Module 2: Reproducible Research and Data

1 Course Description and Objectives

This course is designed to explore the concept of reproducible research for environmental sciences. We will cover proper data storage and the use of the R statistical computing environment to manipulate that data for statistical and graphical analysis. A background in a computer language and statistics is a plus but not required.

1.1 What is "Reproducible Research"?

In a very general sense, reproducible research is one of the fundamental aims of science. When we write a paper we include methods and results sections so that, in theory, other researchers can potentially reproduce and confirm our results.

Lately, the term "Reproducible Research" has been used to really mean "Reproducible Data Analysis". Every step of data analysis (often in the form of code) is prodvided.. * Import, manipulate and transform raw data and meta-data into processed data.
Run statistical analysis. * Create plots and tables. * Incorporate analysis and plots into a report.

1.2 Benefits of "Reproducible Research".

Many journals currently have data sharing and code sharing policies in place and that number has been increasing in recent years (Stodden et al 2013).

Even if you aren't planning on publishing in a journal with such policies, there is much to be said about being able to reproduce your own data analysis in as little effort as possible.

1.2.1 Example of non-reproducible data analysis

1. Enter raw data into Excel
2. Hand edit data for QA/QC
3. Copy and paste data subsets of data into new worksheet as "processed data"
4. Do analysis in Excel or export/import data into other analysis software (R, SAS, Sigmaplot, etc..)
5. Point-n-click your way through your analysis
6. Save formatted plots and table output
7. Insert formatted plots and tables into Word Document

What happens if you obtain new data or find an error in your raw data?

You start over from step 1 or 2 and spend potentially just as much time as you initially did running your analysis.

1.2.2 Example of reproducible data analysis

1. Enter raw data into database, Excel, or CSV files
2. Use R to assist QA/QC
3. Use R to process your data for further analysis
4. Use R to preform statistical and graphical analysis
5. Use R markdown script to incorporate plots and tables into final report
6. Use Pandoc command to convert R markdown into a final report in html, word doc, PDF, etc..
7. Create a master script that combines all of these steps into one.

What happens now if you get more data or need to change something?

You edit your data and scripts as needed and rerun the master script. Yes, you spent a bit more time setting things up initially but now rerunning analyses is trivial.

1.2 CRAN Task View on Reproducible Research

For an overview of R packages for Reproducible Research see the following..

[CRAN Task View: Reproducible Research] (http://cran.r-project.org/web/views/ReproducibleResearch.html)

2 Data Storage

2.1 Spreadsheets

Microsoft Excel and other spreadsheets (such as Google Speadsheets or Open Office) are probably the most common method for data entry and data sharing. However, many people use spreadsheets in a way that makes the data practially unusable to others software.

In this example the spreadsheet is formatted like a data entry sheet and/or reporting table which is fine, unless you need to share the raw data with other software.

A proper data spreadsheet should look more like a database tabale with unique data as rows and variables as columns with the column headers listing those variables.

One other complication to using spreadsheets in this fashion is that there can be incompatibilites based on which version of the spreadsheet software is being used.

2.2 Relational Databases (Microsoft Access, MySQL, PostgreSQL, etc..)

Sometimes data is too big and too complicated to store in a simple spreadsheet. When this happens databases are needed. Relational databases are collections of data tables in which the relationships between the tables is formally described. For example, site information might be in one table called "sites" while sample information might be in a table called "samples". Each site might have multiple samples and each sample belongs to one particular site. This relationship would be called a "one-to-many" relationship. For each record in the samples table there would be a "site id" of some sort which helps define how each sample record is related to the sites table.

Microsoft Access is great for creating quick desktop based data entry forms and has a nice visual method of creating data queries.

Other databases like MySQL and PostgreSQL are a bit more powerful and are often used in conjunction with a webserver to serve datasets online.

There are special packages in R that allow you to access data from relational databases.

2.3 CSV (comma seperated values)

Almost every spreadsheet and database can export properly formatted data into CSV files. Likewise, just about every program used for data analysis can import CSV files. Because CSV files are simple text files, they are the *lingua franca* of data formats. However, large amounts of data in text format can take awhile to load.

