R Code

Joseph M. Smith
Last updated: 1-31-2017

Other resources:

Quick-R: http://www.statmethods.net/
UCLA IDRE: http://www.ats.ucla.edu/stat/r/

R Workspace

```
#Use in the GUI not Rstudio
library(installr)
updateR() #Updates R
installed.packages() #See what packages are installed
update.packages("package") #update packages
```

Clear workspace

```
rm(list=ls()) #this clears the workspace
```

Set Working Directory

```
setwd("C:/Documents and Settings/Desktop/Workspace/Toby")
```

Get working directory

```
getwd()#Show working directory
```

Check version of R being used

```
sessionInfo()#Show what version of R is being used
```

Install packages

```
install.packages("package_name") #install a package
```

Commenting out code

To comment out a large block of code highlight code then ctrl+shift+c

Manipulating data in R

Loading data

Data from a .csv file

```
data<-read.csv(file=file.choose(),header=TRUE) #Choose file from
popup window
data<-read.csv('datafile.csv',header=TRUE)
data <- read.csv("datafile.csv", row.names=1) #data with the
first row as ID

#Importing multiple csv files into one dataframe
filenames <- dir() #Get the file names in the directory
all.data<-do.call("rbind",
     lapply(filenames[2:25], #Pick which file names to import
     read.csv,
     header = TRUE))
head(all.data)

#Makes each column an object so you don't need to use $
attach(all.data)
```

Loading data from Access

```
#Data from an Access database
#Must use 32-bit version of R
library(RODBC)

channel1<-
odbcConnectAccess("Stationary_Receiver_Data_for_r.mdb",
uid = "", pwd = "")
MyData <- sqlFetch(channel1, "data_for_r")
close(channel1)

head(data) #Shows first few rows of the data
```

Make and remove objects from workspace

```
x<-1:10 #creates a vector from 1 to 10
remove(x) #removes object from workspace
```

Creating a sequence

```
#creates a vector from 0 to 1 with intervals of 0.05
seq(0,1,.05)
```

Dimensions of matrix and length of vetor

```
dim(data) #number of rows and columns in matrix
```

```
length(x) #number of values in a vector
```

Change column names of dataframe

```
#Change column names
colnames(df)[1]<- "new.name" #changes the first column name
```

Lagged Differences

```
df$x #vector to get lagged differences from
c(0, diff(df$x)) #Creates a vector that starts with zero then
                 #gives the difference between rows of df$x
```

Concatenate Strings

```
#Combine text from two columns
df$combined <- paste(df$x1, df$x2)
```

Merge dataframes

```
#Merge dataframes
merge.df <- merge(df1, df2, by = "merge.variable")
```

Remove Duplicates

```
#Other resources
```
Cookbook for R – Finding and removing dulicate records

```
#Take away duplicates
#removes rows with columns that all have the same values
no.dups.df<-unique(df)

#Subset dataframe to exclude duplicated values within one column
#creates a new column with TRUE or FALSE for duplicates of x
df$x.dup<-duplicated(df$x)
#Create new dataframe with dups removed
df.xdups.removed<-subset(df,df$x.dup==FALSE)
```

Copy object to the clipboard

```
writeClipboard(x) #Copy data to the clipboard

#Copies data then paste in excel
write.table(df,"clipboard",sep="\t",col.names=NA)
```

Write CSV Table

```
write.table(df,file= "filename.csv" ,sep=',', row.names =TRUE,
col.names =TRUE)
write.csv(df,file="filename.csv")
```

Identify data type

```
#shows the structure of the data (e.g., numeric, factor)
str(data)
```

Convert data type

```
#convert to numeric
numeric.x <- as.numeric(x)

#convert to text string
character.x <- as.character(x)

#convert to logical
logical.x <- as.logical(x)

#create factor and label factors, nominal (factor where order
doesn't matter)
factor.x <- factor(x, labels = ("label1", "label2"))

#create an ordinal variable (factor where the order matters)
ordinal.x <- ordered(df$factor, levels =
c("low_traffic","medium_traffic","high_traffic")
```

For more details on factors:
Cookbook for R – Changing the order of levels of a factor

Create training and test datasets

Other resources:
Split a dataframe into testing and training sets in R

```
# splitdf function will return a list of training and testing sets
splitdf <- function(dataframe, seed=NULL) {
     if (!is.null(seed)) set.seed(seed)
     index <- 1:nrow(dataframe)
     trainindex <- sample(index, trunc(length(index)/2))
     trainset <- dataframe[trainindex, ]
     testset <- dataframe[-trainindex, ]
     list(trainset=trainset,testset=testset)
}

#apply the function
splits <- splitdf(iris, seed=808)

#it returns a list - two data frames called trainset and testset
str(splits)

# there are 75 observations in each data frame
lapply(splits,nrow)
```

```
#view the first few columns in each data frame
lapply(splits,head)

# save the training and testing sets as data frames
training <- splits$trainset
testing <- splits$testset
```

Standardize and Transform data

```
#Standardize
x.scale<-scale(x) #Standardize data mean=0
library(vegan); x.stand<-decostand(x, "standardize")

#Presence-absence
x.pa<-x[x>0]<-1 #transform abundance to presence absence
library(vegan); x.pa<-decostand(x, "pa") #presence/absence with
vegan package

#Arcsin squareroot
asin.x<-asin(sqrt(x)) #Acrsin squareroot transformation for
proportion data
```

Subset Data

```
df.subset<-subset(df,
     df$column.name == 'value.to.subset.by')
df.subset<-df[df$x== "a"|
     df$x== "b",] #selects rows where x = "a" or "b"
```

Dates and times

Method 1 – Formatted time

```
#Format datetime correctly in excel first.
#yyyy-mm-dd hh:mm:ss
#Save the excel file with this format as .xlsx
#After it is saved as an .xlsx with proper formatting then save
as .csv
#Note: if you format while it is .csv it will not be saved

# 1. Check to see how your date.time variable was imported to R
(usually comes in as a factor)
str(df$date.time)

#2 Convert to character then POSIXct with time zone information

df$date.time.gmt<- as.POSIXct(as.character(df$date.time),
     tz = "GMT")
df$date.time.gmt[1] #check that GMT is in the time
```

```
#3 Convert to another time zone

#List of timezones
Wikipedia list
#Availabe timezones in R
OlsonNames()

#Determine the current time zone for your computer
Sys.timezone()

#Add or subtract hours from GMT to NOT INCLUDE daylight savings
     #time
#Convert from GMT to Eastern Standard Time
df$date.time.EST<- df$date.time.gmt-3600*5 #Subtract 6 hours

#Convert from GMT to Central Standard Time
df$date.time.CST<- df$date.time.gmt-3600*6 #Subtract 6 hours

#Convert from GMT to Pacific Standard Time
df$date.time.PST<- df$date.time.gmt-3600*8 #Subtract 8 hours

#This example converts to Eastern time INCLUDING daylight
savings time
df$date.time.eastern<- as.POSIXct(format(df$date.time.gmt,
     tz = "US/Eastern",usetz = TRUE) ,tz="US/Eastern")
df$date.time.eastern[1]

#This example converts to Central time INCLUDING daylight
savings time
df$date.time.central<- as.POSIXct(format(df$date.time.gmt,
     tz = "US/Central",usetz = TRUE) ,tz="US/Central")
df$date.time.central[1]

#This example converts to Pacific time INCLUDING daylight
savings time
df.v$date.time.pacific<- as.POSIXct(format(df$date.time.gmt,
     tz = "US/Pacific",usetz = TRUE),tz="US/Pacific")
df.v$date.time.pacific[1]
```

Method 2 – Unformatted times

```
# PROCESS FOR GETTING DATES AND TIMES INTO R
# 1. Check to see how your date.time variable was imported to R
(usually comes in as a factor)
str(df$date.time)
```

```
# 2. Change date.time variable to a character
df$date.time<-as.character(df$date.time)
str(df$date.time) #check that it worked

# 3. Determine the format of your date.time. Usually in Excel it
     #looks like month/day/year hour:min:sec which translates to
     # format="%m/%d/%Y %H:%M:%S" in R.

# 4. Use strptime to convert the date.time from a character to
     #POSIXlt
df$date.time<-strptime(df$date_time,format="%m/%d/%Y %H:%M:%S")
#convert character to POSIXlt
str(df$date.time) #check that it worked

# 5. Convert to POSIXct which is easier to use when plotting
     #(see axis.POSIXct in Plotting)
df$date.time<-as.POSIXct(df$date.time)
str(df$date.time) #check that it worked

# 6. If you bring the date time in as YYYY-MM-DD HH:MM:SS
     #you can skip strptime. This can go straight from a
     #character to POSIXct
```

Convert date.time to date, month, day, hour

```
#with lubridate package
library(lubridate)
df$date<-floor_date(df$date.time, "day") #datetime to only date
df$date<-as.date(df$date.time) #Another way to go from datetime
                               # to date
df$month<-month(df$date.time)#datetime to month
#Julian day
df$julian.day<-as.numeric(strftime(df$date,format = "%j"))
df$day<-day(df$date.time)#datetime to day
df$hour<-hour(df$date.time)#datetime to hour
```

Hour fraction

```
df$hour.fraction<-as.numeric(format(df$date.time, "%H")) +
(as.numeric(format(df$date.time, "%M")) / 60)
```

Sorting data

```
#Sort df
df.sort<-df[order(df$x1, df$x2),] #don't forget the comma at the
                                  # end

#sort vector
sort(df$x)
```

Reorder columns

```
df.order<-df[,c(5,9,8,1,2,10,6,7,3,4,11)]
```

Insert values to a vector at certain positions

```
library(R.utils)
x0 <- c(1:4, 8:11, 13:15)
x0
x <- insert(x0, at=c(5,9), values=list(5:7,12))
x
```

