Recap From Last Time:

- Why it is important to visualize data during exploratory data analysis.
- Discussed data visualization best practices and how good visualizations optimize for the human visual system.
- Introduced the extensive graphical capabilities of R with a focus on generating and customizing scatterplots, histograms, bar graphs, boxplots, (dendrograms and heatmaps).
- Use the par() function to control fine grained details of the afore mentioned plot types.
- Noted that you can build even more complex charts with additional R packages such as ggplot2 and rgl.

[MPA Link]

Today's Learning Goals

- **More on data import**
  - Pre-check recommendations
  - read.table() and friends for flat files
  - readxl::read_excel() for excel files
  - RData files
- **Writing your own functions**
  - What, Why, When and How
- **The CRAN & Bioconductor R package repositories**
  - Rmarkdown, ggplot2, bio3d, rgl and rentrez
- Additional R packages (and development versions of CRAN packages) on **GitHub** and **BitBucket**.

Today's Learning Goals

- **More on data import**
  - Pre-check recommendations
  - read.table() and friends for flat files
  - readxl::read_excel() for excel files
  - RData files
- **Writing your own functions**
  - What, Why, When and How
- **The CRAN & Bioconductor R package repositories**
  - Rmarkdown, ggplot2, bio3d, rgl and rentrez
- Additional R packages (and development versions of CRAN packages) on **GitHub** and **BitBucket**.

Pre-check recommendations

- **Get organized!**
  - Start a new 'project' in a directory you know about and store all needed project material here (input, scripts and output).
  - In RStudio **File > New Project > New Directory >** …
  - Or use **ls()**; **rm(list=ls())**; **getwd()**; **setwd()**; **dir()** functions.

Pre-check recommendations

- **Get organized!**
  - Start a new 'project' in a directory you know about and store all needed project material here (input, scripts and output).
  - In RStudio **File > New Project > New Directory >** …
  - Or use **ls()**; **rm(list=ls())**; **getwd()**; **setwd()**; **dir()** functions.
- **Inspect the file at the command line**
  - Use the UNIX commands **head**, **less** etc.
  - Does it have a header line or comments to be included, ignored or removed?
  - Avoid file (or field names) with spaces or special characters such as ?, $,%, ^, &, *, } etc.
  - Short names are prefered over longer names.
  - Does the file end with a blank line or a RTN?

read.table() and friends for flat files

- **The read.table() function is the base of all flat file import functions**
  - **read.csv**("filename.txt", stringsAsFactors=FALSE) COMMA
  - **read.csv2**("filename.txt", stringsAsFactors=FALSE) SEMI-COLON
  - **read.delim**("filename.txt", stringsAsFactors=FALSE) TAB
  - **read.table**("filename.txt", stringsAsFactors=FALSE) SPACE or ….
- **What other differences are there between these functions?**
- **MS EXCEL file import options include:**
  - Export (i.e. "Save As…") your excel data to plain text CSV format.
  - Or use **readxl::read_excel()** to read specified parts of your sheets/workbooks etc.
- **For fast and convenient reading of very large flat files files**
  - Try **data.table::fread()** use is similar to **read.table()** but it automatically finds field separators and header rows. It is also much faster!
- **Saving and loading .RData files…**
  - Use the functions **save()** and **load()** for saving and loading multiple objects to space efficient binary format files.

Your turn!

Do it Yourself!

https://bioboot.github.io/bggn213_f17/class-material/test1.txt
https://bioboot.github.io/bggn213_f17/class-material/test2.txt
https://bioboot.github.io/bggn213_f17/class-material/test3.txt

- Start a new RStudio *Project* in a clean directory
- Open a new Rmarkdown document and give it a name and descriptive text.
- Download each of the above files and move them into your *Project*
- Experiment with **read.table()** to get their data successfully input into your R session.

