Functions, Tests, and More

General Homework Notes

- Read the entire question.

General Homework Notes

- Read the entire question.
- Show your answers. Don't give me the code and assume it shows the answers (e.g., plots, output, etc)

General Homework Notes

- Read the entire question.
- Show your answers. Don't give me the code and assume it shows the answers (e.g., plots, output, etc)
- Many things can go in one code blook in a loop

General Homework Notes

- Read the entire question.
- Show your answers. Don't give me the code and assume it shows the answers (e.g., plots, output, etc)
- Many things can go in one code blook in a loop
- Please put spaces between blocks of code. One giant chunk of code with no line breaks is unreadable.

General Homework Notes

- Read the entire question.
- Show your answers. Don't give me the code and assume it shows the answers (e.g., plots, output, etc)
- Many things can go in one code blook in a loop
- Please put spaces between blocks of code. One giant chunk of code with no line breaks is unreadable.
- Read and follow the Google or Hadley Wickham's style guide!

General Homework Notes

- Read the entire question.
- Show your answers. Don't give me the code and assume it shows the answers (e.g., plots, output, etc)
- Many things can go in one code blook in a loop
- Please put spaces between blocks of code. One giant chunk of code with no line breaks is unreadable.
- Read and follow the Google or Hadley Wickham's style guide!
- Try knitr with markdown? (see the blog) (+1 per homework!)

Homework 3, The Final Problem

Often in plots we want to show an estimate and the variation around that estimate. Boxplots do this for a whole sample, but what if we want to see means and the variation around the means? Reproduce the following two plots. These are plots of the median diatom abundance in different months and the bootstrapped 95% confidence interval around the medians. Produce the plot using both the base R graphics package and in ggplot2. You'll need to look at some additional plotting functions to get those error lines in the base graphics package. Likewise, you'll need to play with some additional geoms for ggplot2. Feel free to spice up your graphs beyond what I have presented here.

Resources

- http://rseek.org
- http://stackoverflow.com/

Resources

- `http://rseek.org`
- `http://stackoverflow.com/`
- `http://gallery.r-enthusiasts.com/`
- `http://docs.ggplot2.org/`
- `http://blog.ggplot2.org/`

Read and Clean your Data

```r
plankton <- read.csv("./hampton.5.1-Baikal_74_97_moAvg_plankton.csv",
    skip = 1, na.strings = c("NA", " NA", ".", " "))
# Visually inspected for outliers, this is the cleanup
plankton <- plankton[which(plankton$diatom < 200), ]
plankton <- plankton[which(plankton$green < 100), ]
```

Workflow - think backwards

I need to plot medians, and CIs for each month...

Workflow - think backwards

I need to plot medians, and CIs for each month...

So I need to generate vectors or a dataset with one entry per month.

Workflow - think backwards

I need to plot medians, and CIs for each month...

So I need to generate vectors or a dataset with one entry per month.

Thus, I need to loop over each unique month value, and calculate some stuff.

Workflow - think backwards

I need to plot medians, and CIs for each month...

So I need to generate vectors or a dataset with one entry per month.

Thus, I need to loop over each unique month value, and calculate some stuff.

And before the loop, I'll need to create some blank vectors to store information

Now moving forward

```
# 1) Create a new data frame that will have the information
# for plotting as we need one row per month
newPlankton <- data.frame(Month = unique(plankton$Month))
```

Now moving forward

```r
# 1) Create a new data frame that will have the information
# for plotting as we need one row per month
newPlankton <- data.frame(Month = unique(plankton$Month))
```

Unique gets the unique values of a vector - could have done it with `levels(factor(plankton$Month))`

The for loop

```
#2) For loop to calculate the aggregated properties
for (i in 1:nrow(newPlankton)) {
```

Subsetting the Data

```
# 3) Get the monthly data set
shortDF < -plankton[which(plankton$Month == newPlankton$Month[i]),
    ]
```

Subsetting the Data

```
# 3) Get the monthly data set
shortDF < -plankton[which(plankton$Month == newPlankton$Month[i]),
    ]
```

Alternate solution:

```
shortDF <- subset(plankton, plankton$Month == newPlankton$Month[i])
```

Medians

```r
# 4) Get the Median
newPlankton$Diatom.Median[i] <- median(shortDF$diatom)
```

Bootstrapped CI from the Boostrap Library

```
  #5) Extract the monthly CIs
  shortDiatomMedian<-bootstrap(shortDF$diatom, 5000, median)
#
  newPlankton$Diatom.lowerCI[i] <-
        quantile(shortDiatomMedian$thetastar, 0.025)
#
  newPlankton$Diatom.upperCI[i] <-
        quantile(shortDiatomMedian$thetastar, 0.975)

}
```

Bootstrapped CI from the Boostrap Library

```
  #5) Extract the monthly CIs
  shortDiatomMedian<-bootstrap(shortDF$diatom, 5000, median)
#
  newPlankton$Diatom.lowerCI[i] <-
        quantile(shortDiatomMedian$thetastar, 0.025)
#
  newPlankton$Diatom.upperCI[i] <-
        quantile(shortDiatomMedian$thetastar, 0.975)

}
```

Note `shortDiatomMedian$thetastar` - looks like a data frame, right? This is list syntax

What is a list?

A list is an object with a key-value combination. Each slot in a list has a unique key and can contain anything.

```
newList <- list(a = 1, b = rnorm(3))
```

What is a list?

A list is an object with a key-value combination. Each slot in a list has a unique key and can contain anything.

```r
newList <- list(a = 1, b = rnorm(3))
```

```r
newList$a

## [1] 1

newList$b

## [1]  0.07302  0.67083 -0.91536
```

What is a list?