Calculating row or column sums and means

```
colSums (x, na.rm = FALSE, dims = 1)
rowSums (x, na.rm = FALSE, dims = 1)
colMeans(x, na.rm = FALSE, dims = 1)
rowMeans(x, na.rm = FALSE, dims = 1)
```

Calculate Standard Error

```
stderr <- function(x) sqrt(var(x)/length(x))
stderr2 <- function(x) (sqrt(var(x)/length(x))*2)
```

Species calculations

```
#Species Richness
library(vegan)
(R<-specnumber(x))

#Species Richness by grouping variable
library(vegan)
(R.group<-specnumber(x,groups=grp.variable))

#Shannon's H' Diversity
(H<- diversity(x, index = "shannon"))
#Simpson's Diversity
(Simp<- diversity(x, index = "simpson"))
```

tapply and aggregate

```
#Means of fish depth for each SN
tapply(df$FishDepthM, #variable to be summarized
     df$SN, #Grouping variable
     mean) #function to apply

ag<-aggregate(y ~ x1 + x2 + x3, #variable ~ Groups
     data = df, #Dataset
     mean) #Function
```

```
xtabs(FishDepthM~.,data=ag) #Easier to read format
```

Bin a numeric vector

```
df$bin.depth<-.bincode(df$depth,seq(0,20,1)) #Create bins from 0
to 20 by 1
table(df$bin.depth)
```

Create a categorical variable based on a continuous variable

Cookbook for R – Recoding data

```
df$category<- cut(df$x, #continuous variable
                  breaks= c(-Inf,3000,6000,Inf), #cutoffs <3000,
                                    #>3000 and <6000, >6000
                  labels = c("short","medium","long")) #labels
for each category
```

Creating a vector based on values of another vector

```
df$symb<-df$x
df$symb<-as.character(symb) #makes object character
df$symb [df$trt == "A"]<-1 #changes text "A" to number 1
df$symb [df$trt == "B"]<-16 #changes text "B" to number 16
df$symb<-as.numeric(df$symb) #makes object numeric
df$symb
```

Find values in a dataframe/vector that are not in another dataframe/vector

```
df3<-df2[!(df2[,1] %in% df1[,1])& !(df2[,2]%in% df1[,2]),]
```

Combining columns and rows

```
cbind.data.frame(x1,x2,x3)
cbind(x1,x2)
rbind(x1,x2)
paste(x1,x2,sep="") #concatenate two text variables into one
```

Remove species that are in less than 5% of sites

```
Source("biostats.R")
y.abund.rm<-drop.var(y.abund,min.po=5) #need Biostats
dim(y.abund.rm)
```

Flow control in R – Loops

From Handel 2012 – Yet another R introduction a self-help guide

When you write longer programs, you sometimes want to execute or not execute certain com-
mands, depending on the current status of things. For instance you might want to do one thing
if a>b and another if a<b. In addition, you might often want to do the same thing many times.
To get your program to do such things, i.e. to control the flow of your code, R has several basic
ways of achieving this. We will briefly discuss them now.

For-loops

Loops make it easy to do the same operation over and over again. A for loop runs for a specified
number of steps and is written as

```
for (var in seq)
{
     commands
}
```

Here's an example. Note that it might be a good idea to write this example (and the ones that
follow below) into an R script, give it a filename, and then run it with source.

```
#create vector of length 10 filled with zeros. Set first element to 1.
popsize=rep(0,10); popsize[1]=1;

# calculate population size at times 2 through 10, write to Command Window
for (n in 2:10)
{
     popsize[n]=2*popsize[n-1];
     x=log(popsize[n]);
     cat(n,x,"\n");
}
plot(1:10,popsize,type="l");
```

The first time through the loop, n=2. The second time through, n=3. When it reaches n=10,
the for-loop is finished and R starts executing any commands that occur after the end of the
loop. The result is a table of the log population size in generations 2 through 10.

Several for loops can be nested within each other, which is sometimes useful for working with
matrices as in the example below. It is important to notice that the second loop is completely
within the first. Loops must be either nested (one completely inside the other) or sequential
(one starts after the previous one ends).

```
A=matrix(0,3,3);
for (row in 1:3)
{
for (col in 1:3)
     {
          A[row,col]=row*col
     }
}
```

Type this into a script file and run it; It seems like nothing happens, but if you now type A into
the console, you should see

```
[,1] [,2] [,3]
[1,] 1 2 3
[2,] 2 4 6
[3,] 3 6 9
```

Line 1 creates A as a matrix of all zeros - this is an easy way to create a matrix of whatever
size you need, which can then be filled in with meaningful values as your program runs. Then
two nested loops are used to fill in the entries. Line 2 starts a loop over the rows of A, and
immediately in line 3 a loop over the columns is started. To fill in the matrix we need to
consider all possible values for the pair (row, col). So for row=1, we need to consider col=1,2,3.
Then for row=2 we also need to consider col=1,2,3, and the same for row=3. That’s what the
nested for-loops accomplish. For row=1 (as requested in line 2), the loop in lines 3-5 is executed
until it ends. Then we get to the end in line 6, at which point the loop in line 2 moves on to
row=2, and so on.

Also note in the above example how the code is written, with opening and closing parentheses

in the same position. This is not strictly necessary, but writing code like this helps you to quickly see which blocks of statements belong together and makes reading and programming much easier.

If this discussion of looping doesn't make sense to you, stop now and get help. Loops are an essential component of programming in R or other languages. In fact, the "getting help" applies to any topic we have covered so far and will continue to cover. Make sure you are comfortable with everything explained here and that you fully understand it. If not, keep practicing and asking for clarification.

While-loops

A while loop lets an iteration continue until some condition is satisfied. For example, we can solve a model until some variable reaches a threshold. The format is

```
while(condition)
{
     commands
}
```

The loop repeats as long as the condition remains true. Write the following lines into an R script and run them:

```
n=1
while (n<10)
{
     n=2*n
}
print(sprintf("n=%d",n))
```

Note the double-command print(sprintf()) which has a smiliar effect to cat used above, but differs in the details. Learn about it by reading the help file. Move the print(sprintf()) command into the loop and run it again. What would happen if you replace the statement

inside the loop with n = n2? If you don't know, try it. And you might want to revisit the
discussion about how to interrupt R...

Within a while-loop it is often helpful to have a counter variable that keeps track of how many
times the loop has been executed. In the following code, the counter variable is ct:

```
ct=1;
while(condition)
{
     commands
     ct=ct+1;
}
```

The result is that ct=1 is true while the commands (whatever they are) are being executed for
the first time. Afterward ct is set to 2, and this remains true during the second time that
the commands are executed, and so on. One use of counters is to store a series of results in a
vector or matrix: on the ctth time through the commands, put the results in the ctth entry of
the vector, ctth row of the matrix, etc.

Branching

Logical conditions also allow the rules for "what happens next" in a model to depend on the
current values of state variables. The if statement lets us do this; the basic format is

```
if(condition)
{
     some commands
}
else
{
     some other commands
}
```

More complicated decisions can be built up by nesting one if block within another, i.e. the
"other commands" under else can include a second if block. Here is an example where a

population grows, and the growth tails off in several steps as the population size increases.

```
rm(list=ls()) #this clears the workspace
graphics.off(); #close all graphics windows
popnow=10;
popsize=popnow;
for (i in 1:50)
{
     if(popnow<250)
     {
          popnow=popnow*2;
     }
     else
     {
          if(popnow<500)
          {
               popnow=popnow*1.5
          }
          else
          {
               popnow=popnow*0.95
          }
     }
     popsize=c(popsize,popnow);
} #this ends the for loop
plot(popsize,type="b") #plots with both lines and symbols
```

What does this accomplish?

• If popnow is still < 250, then growth by a factor of 2 occurs. Since the if condition was satisfied, the entire else block isn't looked at.
• If popnow is not < 250, R moves on to the else statement, and immediately encounters another if.
• If popnow is < 500 the growth factor of 1.5 applies.
• If neither of the two if conditions is satisfied, the final else block is executed and population declines by 5% instead of growing.

The final command inside the for-loop adds the current population size, popnow, to the vector popsize, which therefore keeps growing. For a vector or matrix to grow like this, the variable needs to exist. Test this by removing the popsize=popnow command and run it again. In

general, these growing arrays make the code run slower compared to creating an initial array of
all-zeros and filling it. But sometimes you don't know how many numbers your code produces,
and then these growing arrays can be useful.

Also note the comment behind the last }. Such comments can be very useful if you have long
blocks of code that stretch over pages and you can't see where the closing parentheses belong
to.