Today's Learning Goals

- **More on data import**
  - Pre-check recommendations
  - read.table() and friends for flat files
  - readxl::read_excel() for excel files
  - RData files
- **Writing your own functions**
  - What, Why, When and How
- **The CRAN & Bioconductor R package repositories**
  - Rmarkdown, ggplot2, bio3d, rgl and rentrez
- Additional R packages (and development versions of CRAN packages) on **GitHub** and **BitBucket**.

What is a function

```
name.of.function <- function(arg1, arg2) {
    statements
    return(something)
}
```

1. **Name** (can be *almost* anything you want)
2. **Arguments** (i.e. input to your function)
3. **Body** (where the work gets done)

What is a function

```
add <- function(x, y=1) {
    # Sum the input x and y
    x + y
}
```

1. **Name** (in this case "**add**")
2. **Arguments** (here "**x**" and "**y**")
3. **Body** (will return the result of the last statement)

Your function is treated just like any other function…

```
add <- function(x, y=1) {
    # Sum the input x and y
    x + y
}

add(x=1, y=4)
add(1, 4)
add(1)

add( c(1, 2, 3) )
add( c(1, 2, 3), 4 )

add(1, 2, 2)
add(x=1, y="b")
```

Why would you write a function

When you find yourself doing the same thing 3 or more times it is time to write a function.

```
## What does this code do?

df$a <- (df$a - min(df$a)) / (max(df$a) - min(df$a))

df$b <- (df$b - min(df$a)) / (max(df$b) - min(df$b))

df$c <- (df$c - min(df$c)) / (max(df$c) - min(df$c))

df$d <- (df$d - min(df$d)) / (max(df$a) - min(df$d))
```

Why would you write a function

When you find yourself doing the same thing 3 or more times it is time to write a function.

```
## Consider copy and paste errors:

df$a <- (df$a - min(df$a)) / (max(df$a) - min(df$a))

df$b <- (df$b - min(df$a)) / (max(df$b) - min(df$b))

df$c <- (df$c - min(df$c)) / (max(df$c) - min(df$c))

df$d <- (df$d - min(df$d)) / (max(df$a) - min(df$d))
```

Why would you write a function

Consider the advantages:

```
## Here the intent is far more clear

df$a <- rescale(df$a)
```

- Makes the purpose of the code more clear
- Reduce mistakes from copy/paste
- Makes updating your code easer
- Reduce duplication and facilitate re-use.

How would you write this function

Start with a **working code snippet**, simplify, reduce calculation duplication,…

```
## First consider the original code:

df$a <- (df$a - min(df$a)) / (max(df$a) - min(df$a))

df$b <- (df$b - min(df$a)) / (max(df$b) - min(df$b))

df$c <- (df$c - min(df$c)) / (max(df$c) - min(df$c))

df$d <- (df$d - min(df$d)) / (max(df$a) - min(df$d))
```

How would you write this function

Start with a working code snippet, **simplify**, reduce calculation duplication,…

```
## Simplify to work with a generic vector named "x"

x <- (x - min(x)) / (max(x) - min(x))
```

How would you write this function

Start with a working code snippet, simplify, **reduce calculation duplication**,…

```
## Note that we call the min() function twice…

x <- (x - min(x)) / (max(x) - min(x))
```

How would you write this function

Start with a working code snippet, simplify, **reduce calculation duplication**,…

```
## Note that we call the min() function twice…

xmin <- min(x)
x <- (x - xmin) / (max(x) - xmin)
```

How would you write this function

Start with a working code snippet, simplify, **reduce calculation duplication**,…

```
## Further optimization to use the range() function…

rng <- range(x)
x <- (x - rng[1]) / (rng[2] - rng[1])
```

How would you write this function

Start with a working code snippet, simplify, reduce calculation duplication, finally turn it into a **function**

```
## You need a “name”, “arguments” and “body”…

rescale <- function(x) {
   rng <-range(x)
   (x - rng[1]) / (rng[2] - rng[1])
}

# Test on a small example where you know the answer
rescale(1:10)
```

