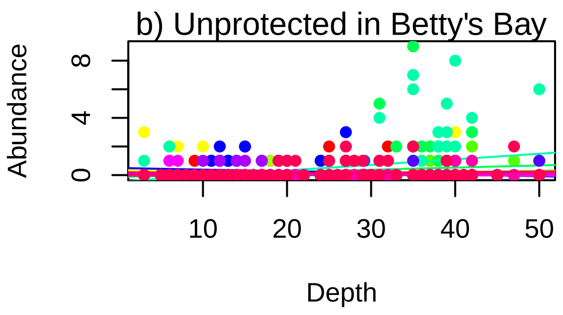


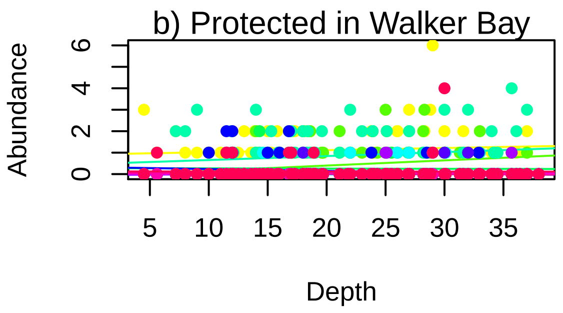
Nested for loops to make multiple plots and add multiple lines to those plots:

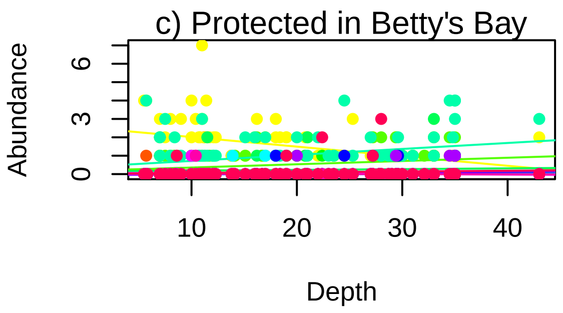
Example: separate plot for each protection level in each area, with different coloured lines and points for each species:

```
#Read in the data:

env <- read.csv("c:/Users/gjosg/Dropbox/R_Handbook_2020_UVic/Data/BRUV_environmentals.csv") #environmen
abundance <- read.csv("c:/Users/gjosg/Dropbox/R_Handbook_2020_UVic/Data/BRUV_abundance.csv") # shark an

sharkCounts <- abundance[,-c(1:4)] #remove extra columns from abundance data so it is only counts

#establish colour palette to pull from, one colour for each shark:
colours <- rainbow(ncol(sharkCounts)) #look up different colour palettes and try different ones

#find the max abundance over all species to set plot's y limit:

#for loop to plot relationship between each species and depth, as well as the regression line, in both 

letters <- c("a)", "b)", "c)", "d)") #set up letters for plot titles
RegionList <- unique(env$Region) # get unique regions
ProtectionList <- unique(env$Protection) #get unique protection levels


par(mfrow=c(2,2)) #set up place to put the plot

for (i in seq_along(RegionList)) {# each unique region
  for (j in seq_along(ProtectionList)) { #each unique protection level
    temp.env <- subset(env, Region == RegionList[i] & Protection == ProtectionList[j]) #subset environm
    temp <- subset(sharkCounts, env$Region == RegionList[i] & env$Protection == ProtectionList[j])
    max <- max(temp)

    plot(temp[,1]~temp.env$Depth, col=colours[1], pch=16, ylab="Abundance", xlab="Depth", ylim=c(0, max
    abline(lm(temp[,1]~temp.env$Depth), col=colours[1])
    mtext(side=3, paste(letters[(i+j-1)], ProtectionList[i], "in", RegionList[j], sep=" ")) #title for

    #use for loop for remaining species
    for (k in 2:ncol(temp)) {
      points(temp[,k]~temp.env$Depth, col=colours[k], pch=16, ylab="Abundance", xlab="Depth") #initiali
      abline(lm(temp[,k]~temp.env$Depth), col=colours[k],)
    }

  }

}
```


a) Unprotected in Walker Bay
Abundance
0
3
6
10
20
30
40
50
Depth


b) Unprotected in Betty's Bay
Abundance
0
4
8
10
20
30
40
50
Depth


b) Protected in Walker Bay
Abundance
0
2
4
6
5
10
15
20
25
30
35
Depth


c) Protected in Betty's Bay
Abundance
0
3
6
10
20
30
40
Depth