Plotting

Margins

```
par(mar=c(2,3,2,1), #c(bottom,left,top,right)
                    # default is (5,4,4,2)+0.1
     mgp=c(4,1,0)) #Axis label distance default mgp=c(3, 1, 0)
```

Multipanal

```
par(mfrow=c(1,2)) #c(rows,columns)
dev.off() #Back to one plot
```

Points plot

```
plot(Y~X, #formula
     type= "p", #plots points
     pch=16, #symbol
     cex = 2, #size of symbols
     xlim = c(0,10), ylim=c(0,10), #axis limits
     xaxp = c(0,10,5), yaxp=c(0,10,5), #extreme coordinates and
                          #number of intervals between tick-marks
     cex.axis = 2, #size of axis numbers
     xlab = "X label", ylab = "Y label", #axis labels
     cex.lab = 2, #size of axis labels
     bty = "n", # no box around plot
     col = "blue", #color of symbols or lines
     las = 1) #axis labels horizontal
```

Identify points

```
identify(X, Y) # identify points by clicking on the map
```

Add a lm line to plot

```
fit<-lm(Y~X)
#add straight line,
abline(fit,
     lwd=1, #line width
```

```
     lty=1) #line type
```

Add loess line

```
lines(loess.smooth(X, Y, col= "blue") #Loess line
```

Smooth line plot

```
scatter.smooth(Y~X) #add smooth curve
```

Add shaded area to plot

```
rect(xmin,ymin,xmax,ymax,col="grey")

#With transparent color
rect(xmin,ymin,xmax,ymax,col=rgb(0,0,0,alpha=0.3)
```

Connect points with line

```
lines(Y~X) #add connecting lines
```

Add horizontal or vertical lines to plot

```
abline(h=100) #Add horizontal line at y=100
abline(v=100) #Add vertical line at x=100
```

Special characters

```
Temperature degree ° = Alt + 0176

Parts per thousand ‰ = Alt + 0137

Micro µ = Alt + 0181

#adding an italic letter
ylab= expression(Chlorophyll~italic(a)~(µg/l))
```

Plotting colors

RGB Color Codes Chart

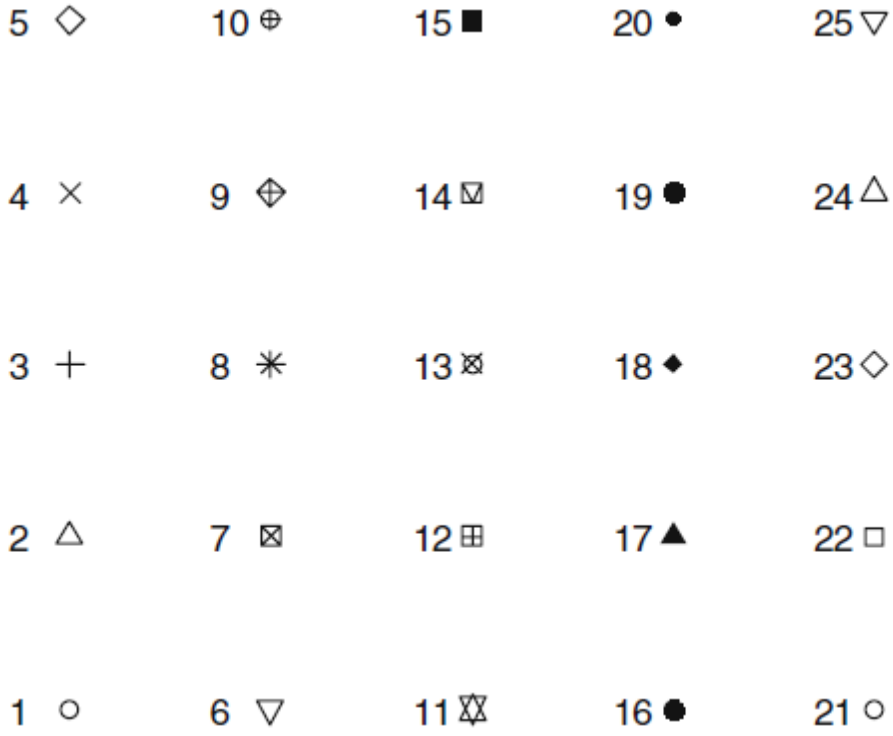
Color Palettes

```
col.rainbow<-rainbow(5) #Create a vector of 5 different colors
col.heat<-heat.colors(5)
col.terrain<-terrain.colors(5)
col.topo<-topo.colors(5)
co.cm<-cm.colors(5)
```

```
gray.colors(n, start = 0.3, end = 0.9, gamma = 2.2, alpha =
NULL)
```

Plotting symbols

pch=



Text symbols

```
plot(y~x, type= "n") #make a plot without points
text(y~x, labels=label.variable)
```

Axes

Box around plot

```
box()
```

Turn of axes

```
plot(…,axes=FALSE) #turn off axes in the plot
```

X axis

```
axis(1, #1 = x axis
     at=c(0,1,2,3,4), #where to put labels
     pos=c(0,0),#where to start the axis
```

```
     cex.axis=1.5, #size of axis labels
     labels=c('','M','M + E','M + E + H','')) #labels
```

Y axis

```
ydig<- c(0,2,4,6,8,10,12)
axis(2, #y axis
     at=ydig, #where to put ticks
     labels=round(ydig,digits=2), #labels at ticks with
                                  #significant digits of 2
     cex.axis=1.5) #y axis
```

axis.Date

```
#Determine the first and last date in data
r <- as.Date(round(range(date.variable)))
axis.POSIXct(1, #which axis
     at = seq(r[1], r[2], by = "month"), #How often to have tick
marks
     format = "%m-%Y") #Format of tick mark labels
```

Error bars

```
plot(y~x) #Plots data

#y error bars
arrows(x,y-ybar, #plots lower error bar (ybar = standard error
or 95% CI)
       x,y+ybar, #plots upper error bar
       code=3, angle=90, length=0.1) #error bar specifics
```

Barplot

with error bars

```
y #barplot value
se #standard error (or other value for error bars)

bp<-barplot(y,las=1) #save barplot as an object
#add the lines
segments(bp, y - se, bp, y + se, lwd=2,col="black")
segments(bp - 0.1, y - se, bp + 0.1, y - se, lwd=2,col="black")
#add lower t on bar
segments(bp - 0.1, y + se, bp + 0.1, y + se, lwd=2,col="black")
#add upper t on bar
```

Bar plot with sciplot

```
library(sciplot)
```

```
bargraph.CI(
  df1$Habitat,  #categorical factor for the x-axis
  df1$Richness,    #numerical variable for the y-axis
  legend=T,
  x.leg=19,
  ylab="Richness",
  xlab="Habitat",
  fun = function(x) mean(x, na.rm=TRUE), #plot the mean
  ci.fun= function(x) c(mean(x)-2*se(x), mean(x)+2*se(x)), #2 SE
error bars
  cex.lab=1.0, cex.axis=1.0, ylim=c(0, 10))
```

Stacked barplot

```
counts <- table(mtcars$vs, mtcars$gear)
counts
barplot(counts, main="Car Distribution by Gears and VS",
        xlab="Number of Gears", col=c("darkblue","red"),
        legend = rownames(counts))
```

Legend

```
legend.text<-c("Dam upstream","Dam difference","Dam
downstream","Control upstream", "Control difference","Control
downstream") #Create text for legend

legend("topright", #where to put legend
     legend = legend.text, #legend text
     col = c(1,1,1,2,2,2), #colors
     pch = c(3,16,24,3,16,24), #symbols
     bty= "n") #remove box

#LEGEND FOR LINES
legend.text<-c("Upstream","Downstream")
legend("topleft",legend = legend.text,col =
c('grey','black'),lwd=3,bty='n')
```

Plot with two Y axes

```
x <- 1:5
y1 <- rnorm(5)
y2 <- rnorm(5,20)
par(mar=c(5,4,4,5)+.1)
plot(x,y1,type="l",col="red")
par(new=TRUE)
plot(x,
y2,,type="l",col="blue",xaxt="n",yaxt="n",xlab="",ylab="")
axis(4)
mtext("y2",side=4,line=3)
legend("topleft",col=c("red","blue"),lty=1,legend=c("y1","y2"))
```

ggplot

```
library(ggplot2)
#Plot means with 95% CI
ggplot(df.dec, aes(x=hour, y=prop.depth, colour=species)) +
  geom_errorbar(aes(ymin=prop.depth-ci, ymax=prop.depth+ci),
colour="black", width=.3) +
  geom_line() +
  geom_point( size=5) + # 21 is filled circle
  xlab("Hour") +
  ylab("Depth proportion (95% CI)") +
  scale_colour_hue(name="Species", # Legend label, use darker
colors
                   breaks=c("Chinook", "coho"),
                   labels=c("Chinook", "Coho"),
                   l=40) + # Use darker colors, lightness=40
  ggtitle("December\nFish depth proportional to max water
depth\nin Puget Sound") +
  scale_y_continuous(limits=c(-1, 0.1))  +  #Set y range
  theme_bw() +
  theme(legend.justification=c(1,0),
     legend.position=c(1,0)) #Position legend in bottom right
```

3D plot

STHDA – 3D plots

```
#Example of 3D fish movement plot
library(scatterplot3d) #Load library

#Save as high resolution plot
png("F1 STB PIE 2015.png", # file name
    width=11, height=7, #width and height of plot in inches
    units="in", #units = inches
    res=600) #resolution in dpi
#Create 3D plot outline (no points in the plot yet)
p<-scatterplot3d(d$Station.Longitude, #x value (longitude)
                 d$date.time.est, #y value (datetime)
                 d$Station.Latitude, #z value (latitude)
                 type='n', #plot type, n = don't plot points
                 pch=16, #point symbols
                 box=FALSE, #Remove box around the plot
                 grid=F, #Remove grid from plot
                 tick.marks = F, #remove tick marks
                 xlab='', #x label
                 ylab='', #y label
                 zlab='') #z label
#Add points and lines to plot
```

```
p$points3d(f1$Longitude,f1$date.time.est,f1$Latitude,type='p',co
l = f1$month.col,pch = 16) #Add pointsto plot
p$points3d(f1$Longitude,f1$date.time.est,f1$Latitude,type='l')
#Add lines to plot
p$points3d(r$Longitude,r$mindatetime,r$Latitude,col=
"gray",cex=1,pch = 16) #Add 2D Receiver locations
#Add text labels for 2D receiver locations
s3d.coords <- p$xyz.convert(r$Longitude, r$mindatetime,
r$Latitude) #convert to 2D
text(s3d.coords$x, s3d.coords$y,labels =
r$Receiver.Name,cex=.5,pos = 1) #add text labels
#Add legend
legend("topright", #where to put legend
       legend = legend.text, #legend text
       col = c("#FF0000FF","#CCFF00FF","#00FF66FF",
               "#0066FFFF","#CC00FFFF"), #colors
       pch = 16, #symbols
       cex= 2, #symbol size
       bty= "n") #remove box
dev.off() #End high resolution plot
```