How would you write this function

Test, **Fail**, Change, Test again,…

```
# Test on a small example where you know the answer
rescale(1:10)

# How would you get your function to work here…
rescale( c(1,2,NA,3,10) )

# What should your function do here?
recale( c(1,10,”string”) )
```

Side-Note: Seeing and using your function in RStudio

- An easy way to visualize the code of a function is to type its name without the parentheses ().
- If you have your new function saved to a separate file then you can load and execute it using the **source()** function. E.g. **source(“MyUtils.R")**
- The **return()** statement is not required in a function but it is advisable to use it when the function performs several computations. It has the effect of ending the function execution and returning control to the code which called it.

```
rescale <- function(x, na.rm=TRUE, plot=FALSE) {
   if(na.rm) {
     rng <-range(x, na.rm=TRUE)
   } else {
     rng <-range(x)
   }
   print("Hello")

   answer <- (x - rng[1]) / (rng[2] - rng[1])

   print("is it me you are looking for?")

   if(plot) {
      plot(answer, typ="b", lwd=4)
   }
   print("I can see it in ...")
}
```

```
rescale <- function(x, na.rm=TRUE, plot=FALSE) {
   if(na.rm) {
     rng <-range(x, na.rm=TRUE)
   } else {
     rng <-range(x)
   }
   print("Hello")

   answer <- (x - rng[1]) / (rng[2] - rng[1])
   return(answer)
   print("is it me you are looking for?")

   if(plot) {
      plot(answer, typ="b", lwd=4)
   }
   print("I can see it in ...")
}
```

Do it Yourself!

Your turn!

http://tinyurl.com/bggn213-L9

Concentrate on **Section 1B** and questions 1 to 6.
Other sections are there for your benefit.

Do it Yourself!

```
# Can you improve this analysis code?
library(bio3d)
s1 <- read.pdb("4AKE")  # kinase with drug
s2 <- read.pdb("1AKE")  # kinase no drug
s3 <- read.pdb("1E4Y")  # kinase with drug

s1.chainA <- trim.pdb(s1, chain="A", elety="CA")
s2.chainA <- trim.pdb(s2, chain="A", elety="CA")
s3.chainA <- trim.pdb(s1, chain="A", elety="CA")

s1.b <- s1.chainA$atom$b
s2.b <- s2.chainA$atom$b
s3.b <- s3.chainA$atom$b

plotb3(s1.b, sse=s1.chainA, typ="l", ylab="Bfactor")
plotb3(s2.b, sse=s2.chainA, typ="l", ylab="Bfactor")
plotb3(s3.b, sse=s3.chainA, typ="l", ylab="Bfactor")
```

Today's Learning Goals

- **More on data import**
  - Pre-check recommendations
  - read.table() and friends for flat files
  - readxl::read_excel() for excel files
  - RData files
- **Writing your own functions**
  - What, Why, When and How
- **The CRAN & Bioconductor R package repositories**
  - Rmarkdown, ggplot2, bio3d, rgl and rentrez
- Additional R packages (and development versions of CRAN packages) on **GitHub** and **BitBucket**.

R Highlight!

CRAN & Bioconductor

Major repositories for **R packages** that extend R functionality

CRAN: Comprehensive R Archive Network

- CRAN is a network of mirrored servers around the world that administer and distribute R itself, R documentation and **R packages** (basically add on functionality!)
- There are currently ~11,700 packages on CRAN in the areas of finance, bioinformatics, machine learning, high performance computing, multivariate statistics, natural language processing, *etc. etc.*