The following code shows how long it takes to load a 44.6 Mb file with 332,633 rows of data and 13 variables.

```
csv_speed <- system.time(data <- read.csv("~/WSA/Data/WSA_data.csv"))
print(csv_speed)
```

```
##    user  system elapsed
##   3.973   0.015   3.990
```

Here I've wrapped the 'system.time' function call around the 'read.csv' function that loads the data file so that I can time it. This can be useful if you ever want to know how long a particular part of your analysis takes.

	A	B	C	D	E	F	G	H	I
1	Detention Pond Outlet Flow Control Design - U.S. units								
2									
3	2. Single Stage Weir Control								
4									
5	**Inputs**					**Calculations**			
6									
7	Water elevation at						Weir length, L_w =	0.70	ft
8	design volume, E_s =		5.7	ft					
9									
10	Weir crest								
11	elevation, E_w =		2.4	ft					
12									
13	Pre-development peak								
14	runoff rate, q_{pb} =		13	cfs					
15									
16	Weir coefficient, C_w =		3.1						
17									

Figure 1: Example of a bad spreadsheet design

2.4 Binary Files (e.g. '.Rdata')

One option to speed this process up in R is to save the data in a binary Rdata file. This file format compresses the data in such a way that it can be reopened at least 10 times faster then .

```
save(data, file = "data/WSA_data.Rdata")
binary_speed <- system.time(load(file = "data/WSA_data.Rdata"))
print(binary_speed)
```

```
##    user  system elapsed
##   0.087   0.008   0.095
```

The loading of the binary data file was over **42** times faster than loading the csv file!

2.5 R Objects

Dimensions	Homogeneous	Heterogeneous
1d	Atomic Vector	List
2d	Matrix	Data Frame
3d	array	

Vectors The most basic variable structure is a vector (1d, homogeneous). Generally created using "c()" to concatenate values of the same type.

A vector of numbers

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12
```

A vector of characters

```
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l"
```

A vector of booleans

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

to access values in a vector we use brackets "[]"

To access the third element of our previous vectors

```
n[3]
```

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

```
l[3]
```

```
## [1] "c"
```

```
b[3]
```

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

Matrices Matrices are two dimensional structures of a single data type.

A matrix of numbers

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    1    5    9   13   17   21
## [2,]    2    6   10   14   18   22
## [3,]    3    7   11   15   19   23
## [4,]    4    8   12   16   20   24
```

A matrix of characters

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,] "a"  "e"  "i"  "m"  "q"  "u"
## [2,] "b"  "f"  "j"  "n"  "r"  "v"
## [3,] "c"  "g"  "k"  "o"  "s"  "w"
## [4,] "d"  "h"  "l"  "p"  "t"  "x"
```

To access values in a matrix we use the "[]" but with two elements inside representing rows and columns. The first element is for the first dimension (rows) and the second element is for the second dimension (columns)

```
# first element row = 1 , column = 1
ml[1, 1]
```

```
## [1] "a"
```

```
# entire first row
ml[1, ]
```

```
## [1] "a" "e" "i" "m" "q" "u"
```

```
# entire first column
ml[, 1]
```

```
## [1] "a" "b" "c" "d"
```

Arrays An array is like a matrix to but with more dimensions..

```
## , , 1
##
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12
##
## , , 2
##
##      [,1] [,2] [,3] [,4]
## [1,]   13   16   19   22
## [2,]   14   17   20   23
## [3,]   15   18   21   24
```

Accessing values in an array is similar to that of matrices but you add more elements in the brackets for each additional dimension [Rows, Columns, Extra Dimension 1, Extra Dimension 2, etc..]

```
# first element (row = 1, column = 1, extra dimension element = 1)
a[1, 1, 1]
```

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

```
# second column of second extra dimension element
a[, 2, 2]
```

```
## [1] 16 17 18
```

```
# third row of first extra dimension element

a[3, , 1]
```

```
## [1]  3  6  9 12
```

Data Frames Most data tables are represented as data frames in R. They are two dimensional like a matrix, but allow for mulitple data types with in a single structure. However, each column of a data frame is has to be a consistant type and all columns must be the same length.