3D barplot

StatMethods blog – 3D

```
s3d.tot<-
scatterplot3d(res.data$Longitude,res.data$Latitude,res.data$sum.
res,
              zlim=c(0,100), #Change the 100 to whatever your
max residence time, use the same zlim for all plots
              grid=F, #Turn off grid
              axis=F, #Turn off axis
              pch=16, #symbol
              angle=90, #Change the angle of the plot
              box=F, #Turn off box
              lwd=7, #Width of line
              type="h") #type of plot

#create label coordinates
s3d.coords.tot <-
s3d.tot$xyz.convert(res.data$Longitude,res.data$Latitude,res.dat
a$sum.res) # convert 3D coords to 2D projection
# residence time labels
text(s3d.coords.tot$x,
s3d.coords.tot$y,labels=round(res.data$sum.res,2),cex=.5, pos=3)
# station labels
text(s3d.coords.tot$x,
s3d.coords.tot$y,labels=res.data$STATIONNAME,cex=.5, pos=3)
```

Maps

Puget Sound Maps

```
library(PBSmapping) #powerful mapping functions developed by
                    # Pacific Biological Station
data(nepacLLhigh)
plotMap(nepacLLhigh, xlim=c(-125, -121.9),
     ylim=c(47,48.9),las=1,col="gray",bg="white")
edata<-as.EventData(p.data)#convert lat longs to EventData
edata.all<-as.EventData(all.d) #convert lat longs to EventData

addPoints(edata.all,pch=16) #Add points to the plot
addPoints(edata,col="red",pch=16) #Add points to the plot
```

Adding Scale bar

```
library(maps)
map.scale(-122.5,relwidth=.2,metric=T,ratio=F,cex=.75)
```

Maps from shapefile

```
library(maptools)
milford <- readShapePoly("Milford")
plot(milford, axes=TRUE, border="gray",las=1,col="skyblue")
```

Google earth type maps

```
library(OpenStreetMap) #Need to use 32-bit version of R

#Get a map based on lat and long coordinates
#Example for Head of Old River
map <- openmap(c(37.813,-121.33), # Upper left coordinates
               c(37.805,-121.32), # Lower right coordinates
             type="bing") # Type of map to get

#Projects the open street map in an alternate coordinate system
map_longlat <- openproj(map, projection = "+proj=longlat")

#Plots the map
plot(map_longlat)

#Add points
points(df$lat~df$long)

#Add lines/tracks
Lines(df$lat~df$long)
```

High resolution plot

```
png("desired file name.png", # file name
     width=5, height=5, #width and height of plot in inches
     units="in", #units = inches
     res=600) #resolution in dpi
Plot code #Run the plotting code here
dev.off() #turn off plot
```

Distribution analysis

Chi-Squared tests

```
#Chi-squared test with Monte Carlo p-value with 2000 replicates
chisq.test(X,simulate.p.value=TRUE, B=2000)
E<-chisq.test(sum.bass.stand,simulate.p.value=TRUE,
     B=2000)$expected #Expected values
O<-chisq.test(sum.bass.stand,simulate.p.value=TRUE,
     B=2000)$observed #Observed values
O-E #difference between observed and expected
(O-E)^2/E #chi-squared values
```

Kernal Density

```
d2<-density(x$lat)
plot(d2)
lines(d2$y~d2$x, col="red", lwd=5)
lines(d$y~d$x)
```

Correlation and Regression

Correlation

```
cor(data)
cor.test(x1,x2)
```

Correlation matrix with scatterplot

```
#Make a correlation matrix with a scatter plot
panel.cor <- function(x, y, digits=2, prefix="", cex.cor)
{
  usr <- par("usr"); on.exit(par(usr))
  par(usr = c(0, 1, 0, 1))
  r = (cor(x, y))
  txt <- format(c(r, 0.123456789), digits=digits)[1]
  txt <- paste(prefix, txt, sep="")
  if(missing(cex.cor)) cex <- 0.8/strwidth(txt)
  text(0.5, 0.5, txt, cex = cex * abs(r))
}
```

```
#Now run this code to make the plot
pairs(data, lower.panel=panel.smooth, upper.panel=panel.cor)
```

Linear regression

The basics

```
fit<-lm(y~x)
summary(fit)
plot(fit) #model diagnostics
plot(y~x) #Scatter plot
abline(fit) #Add straight line to plot
lines(lowess(x,y), col="blue") # Add a smoothed line
a1 <- predict(m1, interval="confidence") # calculate 95% CI
a1<-cbind(a1,x) #add X variable
a1<-a1[order(x),] #Sort by X variable
lines(a1[,4], a1[,2], lty=2) #add lower CI
lines(a1[,4], a1[,3], lty=2) #Add upper CI
```

Model Averege, Variable Importance, Model weights with glmulti

```
#Must have Java installed on your PC
library(rJava) #Have to use 32-bit version of R
library(glmulti)
g.lm<-glmulti(LogAbund~NumGS+MeanInvertebrates+MeanSecchiD,
#calculates all combinations
              data=data, #dataset
              level=1, #1=no interactions 2=include interactions
              crit=aicc, #which AIC to use
              fitfunction=lm) #type of model to run
summary(g.lm,fitting)
weightable(g.lm) #Model weights
coef(g.lm) #Variable importance and averaged Estimates (slopes)
```

Model selection, averaging and pulling top models with MuMIn

```
#Model Selection ####
library(MuMIn)
m.full<-lm(Y ~ het.dta + X2 + X3 + X4) # your full model
options(na.action=na.fail) #Need to change this to fail to
          #dredge, but change back to omit to run models above
ms<-dredge(m.full)
ms #AIC table of all models

#Make a CSV file of model selection table
write.table(modsel.nb, file="HurdleModelSelection.csv", sep=",",
col.names=T)
```

```
#Model Averaging ####
library(MuMIn)
#Model average deltaAIC < 4
(m.ave<-model.avg(ms, subset = delta < 4))
#Summary of averaged top models including variable importance
summary(m.ave)

msaic4<-get.models(ms, subset = delta < 4) #gets the top models
summary(msaic4) #shows the number of the top models top models
summary(msaic4$'6') #Pull out the model info for any model in
  #this case it is model six from which was the first row from
summary(msaic4)
#VIF for top models
library(faraway)#library for VIF
vif(msaic4$'6')
vif(msaic4$'8')
```

Regression diagnostics

Quick-R – Regression Diagnostics

```
library(car)

# Assessing Outliers
outlierTest(fit) # Bonferonni p-value for most extreme obs
qqPlot(fit, main="QQ Plot") #qq plot for studentized resid
leveragePlots(fit) #leverage plots

# Influential Observations
# added variable plots
av.Plots(fit)
# Cook's D plot
# identify D values > 4/(n-k-1)
cutoff <- 4/((nrow(mtcars)-length(fit$coefficients)-2))
plot(fit, which=4, cook.levels=cutoff)
# Influence Plot
influencePlot(fit, id.method="identify", main="Influence Plot",
sub="Circle size is proportial to Cook's Distance")

# Normality of Residuals
# qq plot for studentized resid
qqplot(fit, main="QQ Plot")
# distribution of studentized residuals
library(MASS)
sresid <- studres(fit)
hist(sresid, freq=FALSE, main="Distribution of Studentized
Residuals")
```

```
xfit<-seq(min(sresid),max(sresid),length=40)
yfit<-dnorm(xfit)
lines(xfit, yfit)

# Evaluate homoscedasticity
# non-constant error variance test
ncvTest(fit)
# plot studentized residuals vs. fitted values
spreadLevelPlot(fit)

# Evaluate Collinearity
library(faraway)
vif(fit) #variance inflation factors
sqrt(vif(fit)) > 2 #problem?

#Condition index
library(perturb)
colldiag(fit)

#Rules of thumb according to Quinn and Keough page 128
#VIF > 10 suggest strong collinearity
#Condition index > 30 are collinear
#Condition number < 0.5 are collinear

# Evaluate Nonlinearity
# component + residual plot
crPlots(fit)
# Ceres plots
ceresPlots(fit)

# Test for Autocorrelated Errors
durbinWatsonTest(fit)

# Global test of model assumptions
library(gvlma)
gvmodel <- gvlma(fit)
summary(gvmodel)

#Model Validation ####
plot(m1)

#Check for normality
E<-rstandard(m1) #standardized residuals
hist(E)
qqnorm(E)
```

```
#Check for independence and homogeneity: residuals vs individual
explanatory variables
plot(E~data$Mean_Secchi)
plot(E~data$Abs_Dev_Mean_Depth)
plot(E~data$Categorical_Habitat)
abline(0,0)
```

Logistic regression for binary (presence-absence)

```
data.sort<-data[order(x1.sorting),] #sort first for plotting
Logistic.model<-glm(data.sort$Y ~ data.sort$X,
family='binomial')
Logistic.model
plot(data.sort$Y~ data.sort$X)
lines(Logistic.model$fitted.values ~ data.sort$X) #sorted by X
```

Psuedo R2

```
library(pscl)
pR2(model.object)
```

Poisson regression for count response variables

```
data.sort<-data[order(x1.sorting),] #sort first for plotting
Poisson.model<-glm(data.sort$Y ~ data.sort$X, family='poisson')
Poisson.model
plot(data.sort$Y ~ data.sort$X)
lines(Poisson.model$fitted.values~X) #sorted by X
```