You can reference the name of an element in a list many ways

```
newList[["a"]]

## [1] 1

newList[[1]]

## [1] 1
```

What is a list?

You can reference the name of an element in a list many ways

```
newList[["a"]]

## [1] 1

newList[[1]]

## [1] 1
```

```
newList

## $a
## [1] 1
##
## $b
## [1]  0.07302  0.67083 -0.91536
```

What is a list?

Lists can even contain lists - it can get a little silly.

```
newList$foo <- list(bar = 13)

newList$foo$bar

## [1] 13
```

Plotting Using the Graphics Package

```r
#6) plot for points, segments for error bars
plot(Diatom.Median ~ Month,
     data = newPlankton,
     pch = 19, ylim = c(0,15))
```

Plotting Using the Graphics Package

```
#6) plot for points, segments for error bars
plot(Diatom.Median ~ Month,
     data = newPlankton,
     pch = 19, ylim = c(0,15))
```

```
#segments takes x0, y0, x1, y1 as arguments
segments(newPlankton$Month, newPlankton$Diatom.lowerCI,
         newPlankton$Month, newPlankton$Diatom.upperCI)
```

Plotting Using Ggplot2

```
#7) the ggplot2 way uses geom_point and geom_linerange
#although geom_pointrange would also work
  ggplot(data=newPlankton,
         mapping = aes(x = Month, y = Diatom.Median,
                       ymin = Diatom.lowerCI,
                       ymax = Diatom.upperCI)) +
  geom_point() +
  geom_linerange() +
  theme_bw()
```

Functions!

What is a function?

Functions take some object(s) and use it to give us either a new object or perform an action.

```
sum(1:10)

## [1] 55
```

What is inside of that function

Functions take some object(s) and use it to give us either a new object or perform an action.

```
sum

## function (..., na.rm = FALSE)  .Primitive("sum")
```

What is inside of that function

`function(`*arguments*`)` **Code Block**

Example: addOne

```r
addOne <- function(x) x + 1
```

Example: addOne

```r
addOne <- function(x) x + 1
```

```r
addOne(3)

## [1] 4
```

Default Values

```
addOne <- function(x = 0) x + 1
```

Default Values

```r
addOne <- function(x = 0) x + 1
```

```r
addOne()
```

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

More Hygenic Code: Code Blocks

```r
addOne <- function(x = 0) {
    x + 1
}
```

More Hygenic Code: Code Blocks

```
addOne <- function(x = 0) {
    x + 1
}
```

Note that the last output is returned to the user.

More Hygenic Code: Return

```r
addOne <- function(x = 0) {
    return(x + 1)
}
```

Exercise: Two Functions

1. Write a squaring function (i.e., square(3) $= 9$)
2. Write an add function that returns the sum of two numbers. If no numbers are supplied, it returns 0. If only one is supplied, it returns that number.

Exercise: Two Functions

```
square <- function(x) x * x
```

Exercise: Two Functions

```r
square <- function(x) x * x
```

```r
add <- function(x = 0, y = 0) {
    return(x + y)
}
```

Functions for Repetitive Tasks With a Lot of Code

```r
sumFun <- function(aVec){
  #start with 0
  out <- 0

  #loop over the vector, adding
  #each element together
  for(i in aVec){
    out <- out + i
  }

  #return the result
  return(out)
}
```

... - the Garbage Collector

Don't you just hate how you need to make a vector for sum?
`sum(c(4,5,6,1,2,3))`

```
sumNoC <- function(...) {
    # convert ... into a vector
    avec <- c(...)

    # NOW sum the vector
    sum(avec)
}
```

... - the Garbage Collector

Don't you just hate how you need to make a vector for sum?
`sum(c(4,5,6,1,2,3))`

```
sumNoC <- function(...) {
    # convert ... into a vector
    avec <- c(...)

    # NOW sum the vector
    sum(avec)
}
```

This may seem trivial, but it's a nice way to pass arguments between functions.

Exercise: Cummulative Vectors

Write a function that returns a list with the cummulative sum, product, and mean of a vector. Allow it to pass arguments to other functions (e.g., mean takes arguments to deal with NAs).

Exercise: Cummulative Vectors

```r
cumSumProdMean <- function(aVec, ...) {
    # get our sum and product vectors ready
    s <- rep(NA, length(aVec))
    s[1] <- aVec[1]
    m <- p <- s

    # now loop!
    for (i in 2:length(aVec)) {
        s[i] <- s[i - 1] + aVec[i]
        p[i] <- p[i - 1] * aVec[i]
        m[i] <- mean(aVec[1:i], ...)
    }

    # return the results in a list
    return(list(sums = s, prod = p, mean = m))
}
```

Exercise: Cummulative Vectors

```
cumSumProdMean(1:10)

## $sums
##  [1]  1  3  6 10 15 21 28 36 45 55
##
## $prod
##  [1]       1       2       6      24     120     720    5040
##  [8]   40320  362880 3628800
##
## $mean
##  [1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5
```

Exercise: Cummulative Vectors

```
cumSumProdMean(c(1:5, NA, 7:10))

## $sums
##  [1]  1  3  6 10 15 NA NA NA NA NA
##
## $prod
##  [1]   1   2   6  24 120  NA  NA  NA  NA  NA
##
## $mean
##  [1] 1.0 1.5 2.0 2.5 3.0  NA  NA  NA  NA  NA
```

Exercise: Cummulative Vectors

```
cumSumProdMean(c(1:5, NA, 7:10), na.rm = T)

## $sums
##  [1]  1  3  6 10 15 NA NA NA NA NA
##
## $prod
##  [1]   1   2   6  24 120  NA  NA  NA  NA  NA
##
## $mean
##  [1] 1.000 1.500 2.000 2.500 3.000 3.000 3.667 4.286 4.875
## [10] 5.444
```