```
##    l  n     b
## 1  a  1  TRUE
## 2  b  2  TRUE
## 3  c  3 FALSE
## 4  d  4  TRUE
## 5  e  5  TRUE
## 6  f  6 FALSE
## 7  g  7 FALSE
## 8  h  8 FALSE
## 9  i  9  TRUE
## 10 j 10 FALSE
## 11 k 11  TRUE
## 12 l 12 FALSE
```

Accessing elements of a data frame can be done the same way as a matrix using "[row, column]" notation. Data frame columns can also be referred to by name using the "$" to separate the data frame name and the column name

```
# first element of the data frame
df[1, 1]
```

```
## [1] a
## Levels: a b c d e f g h i j k l
```

```
# first row of the data frame
df[1, ]
```

```
##   l n    b
## 1 a 1 TRUE
```

```
# first column of the data frame
df[, 1]
```

```
##  [1] a b c d e f g h i j k l
## Levels: a b c d e f g h i j k l
```

```
# or access the first column by name 'l'
df$l
```

```
##  [1] a b c d e f g h i j k l
## Levels: a b c d e f g h i j k l
```

```
# first value of the 'l' column

df$l[1]
```

```
## [1] a
## Levels: a b c d e f g h i j k l
```

Lists You can think of lists as mutli-dimensional data.frames. Another way to think about them is a list of various objects. They take a bit to wrap your head around, but can be powerful once you do. Like data frames, lists can contain a heterogenaous set of data types. However, unlike data frames, the elements of a list don't have to be the same size.

```
## [[1]]
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l"
##
## [[2]]
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12
##
## [[3]]
##  [1]  TRUE  TRUE FALSE  TRUE  TRUE FALSE FALSE FALSE  TRUE FALSE  TRUE
## [12] FALSE
##
## [[4]]
##    l  n     b
## 1  a  1  TRUE
## 2  b  2  TRUE
## 3  c  3 FALSE
## 4  d  4  TRUE
## 5  e  5  TRUE
## 6  f  6 FALSE
## 7  g  7 FALSE
## 8  h  8 FALSE
## 9  i  9  TRUE
## 10 j 10 FALSE
## 11 k 11  TRUE
## 12 l 12 FALSE
```

To access values in a list you access the object in the list using "[[]]" and then, depending on that object's type, you access the values accordingly..

```
# the first object in lst
lst[[1]]
```

```
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l"
```

```
# the third item of the first object.
lst[[1]][3]
```

```
## [1] "c"
```

```
# the 4th object
lst[[4]]
```

```
##    l  n     b
## 1  a  1  TRUE
## 2  b  2  TRUE
## 3  c  3 FALSE
## 4  d  4  TRUE
## 5  e  5  TRUE
## 6  f  6 FALSE
## 7  g  7 FALSE
## 8  h  8 FALSE
## 9  i  9  TRUE
## 10 j 10 FALSE
## 11 k 11  TRUE
## 12 l 12 FALSE
```

```
# the fourth oject in the first column of the 4th object

lst[[4]][4, 1]
```

```
## [1] d
## Levels: a b c d e f g h i j k l
```

```
# or

lst[[4]]$l[4]
```

```
## [1] d
## Levels: a b c d e f g h i j k l
```

Numeric Numeric variables are simply numbers with a decimal point

Integer Numbers without a decimal point.

Character Strings.

Date Dates can be tricky and there a variety of R functions and packages designed to help deal with them. Often dates are imported as character strings and need to be converted to date format if you want to treat them as dates. This is important for timelines and grouping by year, month, day, etc..

```
today <- Sys.Date()

format(today, "%d %B %Y")

## [1] "20 March 2014"

months(today)

## [1] "March"

weekdays(today)

## [1] "Thursday"


mydates <- c("1jan1960", "2jan1960", "31mar1960", "30jul1960")
months(mydates)

## Error: no applicable method for 'months' applied to an object of class
## "character"

mydates2 <- as.Date(mydates, "%d%b%Y")
months(mydates2)

## [1] "January" "January" "March"   "July"
```

More on dates later..

logical Simply TRUE or FALSE.

Factors Factors as a data type in R serve two main purposes... 1.) Factors reduce memory usage by coding large repeating portions of the data and replacing it with an integer.

2.) We often need factors anyway in our statistics. For example, in an ANOVA we might want to compare a dependant variable like growth among treatments. These treatments are "factors". By identifing this variable as a factor, R will treat it as such in analysis.