Negative binomial

```
library(MASS)
Negative.binomial.model<-glm.nb (y~x1+x2+x3, link = "log",
     data = dataset)
```

Zero inflated Poisson (ZIP)

```
library(pscl)
f1<-formula(y~x1+x2+x3)
zip<-zeroinfl(f1, dist = "poisson", link = "logit",
     data = dataset)
```

Zero inflated negative binomial (ZINB)

```
library(pscl)
f1<-formula(y~x1+x2+x3)
zinb<-zeroinfl(f1, dist = "negbin", link = "logit",
     data = dataset)
```

Hurdle model

```
library(pscl)
#Create full model formula ####
Global <- formula("pull.float.min ~ temp + light + am.pm +
     offset(log(tot.dist.mod)) |temp + light + am.pm
     + offset(log(tot.dist.mod))")

#Check to see what distribution to use ####
HurdPois <- hurdle(Global, dist = "poisson", link= "logit",
     data=teth.data)
HurdNB <- hurdle(Global, dist = "negbin", link= "logit",
     data=teth.data)
HurdGeo <- hurdle(Global, dist = "geometric", link= "logit",
     data=teth.data)
#Compare using AIC
AIC(HurdPois, HurdNB, HurdGeo)
```

Mixed effect models and correlation structure

```
library(nlme)

http://freshbiostats.wordpress.com/2013/07/28/mixed-models-in-r-
lme4-nlme-both/

model<-lme(FishDepthM~Species*dielperiod,data=data,#forumula

correlation=corExp(form=~date.numeric),#correlation structure
                random=~1|SN)#random slope for each individual

#other correlation structures
#corLin
#corSpher
#corGaus
#corRatio

#Multiple comparisons for interaction
http://stats.stackexchange.com/questions/5250/multiple-
comparisons-on-a-mixed-effects-model

Library(multcomp)
data$inter <- interaction(data$Species, data$dielperiod)
comp.inter <- glht(model, linfct=mcp(inter="Tukey"))
```

GAM

```
#Run GAM model
model<- gam(y~s(x), data = df)
```

```
fit <- predict(model , se = TRUE )$fit
se <- predict(model , se = TRUE)$se.fit
lcl <- fit - 1.96 * se
ucl <- fit + 1.96 * se
#Plot
plot(0 , type = "n" , bty = "n" ,
      xlab = "x label" , ylab = "y label" ,
      main = "" , xlim = c(0,100) , ylim = c(30,50),las=1)
i.for <- order(df$x)
i.back <- order(df$x , decreasing = TRUE)
x.polygon <- c(df$x[i.for] , df$x[i.back])
y.polygon <- c(ucl[i.for] , lcl[i.back])
polygon(x.polygon , y.polygon , col = "#A6CEE3" , border = NA)
lines(bfef$day[i.for] , fit[i.for], col = "#1F78B4" , lwd = 3)
```

Survival Analysis

Cox proportional hazards

Fox and Weisberg 2011

Therneau 1999 – A Package for Survival Analysis in S

Cox regression in R

```
library(survival)

#Fit Cox model
#This example is for a model with time varying covariates
fit.cox <- coxph(Surv(start,end,predation) ~
                 Temp_C +
                 Light_Lux +
                 Ve +
                 Tu +
                 tide_level +
                 time_to_night_mins +
                 depth +
                 dist_shore +
                 Site+
                 cluster(DeploymentID),
                 data = df)
summary(fit.cox) #summary of model

#Check the assumption of proportional hazards of the model
cox.zph(fit.cox)

plot(survfit(fit.cox))#plot survival curve for the model
```

```
#Create an object to use as new data for plotting
#This example uses the mean of all continuous covariates
#Using this as "newdata" will plot a curve for each site.
site.surv <- with(df,
     data.frame(Temp_C = rep(mean(Temp_C),9),
     Light_Lux = rep(mean(Light_Lux), 9),
     Ve = rep(mean(Ve),9),
     Tu = rep(mean(Tu),9),
     tide_level = rep(mean(tide_level),9),
     time_to_night_mins = rep(mean(time_to_night_mins),9),
     depth = rep(mean(depth),9),
     dist_shore = rep(mean(dist_shore),9),
     Site = c( "C1", "C2", "C3", "R1", "R2", "R3", "A1", "A2",
          "A3")))

#Plot showing a line for each site
plot(survfit(fit.cox, #model object
     newdata= site.surv, #new data to plot
     censor = T), #show censored data
     las=1, #turn y axis labels
     col = c("blue","blue","blue","red","red","red","green",
     "green","green"), #colors for lines
     lty = c(1,2,3), #type of lines
     lwd =4, #width of lines
     mark.time = F, #put a cross at each time
     conf.int = F, #show confidence intervals
     cex.axis =2) #font size of axis labels
legend(100,.2, # x and y position of legend box
     legend = c("Control", "Removal", "Addition"),#legend labels
     col = c("blue", "red", "green"), #line colors
     lty = c(1,1,1), #line types
     lwd = 6)#line width
legend(1600,.2,
     legend = c("Block 1", "Block 2", "Block 3"),#legend labels
     col = c("black", "black", "black"), #line colors
     lty = c(1,2,3), #line types
     lwd = 2) #line width
```

Difference among groups

Univariate

Anova

```
fit<-aov(y~grp)
library(car)
Anova(fit, type = 3) #type 3 sum of squares anova table
TukeyHSD(fit)
```

Kruskal-Wallis test

```
kruskal.test(df$y~df$grp)

#MEAN COMPARISONS AFTER KRUSKAL-WALLIS TEST
library(pgirmess)
kruskalmc(df$y~df$grp)
```

Wilcoxon rank sum test

```
wilcox.test(grp.1,grp.2)
```

Boxplots

```
boxplot(y~grp)
```

Multivariate

MRPP (Multi Response Permutation Procedure)

```
library(vegan)
#Create distance matrix of y data
y.bray<-vegdist(y, method= "bray")
y.mrpp<-(y.bray,grp) #grp in a vector containing group identity
y.mrpp #print results

#Plot MRPP monte carlo simulations
with(y.mrpp, {
  fig.dist <- hist(boot.deltas,
xlim=range(c(delta,boot.deltas)),
                   main="Test of Differences Among Groups")
  abline(v=delta,lwd=2,col="red");
  text(delta, 2*mean(fig.dist$counts), adj = -0.5,
       expression(bold(delta)), cex=1.5 )  }
)

#NMDS depicting MRPP
y.ord <- metaMDS(y,distance="bray")

#Plot NMDS, convex hulls, and 95% CI ellipses
```

```
plot(y.ord,display="sites")
ordihull(y.ord, grp,col="red",label=F,show.groups="Pool")
ordihull(y.ord, grp,col="darkgreen",label=F,show.groups="Run")
ordihull(y.ord, grp,col="blue",label=F,show.groups="Riffle")
ordiellipse(y.ord,
grp,col=c("red"),kind="se",conf=0.95,label=T,show.groups="Pool")
ordiellipse(y.ord,
grp,col=c("darkgreen"),kind="se",conf=0.95,label=T,show.groups="
Run")
ordiellipse(y.ord,
grp,col=c("blue"),kind="se",conf=0.95,label=T,show.groups="Riffl
e")

## meandist
y.md <- meandist(vegdist(y,method="bray"), grp)
y.md
summary(y.md)
plot(y.md,las=1)
plot(y.md, kind="histogram")
```

ANOSIM

```
library(vegan)
source('biostats.R')
y.bray<-vegdist(y, method= "bray") #Create distance matrix of y
data

y.anosim<-anosim(y.bray, grp)
summary(y.anosim)
#function from biostats creates boxplots
plot.anosim(y.anosim,las=1)
# The anosim statisticR is based on the difference of mean ranks
# between groups (r_B) and within groups (r_W):
# R = (r_B - r_W)/(N (N-1) / 4)
# The divisor is chosen so that R will be in the interval -1 …
# +1, value 0 indicating completely random grouping.
```

SIMPER

```
library(vegan)
#Contribution of species to bray-curtis differences
y.simper<-simper(y, grp)
y.simper #species that account for at least 70% of dissimilarity
summary(y.simper) #all species
```

Creating distance matrices

Distance matrix

vegdist {vegan}:
http://127.0.0.1:22657/library/vegan/html/vegdist.html # see for details on calculations

```
#see help for other distances
y.eucl<-vegdist(datset ,method='euclidean')
y.eucl

library(cluster)
y.gower<-daisy(x,'gower')
```

Bray-Curtis: *d[jk] = (sum abs(x[ij]-x[ik]))/(sum (x[ij]+x[ik]))*

Gower: *d[jk] = (1/M) sum(abs(x[ij]-x[ik])/(max(x[i])-min(x[i])))*

Gower's similarity coefficient (similarity that can handle missing values)
Mims et al. 2010, "Life history trait diversity of native freshwater fishes in North America"

Gower 1971 "General coefficient of similarity and some of its properties"

Other resources:
Numerical Ecology with R Page 51

Distance matrix for proportion data

```
kibrary(ade4)
dist.prop(y)
```

Cluster analysis

Other resources:
Oksanen. 2014. Cluster Analysis: Tutorial with R

Quick-R: Cluster Analysis

Statistical tools for high-throughput data analysis

Hierarchical Agglomerative Cluster Analysis

```
library(vegan)
library(factoextra)
library(cluster)
library(fpc)

#Dataset
y #dataframe of variables for cluster analysis
y.scale <- scale (y) #Scale variables if measured with different
                     # units
```