https://cran.r-project.org/

Side-note: R packages come in all shapes and sizes





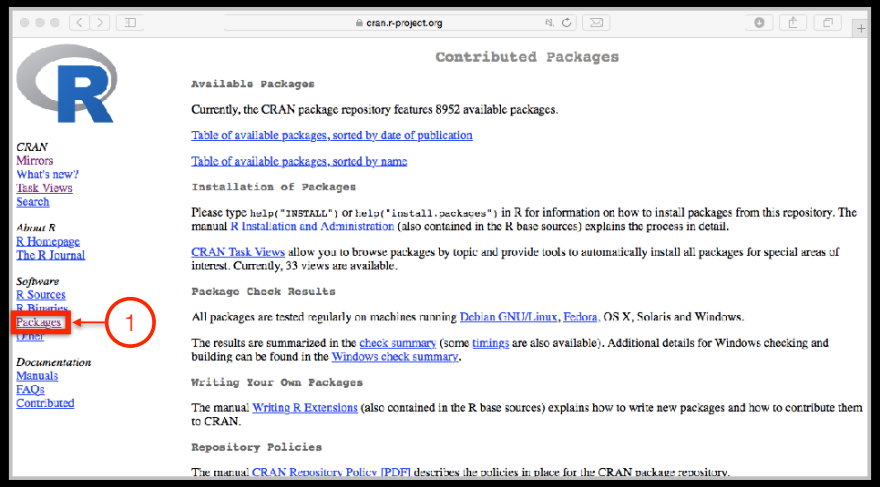
R packages can be of variable quality and often there are multiple packages with overlapping functionality.

Refer to relevant publications, package citations, update/maintenance history, documentation quality and your own tests!

> "The journal has sufficient experience with CRAN and Bioconductor resources to endorse their use by authors. We do not yet provide any endorsement for the suitability or usefulness of other solutions."

From: "Credit for Code". *Nature Genetics* (2014), 46:1

https://cran.r-project.org



Installing a package

RStudio > Tools > Install Packages

> install.packages("bio3d")

> library("bio3d")

Do it Yourself!

Your Turn: Pick a package to explore and install

(~~Rmarkdown~~), ggplot2, bio3d, rgl, rentrez, igraph

Questions to answer:

- How does it extend R functionality? (i.e. What can you do with it that you could not do before?)
- How is it's documentation, vignettes, demos and web presence?
- Can you successfully follow a tutorial or vignette to get started quickly with the package?
- Can you find a GitHub or Bitbucket site for the the package with a regular heartbeat?

Collaborative Google Doc Link

Bioconductor

R packages and utilities for working with high-throughput genomic data

http://bioconductor.org

More pragmatic:
Bioconductor is a software repository of R packages with some rules and guiding principles.

Version 3.3 had 1211 software packages.

Bioconductor has emphasized
Reproducible Research
since its start, and has been an early adapter and driver of tools to do this.

"Bioconductor: open software development for computational biology and bioinformatics"
Gentleman et al
Genome Biology 2004, 5:R80

"Orchestrating high-throughput genomic analysis with Bioconductor"
Huber et al
Nature Methods 2015, 12:115-121

Installing a bioconductor package

```
> source("https://bioconductor.org/biocLite.R")
> biocLite()
> biocLite("GenomicFeatures")
```

See: http://www.bioconductor.org/install/

Summary

- R is a powerful data programming language and environment for statistical computing, data analysis and graphics.
- Introduced R syntax and major R data structures
- Demonstrated using R for exploratory data analysis and graphics.
- Exposed you to the why, when, and how of writing your own R functions.
- Introduced CRAN and Bioconductor package repositories.

Learning Resources

- **TryR**. An excellent interactive online R tutorial for beginners.
  < http://tryr.codeschool.com/ >
- **RStudio**. A well designed reference card for RStudio.
  < https://help.github.com/categories/bootcamp/ >
- **DataCamp**. Online tutorials using R in your browser.
  < https://www.datacamp.com/ >
- **R for Data Science**. A new O'Reilly book that will teach you how to do data science with R, by Garrett Grolemund and Hadley Wickham.
  < http://r4ds.had.co.nz/ >