Determine number of hierarchical clusters with scree plot

```
#Within-clusters sum of squares method
fviz_nbclust(y.scale, hcut, method = "wss")

#Average silhouette method
fviz_nbclust(y.scale, hcut, method = "silhouette",
             hc_method = "complete")

#Gap statistic method
# Compute gap statistic
set.seed(123)
gap_stat <- clusGap(y.scale, FUN = hcut, K.max = 10, B = 500)
# Plot gap statistic
fviz_gap_stat(gap_stat)
```

Run hierarchical cluster analysis

```
# distance matrix
y.eucl <- vegdist(y.scale, method = "euclidean")
hc <- hclust(y.eucl, method="ward.D") #run cluster analysis
# Other methods include: "ward.D2", "single", "complete",
# "average", "mcquitty", "median", "centroid"
plot(hc,labels = df$labeling.variable) # display dendogram
groups <- cutree(hc, k=5) # cut tree into 5 clusters, clusterID
rect.hclust(hc, k=3, border="red") # draw dendogram with red
                                   # borders around the K
                                   # clusters
```

Examine hierarchical cluster stability

```
#Jaccard similarity value >0.85 = Highly stable,
# >0.75 = valid and stable, >.6 = patterns in the data,
#< 0.6 = not stable
clusterboot(y.eucl, #Distance matrix
            B=500, #number of resampling runs
            Distances=T, #distance matrix = TRUE
            bootmethod=c('boot'),  #Methods used for resampling
            clustermethod=hclustCBI, method="ward.D",
            k=4, #number of clusters
            count=FALSE) #don't show resampling runs
```

K-means cluster analysis

```
library(vegan)
library(factoextra)
library(cluster)
library(fpc)

#Dataset
y #dataframe of variables for cluster analysis
y.scale <- scale (y) #Scale variables if measured with different
                     # units
```

Determine number of k-means clusters with scree plot

```
#Within-clusters sum of squares method
fviz_nbclust(y.scale, kmeans, method = "wss")

#Average silhouette method
fviz_nbclust(y.scale, kmeans, method = "silhouette")

#Gap statistic method
# Compute gap statistic
set.seed(123)
gap_stat <- clusGap(y.scale, FUN = kmeans, K.max = 10, nstart =
                    25, B = 500)
# Plot gap statistic
fviz_gap_stat(gap_stat)
```

Run K-means cluster analysis

```
# distance matrix
y.eucl <- vegdist(y.scale, method = "euclidean")
set.seed(123)
km.clus <- kmeans (y.eucl, #data to cluster
                   3, #number of clusters
                   Nstart = 25) #number of random sets
```

```
km.clus$cluster #Vector of cluster ID

# Visualize k-means clusters
fviz_cluster(km.res, data = y.scale, geom = "point",
             stand = FALSE, frame.type = "norm")
```

Examine k-means cluster stability

```
#Jaccard similarity value >0.85 = Highly stable,
# >0.75 = valid and stable, >.6 = patterns in the data,
#< 0.6 = not stable
clusterboot(y.eucl, #Distance matrix
            B=500, #number of resampling runs
            Distances=T, #distance matrix = TRUE
            bootmethod=c('boot'),  #Methods used for resampling
            clustermethod= kmeansCBI,
            k=4, #number of clusters
            count=FALSE) #don't show resampling runs
```

Non-hierarchical PAM (Partitioning Around Medoids) cluster analysis

```
# use pam for N<200
library(vegan)
library(factoextra)
library(cluster)
library(fpc)

#Dataset
y #dataframe of variables for cluster analysis
y.scale <- scale (y) #Scale variables if measured with different
                     # units
```

Determine number of pam clusters with scree plot

```
#Within-clusters sum of squares method
fviz_nbclust(y.scale, pam, method = "wss")

#Average silhouette method
fviz_nbclust(y.scale, pam, method = "silhouette")

#Average silhouette method print number of clusters
y.eucl <- vegdist(y.scale, method = "euclidean")
pamk.best <- pamk(y.eucl, #distance matrix
     krange = 2:10, #Number of cluster to consider
     diss=T, #Distance matrix = TRUE
     critout = T, #show criterion results
     criterion = "asw") #Type of criterion, large dataset use
```

```
                          #"multiasw" and pam = F,
                          # or use "ch" for Calinski-Harabasz
cat("number of clusters estimated by optimum average silhouette
width:", pamk.best$nc, "\n")

#Gap statistic method
# Compute gap statistic
set.seed(123)
gap_stat <- clusGap(y.scale, FUN = pam, K.max = 10, B = 500)
# Plot gap statistic
fviz_gap_stat(gap_stat)
```

Run PAM cluster analysis

```
y.eucl <- vegdist(y.scale, method = "euclidean")
y.pam<-pam(y.eucl, #distance matrix
     k=4) #number of clusters
summary(y.pam)
plot(y.pam, which.plots = 1) #PCA type plot
plot(y.pam, which.plots = 2) #Silhouette width plot
fviz_cluster(y.pam, stand = FALSE, geom = "point",
             frame.type = "norm")
y.pam$clustering #Cluster groups
```

Visualizing PAM with PCA plot

```
#Plot cluster results in ordination
ord.pca<-rda(y.stand)
plot(ord.pca,display='sites',type='none',las=1,cex.axis=1.5)
points(ord.pca,
     choices=c(1,2), #Axes shown
     'sites', #Site scores
     pch=y.pam$clustering,#different symbols per cluster
     cex=2,lwd=2)
text(ord.pca,
     labels=y.pam$clustering, #Add text ID to symbols
     cex=1, pos=2)
#Identify clusters
ordihull(ord.pca,y.pam$clustering,lty=2,col='red')
ordispider(ord.pca,y.pam$clustering,lty=1,col='black')
#Add vectors
ef<-envfit(ord.pca,y.stand,permu=1000,choices=c(1,2))
plot(ef,col='light blue', p.max=.001)
```

Examine PAM cluster stability

```
library(fpc)
#Jaccard similarity value >0.85 = Highly stable,
#>0.75 = valid and stable, >.6 = patterns in the data,
```

```
#< 0.6 = not stable
clusterboot(y.eucl, #Distance matrix
     B=500, #number of resampling runs
     distances = T, #distance matrix = TRUE
     bootmethod =c('boot'),  #Methods used for resampling
     clustermethod=pamkCBI) #Clustering method
     k=4, #number of clusters
     count=FALSE) #don't show resampling runs on the console
```

Testing significant clusters Calinski-Harabasz pseudo F-statistic

```
library(clusterSim)
index.G1 (y.scale,clusterID) #(Mims et al. 2010)
```

Ordination

NMDS

```
#Scree plot
nmds.scree(fish.comb.bin, distance='bray', k=6, trymax=50,
autotransform=FALSE)

#NMDS
library(vegan)
x.nmds<-
metaMDS(x,distance='bray',k=2,trymax=50,autoransform=FALSE)

#NMDS monte carlo
nmds.monte(x, 2, distance='bray', autotransform=FALSE,perm=100)

stressplot(x)

#Plot NMDS
plot(x.nmds,display='sites',type='n')
text(x.nmds,labels=names.vector)
```

Principal Components Analysis (PCA)

```
# Pricipal Components Analysis
# entering raw data and extracting PCs
# from the correlation matrix
fit <- princomp(mydata, cor=TRUE)
summary(fit) # print variance accounted for
loadings(fit) # pc loadings
plot(fit,type="lines") # scree plot
fit$scores # the principal components
biplot(fit)

#PCA with the rda function
library(vegan)
ord.pca<-rda(y,scale=TRUE)
summary(ord.pca)

ord.prcomp<-prcomp(y,scale=TRUE)
ord.prcomp
summary(ord.prcomp)

pca.eigenvec(ord.prcomp)
pca.structure(ord.prcomp,y,dim=2)
pca.communality(ord.prcomp,y,dim=2)
biplot(ord.prcomp,display='sites')
```

```
plot(ord.pca,display='sites',type='none')
points(ord.pca,choices=c(1,2),'sites',pch=1,cex=1)
text(ord.pca, labels = data.up$ID,cex = 0.5,pos=1)
ordihull(ord.pca,grp,lty=2,col='red')
ordispider(ord.pca,grp,lty=1,col='black')

#Vectors
ef<-envfit(ord.pca,y,permu=1000,choices=c(1,2))
plot(ef,col='light blue', p.max=.001)

# PCA with VARIMAX ROTATION needs 'psych' package
pca.none<-
principal(confl.stand,nfactors=2,rotate='none',scores=TRUE)
pca.varimax<-
principal(confl.stand,nfactors=2,rotate='varimax',scores=TRUE)
pca.varimax
plot(pca.varimax)
pca.varimax$scores
RC1<-pca.varimax$scores[,1]
RC2<-pca.varimax$scores[,2]
plot(RC2~RC1,pch=16, xlim =c(-2,5),col='gray')
ef<-envfit(pca.varimax,confl.data,permu=1000, choices = c(1,2))
ef
plot(ef,p.max=0.05,col='black')
```

Constrained Ordination

```
Canonical Correspondence Analysis

library(vegan)

#Redundancy Analysis
y.rda<-rda(y.matrix~.,condition=condition.matrix,data=x.full,
scale=TRUE)
y.rda

#Significance
anova(y.rda)
anova(y.rda,by='terms')
anova(y.rda,by='axis')

#tri-plot
plot(y.rda,choices=c(1,2), type='none',scaling=3)
points(y.rda,choices=c(1,2),dislplay='sites',pch=type,cex=1,scal
ing=3,label=)
```

```
text(y.rda,choices=c(1,2),display='species',col='red',cex=.75,sc
aling=3)
text(y.rda,choices=c(1,2), display='bp,col='blue')
```

Constrained Ordination Variance Partition

```
library(vegan)
source('biostats.R')

z<-ordi.part(y.matrix,x1.matrix,x2.matrix,method='rda')

#Venn diagram
plot.ordi.part(z,which='total')
plot.ordi.part(z,which='constrained')
```

Indicator species

```
#Dufrene and Legendre 1997 Ecological Monographs
library(labdsv)
library(ade4)
#Example from Numerical Ecology with R page 97
#Doubs fish data
data(doubs)
doubs

spe<-doubs$fish
env<-doubs$env

das.D1<-dist(data.frame(das=env[,1], row.names=rownames(env)))
das.D1

dasD1.kmeans<-kmeans(das.D1, centers=4, nstart=100)
dasD1.kmeans$cluster

(iva<-indval(spe, dasD1.kmeans$cluster))
gr<-iva$maxcls[iva$pval<=0.05]
iv<-iva$indcls[iva$pval<=0.05]
pv<-iva$pval[iva$pval<=0.05]
fr<-apply(spe>0,2,sum)[iva$pval<=0.05]
fidg<-data.frame(group=gr, indval=iv, pvalue=pv, freq=fr)
(fidg<-fidg[order(fidg$group, -fidg$indval),])
```

Spatial autocorrelation

```
# Spatial Autocorrelation Test using Moran's I
library(ape)
site.dists <- as.matrix(dist(cbind(data.up$Long, data.up$Lat)))
site.dists.inv <- 1/site.dists
diag(site.dists.inv) <- 0
```

```
site.dists.inv
Moran.I(y, site.dists.inv)

# OR with

library(spdep)
site.dists <- as.matrix(dist(cbind(data.up$Long, data.up$Lat)))
site.dists.inv <- 1/site.dists
diag(site.dists.inv) <- 0
site.dists.inv
lw <- mat2listw(site.dists.inv)
lwW <- nb2listw(lw$neighbours, glist=lw$weights, style="W")
moran.test(Y,lwW, alternative="two.sided")
moran.mc(Y,lwW,1000)
moran.plot(Y,lwW)
```

CART

For overview see: http://www.statmethods.net/advstats/cart.html
For more details see: https://cran.r-project.org/web/packages/rpart/vignettes/longintro.pdf

```
library(randomForest)
library(tree)
library(rpart)
```

Full Tree

```
#Full tree (most possible splits) with rpart and cartware
y<-y.variable #abundance or presence absence
c.full<-cart(y~., #Formula, cart is a 'cartware.R' function
     data=x.var, #x variables
     method='anova', # 'anova' = regression tree, 'class'=
categorical tree
     parms=list(split='gini'), #Method for node splitting
(calculating node impurity), 'gini' usually prefered
     pick=FALSE, #prune tree with 1-SE rule
     control=rpart.control(minsplit=2,minbucket=1))
     #minsplit=minimum observations in each node,
     minbucket=minimum #observations of child nodes
summary(c.full)
```

Pruned tree

```
#Pruned tree with 1-SE rule
c.prune<-cart(y~., #Formula, cart is a 'cartware.R' function
     data=x.var, #x variables
     method='anova', # 'anova' = regression tree, 'class'=
categorical tree
```

```
     parms=list(split='gini'), #Method for node splitting
(calculating node impurity), 'gini' usually prefered
     smooth=300, # number of times for V-fold cross validation
     pick=TRUE) #prune tree with 1-SE rule
summary(c.prune)
```

Monte Carlo

```
#Monte carlo P-value for pruned tree ####
monte.cart(log.total.abund~Mean_Secchi+Abs_Dev_Mean_Depth+Catego
rical_Habitat, #Formula, only variables in pruned tree
     data=x.grp, #x variables
     method='anova', #regression tree
     parms=list(split='gini'), #gini splits
     n=100, #number of permutations
     size=4, #number of nodes in pruned tree
     ifplot=TRUE) #plot monte carlo results
```

Random forest

```
#Random forest
fit.mean.forest <-randomForest(Y~X1+X2…,data=data.name)
print(fit.mean.forest) # view results
importance(fit.mean.forest) # importance of each predictor
varImpPlot(fit.mean.forest) # importance plot
```

With tree package

```
#Regression tree with 'tree' package
#method='anova' for regression tree
#method='class' for classification tree

fit<-tree(Y~X1+X2…, method='anova', data=data.name)
print(fit)
plot(fit)
text(fit)

#ID which sites are in each node from (tree) function
#vector giving row number of the frame detailing the node
assignment
tree.id<-fit$where
table(node.id) #shows the number of sites in each node

row.id<-1:9 #Create a vector with the number of rows in the
frame
key<-cbind(row.id,fit$frame) #Combine row.id with frame
```

Random Forest Survival Analysis

Ehrlinger, J., and E. H. Blackstone. 2014. ggRandomForests : Survival with Random Forests. R Vignette.

```
library(randomForestSRC)
library(ggRandomForests)
library(ggplot2)
```

Network Analysis

See Auerbach and Poff 2011 JNABS

```
library(igraph)

# http://cran.r-project.org/web/packages/igraph/igraph.pdf
```

Path Analysis

```
library(QuantPsyc)
lm.beta(model) #Standardized beta coefficients
```

Circular Statistics

```
library(circular)
library(lubridate) #creates hour from date time
library(ggplot2)

hour.data #This would be a vector indicating the hour (of
detection or movement or whatever you are testing)
circular.hour <- circular(hour.data%%24,units="hours",
template="clock24") #Converts to circular data

#Test to see if there is unimodal response
rayleigh.test(circular.hour, mu = NULL) #See Chamberlin et al.
2011 (Figure 5)

#Test to see if there is a multimodal response
http://www.pstat.ucsb.edu/faculty/jammalam/html/favorite/test.ht
m
rao.spacing.test(circular.hour)

#Rose diagram (circular package)
library(circular) #load library
#Make circular object
```

```
circular.object<-
circular(hour.data%%24,units="hours",template="clock24")
#Rose Diagram
rose.diag(circular.object, bin = 24, main = "Figure Title", prop
= 2.7)
#Add arrow of mean direction with length rho
arrows.circular(mean(circular.object),y=rho.circular(circular.ob
ject),lwd=3)

#circular plot (ggplot2 package)
ggplot(circular.hour, aes(x = hour, fill = diel.period)) +
  geom_histogram(breaks = seq(0,24), width = 2, colour = "grey")
+
  coord_polar(start = 0) + theme_minimal() +
  scale_fill_brewer(palette="Greys") + #makes gray fill
  ylab("Count") +
  ggtitle("Coho descents (>20 m)") +
  scale_x_continuous("", limits = c(0, 24), breaks = seq(0, 24),
  labels = seq(0,24))
```

Circular GAM

```
#From: http://casoilresource.lawr.ucdavis.edu/blog/modeling-
infants-feeding-schedule-periodic-smoothing-splines/

df$date.time.CT<-as.character(df$date.time.CT)
df$date.time.CT<-as.POSIXct(df$date.time.CT)
df$d<-df$corrected.depth.m
head(df)

df$hour.fraction<-as.numeric(format(df$date.time.CT, "%H")) +
(as.numeric(format(df$date.time.CT, "%M")) / 60)
head(df)

df$date.ct <- as.POSIXct(strptime(df$date, format='%m/%d/%Y' ))


# get the range of our data as a POSIXct object, rounded to
hours
r <- as.POSIXct(round(range(df$date.time.CT[-1], na.rm=TRUE),
"hours"))
r

# generate sequences that we will use later
r.seq <- seq(r[1], r[2], by="12 hours") # for the x-axis of fig.
1
p.seq <- seq(r[1], r[2], by="1 hours") # used for model
predictions in fig. 1
```

```
r.seq
p.seq

# fit a GAM to the time-series
l.ts <- gam(d ~ s(as.numeric(date.time.CT)), data=df)

# fit a GAM to periodic function of hour
l <- gam(d ~ s(hour, bs='cc'), data=df)
summary(l)

# generate predictions from our time-series model
d.ts <- data.frame(date.time.CT=p.seq)
d.ts
p.ts <- predict(l.ts, d.ts)
p.ts
p.ts <- data.frame(d.ts, fit=as.numeric(p.ts))


# generate predictions from our hourly model
d <- data.frame(hour=seq(0, 23, length.out=100))
d
p <- predict(l, d, se.fit=TRUE)
p <- data.frame(d, fit=as.numeric(p$fit),
se.fit=as.numeric(p$se.fit))
head(p)

# estimate 95% CI from standard error
p$upper <- p$fit + 1.96*p$se.fit
p$lower <- p$fit - 1.96*p$se.fit

# generate hourly predictions for use in fig. 1
p.ts.hourly <- predict(l,
data.frame(hour=as.numeric(format(p.seq, "%H"))))
p.ts.hourly

# combine time-series model with hourly model predictions for
fig. 1
p.ts.hourly.adjusted <- p.ts.hourly - mean(df.summer.9$d,
na.rm=TRUE) + p.ts$fit
p.ts.hourly.adjusted

#Plot ####

# setup plot layout
layout(matrix(c(1,1,1,2,3,4), nrow=2, ncol=3, byrow=TRUE),
widths=c(1, 1, 1, 1))
par(mar=c(3,4.5,1,0), cex.axis=0.6, cex.lab=0.6)
```

```
# All data with GAM plot
plot(d ~ date.time.CT, data=df, type='b', axes=FALSE, xlab='',
ylab='', pch=1, cex=0.75, col='RoyalBlue', lwd=1.5)
axis(2, cex.axis=0.75, line=-0.5, las=1, at=seq(1, 12, 1))
mtext('', side=2, cex=0.8, font=2, line=2)
lines(p.seq, p.ts.hourly.adjusted, lty=2)
lines(p.ts, lty=3)
axis.POSIXct(side=1, at=r.seq, cex.axis=0.75,
format="%m/%d\n%H:%M")
grid()
#legend('topright', lty=c(NA, NA, 3, 2), pch=c(1, 16, NA, NA),
legend=c('breast milk', 'formula', 'trend', 'trend+model'),
bty='n', cex=0.8, col=c('RoyalBlue', 'RoyalBlue', 'black',
'black'), horiz=TRUE)

# Data by hour plot
z<-as.numeric(df$diel.period)
plot(d ~ hour.fraction, data=df, axes=FALSE, xlab='', ylab='',
xlim=c(0,24), cex=0.75, col=z,ylim=c(0,10))
lines(p$hour, p$fit, lwd=2, col='RoyalBlue')
lines(p$hour, p$lower, lty=2, col='RoyalBlue')
lines(p$hour, p$upper, lty=2, col='RoyalBlue')
axis(2, cex.axis=0.75, line=-0.5, las=1, at=seq(0, 10, 1))
axis(1, cex.axis=0.75,pos=c(0,0), at=seq(0, 24, by=4))
grid()

# Circular plot
polar.plot(lengths=df.summer.9$d,
polar.pos=(df.summer.9$hour.fraction)*360/23, rp.type='s',
clockwise=TRUE,  start=0, labels=0:23, label.pos=1:24*360/24,
radial.lim=c(0,13), point.col=z, cex=0.5,point.symbols=1)
polar.plot(lengths=p$fit, polar.pos=(p$hour)*360/23,
rp.type='p', clockwise=TRUE, start=0, labels=0:23,
label.pos=1:24*360/24, radial.lim=c(0,5), lwd=2,
line.col='RoyalBlue', add=TRUE)
polar.plot(lengths=p$lower, polar.pos=(p$hour)*360/23,
rp.type='p', clockwise=TRUE, start=0, labels=0:23,
label.pos=1:24*360/24, add=TRUE, lty=2, radial.lim=c(0,5),
line.col='RoyalBlue')
polar.plot(lengths=p$upper, polar.pos=(p$hour)*360/23,
rp.type='p', clockwise=TRUE, start=0, labels=0:23,
label.pos=1:24*360/24, add=TRUE, lty=2, radial.lim=c(0,5),
line.col='RoyalBlue')

# Hour boxplot
```

```
z2<-
c(rep("blue",4),rep("black",2),rep("red",11),rep("green",2),rep(
"blue",4))
boxplot(d ~ hour, data=df, horizontal=TRUE, border=z2,
axes=FALSE, boxwex=0.5)
lines(p$fit, p$hour+1, lwd=2, col='RoyalBlue')
lines(p$lower, p$hour+1, lty=2, col='RoyalBlue')
lines(p$upper, p$hour+1, lty=2, col='RoyalBlue')
axis(1, cex.axis=0.75, las=1, line=-0.5, at=seq(1, 12, 1))
axis(2, cex.axis=0.75, las=1, at=1:24, labels=0:23, tick=FALSE,
line=-1)
stripchart(x$hour+1, method='stack', vertical=TRUE, axes=FALSE,
pch='|', cex=0.5, add=TRUE, at=5.75, offset=-0.25)
```

V-Track

```
#Load VTrack ####
library(VTrack)

##VTrack example data sets ####
data(AATAMS1)
data(crocs)

#Load detections ####
df<-read.csv('Detections file.csv')
head(df)


#### PUT DETECTIONS IN THIS ORDER AND FORMAT ####

DATETIME :POSIXct
TRANSMITTERID :Factor
SENSOR1 :Factor
UNITS1 :Factor
RECEIVERID :Factor
STATIONNAME :Character

df.v<- df[,c(1,3,4,5,11,12)] #Order and choose columns from
detections file

#Change column names
colnames(df.v)[1]<- "DATETIME"
colnames(df.v)[2]<- "TRANSMITTERID"
colnames(df.v)[3]<- "SENSOR1"
colnames(df.v)[4]<- "UNITS1"
colnames(df.v)[5]<- "RECEIVERID"
```

```
colnames(df.v)[6]<- "STATIONNAME"

df.v$DATETIME<- as.POSIXct(as.character(df$DATETIME),tz = "GMT")
#Change to character, convert to POSIXct, indicate time zone
df.v$TRANSMITTERID<- factor(df.v$TRANSMITTERID) #Change to
factor
df.v$RECEIVERID<- factor(df.v$RECEIVERID) #Change to factor
df.v$STATIONNAME<- as.character(df.v$STATIONNAME) #Change to
character
summary(df.v) #Check for NAs etc.

#Detections per fishta
table(df.v$TRANSMITTERID)

#Detections per receiver
table(df.v$STATIONNAME)

#Detections per receiver per fish
x<-table(df.v$TRANSMITTERID,df.v$STATIONNAME)
x

#Number of fish that visited each receiver
library(vegan); x.pa<-decostand(x,'pa') #convert to presence-
absence
x.pa
as.data.frame(colSums(x.pa)) #Number of fish that visited each
receiver

#Generate receiver distances from lat, longs, and receiver
radius ####
df.rec<-read.csv('Receiver locations.csv')
df.rec$RADIUS<- rep(0,26) #Create a variable for detection
radius, this example has a radius of 0 for 26 receivers

#format = LOCATION, LATITUDE, LONGITUDE, RADIUS in meters
df.rec<-df.rec[,c(1,4,5,8)] #Order columns
df.rec
library (VTrack)
rec.dist<-GenerateDirectDistance(df.rec) #Create distance matrix
of distances between receivers
rec.dist
summary(rec.dist) #Check that distances make sense

#Residence Extraction ####
library(VTrack)
a<-Sys.time()
res.output<-RunResidenceExtraction(df.v, #Formated detections
```

```
                                    "RECEIVERID", #use RECEIVERID
OR STATIONNAME
                                    2, #Min number of detections
                                    3600, #Max time of before
residence time stops (1 hr)
                                    sDistanceMatrix = rec.dist)
#Receiver distances
b<- Sys.time()
b-a #Time it took to run extraction

res<-res.output$residences #Residence output
non.res<-res.output$nonresidences #Movment output

#Sum duration at each station by fish
sum.dur.perfish<- aggregate(DURATION ~ TRANSMITTERID +
STATIONNAME, data = res, sum)
sum.dur.perfish

#Cross tabulation of trasmiters by stations
sum.dur.perfish.x<-xtabs(DURATION ~ TRANSMITTERID + STATIONNAME,
data=sum.dur.perfish)
sum.dur.perfish.x

write.table(sum.dur.perfish.x,"clipboard -
16384",sep="\t",col.names=NA) #Copy to paste in Excel
```

Home ranges

```
Websites:
http://www.mikemeredith.net/blog/1212 Data for home range analys
is in R.htm blog
http://cran.r-
project.org/web/packages/adehabitatHR/vignettes/adehabitatHR.pdf
adehabitatHR

Joe Code:
library(PBSmapping) #powerful mapping functions developed by
Pacific Biological Station
data(nepacLLhigh)

detect3<-read.csv('53 fish detections for Joe - jms edit 7-21-
2014 with basins-detections removed.csv',header=TRUE)

names(detect3)

(mean.lat.all<-
aggregate(Latitude~Serial_Number,data=detect3,mean))
```

```
(mean.long.all<-
aggregate(Longitude~Serial_Number,data=detect3,mean))

#Fish 754 ####
f754<-subset(detect3,Serial_Number==754) #subset by fish SN
head(f754)
dim(f754)
(f754.lat.long<-f754[,8:9]) #only lat and long
(f754.chull<-chull(f754.lat.long)) #compute convex hull of a set
of points
(f754.chull <- c(f754.chull, f754.chull[1])) #add the first to
the last to complete the polygon
(f754.out.points<-f754.lat.long[f754.chull,]) #get lat and longs
for convex hull
plotMap(nepacLLhigh, xlim=c(-125, -122), ylim=c(47, 48.6),las=1,
        col="lightgrey",bg="white",main="Fish 754",cex.main=2)
polygon(f754.out.points$Latitude~f754.out.points$Longitude,col="
lightblue",lty=2)
points(f754$Latitude~f754$Longitude,pch=16,
       #col=f754$SiteIndex,
)
points(mean(f754$Latitude)~mean(f754$Longitude),pch=9,col="red")
#Plot mean lat and long
```

Raster data in R

Puget Sound Bathymetry example

```
library(raster)
library(rgdal)

#Import raster
ras<- raster("psdem.txt") #PSDEM2000 from
http://www.ocean.washington.edu/data/pugetsound/

#Define projection (find in metadata)
projection(ras)<-"+proj=utm +zone=10 +datum=NAD27"

#Convert projection to longlat
ras2<-projectRaster(ras,crs="+proj=longlat +datum=NAD27")

df #Dataframe that has lat longs to extract
long.lat #Dataframe with only lat and longs (needs to be ordered
long=column1 and lat=column2)

#Extract Depth ####
```

```
#This dataset has depth in decimeters. Divide values by 10 to
get meters
df$depth.decimeter<-extract(ras2,long.lat)
```

Plum Island Estuary

Data from: http://www.mass.gov/anf/research-and-tech/it-serv-and-support/application-serv/office-of-geographic-information-massgis/datalayers/ftplidar-2013-14-sandy.html

```
library(raster)
library(rgdal)

filenames<-dir() #directory file names
filenames<- filenames[1:36] #pick only the raster file names
filenames
all.ras<-do.call("merge", #merge rasters
                 lapply(filenames, #raster file names
                        raster)) #read in files as a raster
projection(all.ras)<-"+proj=utm +zone=19N +datum=NAD83" #define
projection

plot(all.ras)
```