```
# list of treatments by sample
treatments <- c(1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 4, 4, 4, 4)
print(treatments)

##  [1] 1 1 1 1 1 2 2 2 2 2 2 2 2 3 3 4 4 4 4

summary(treatments)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1.00    1.50    2.00    2.26    3.00    4.00
```

```r
# This is Min, Median, Max, etc.. of the numeric values of the treatment
# numbers. Not what we want.

# Turn treatments into a factor
treatments <- factor(treatments)
print(treatments)
```

```
##  [1] 1 1 1 1 1 2 2 2 2 2 2 2 2 3 3 4 4 4 4
## Levels: 1 2 3 4
```

```r
summary(treatments)
```

```
## 1 2 3 4
## 5 8 2 4
```

```r
# now the summary of the treatments is the count by treatment. Makes much
# more sense.
```

2.6 NULL and NA

There is often some confusion between NA and NULL values in R.

NULL represents a completely NULL object (i.e. the object contains no values). If an object doesn't exist R will return a NULL.

NA represents a missing value within an object. NA's appear within an object as "NA".

2.7 Data Format (Wide vs Narrow)

2.7.1 Wide Data Format Most people are familiar with the "wide" data format where each row represents an observation and each column represents all the various attributes for that observation. In the following example, each species has its own column of count data.

Site	Date	species1	species2	species3	species4
A	2009-03-10	1	0	0	4
B	2009-03-10	2	7	0	0
C	2009-03-10	0	2	2	0
A	2009-04-13	0	1	1	5
B	2009-04-13	0	2	1	2
C	2009-04-13	1	5	2	0

2.7.2 Narrow Data Format Another representation of this data is called the "narrow" data format. In this next example, species name becomes a single data field called "Species" and a new column called "Count" is created. Also in this example, counts of zero for species not present are dropped.

Site	Date	Species	Count
A	2009-03-10	species1	1
A	2009-03-10	species4	4

B	2009-03-10	species1	2
B	2009-03-10	species2	7
C	2009-03-10	species2	2
C	2009-03-10	species3	2
A	2009-04-13	species2	1
A	2009-04-13	species3	1
A	2009-04-13	species4	5
B	2009-04-13	species2	2
B	2009-04-13	species3	1
B	2009-04-13	species4	2
C	2009-04-13	species1	1
C	2009-04-13	species2	5
C	2009-04-13	species3	2

2.7.3 Which format is best? Initally, the wide format probably the easiest to read. However, the narrow format is has many advantages, particularly in this example with species counts ...

1. It is easier to transform from narrow to wide than it is from wide to narrow
2. Adding a new species doesn't require a new data column
3. No need to record zeros all the time
4. Species becomes a factor which makes subsetting the data easier (more on this later)

The choice between wide and narrow table structure will depend on nature of the data being represented. In general, if you have multiple columns that could be represented easily as a single factor and a related value field (like species counts) than narrow is the way to go.

2.8 Normalizing Data Tables

I mentioned eariler that if your data set is big and complex a database is often needed. The process of deciding how to break up your data into related tables is called "normalization". We do this primarly to organize the data and to reduce redundancy.

Imagine you have sites and sample information. Each site has a name, a latitude and longitude, and several other descriptive fields and each sample within a site has information on how and when it was sampled as well as what was found.

The following table is includes everything but is unnormalized.

sampleID	site	lake	lat	lon	date	sampler	temp_C	fish
1	Boat Ramp A	Henry Hagg Lake	45.48	-123.2	2008-02-22	BPS	9.0	0
2	Boat Ramp A	Henry Hagg Lake	45.48	-123.2	2008-02-26	BPS	9.2	3
3	Boat Ramp A	Henry Hagg Lake	45.48	-123.2	2008-03-01	BPS	10.0	0
4	Boat Ramp C	Henry Hagg Lake	45.49	-123.2	2008-02-22	BPS	8.9	11
5	Boat Ramp C	Henry Hagg Lake	45.49	-123.2	2008-02-26	BPS	9.1	14

Notice that all of the site related information gets repeated for each sample it has.

Here is that same information normalized into a "sites" table

siteID	site	lake	lat	lon
1	Boat Ramp A	Henry Hagg Lake	45.48	-123.2
2	Boat Ramp C	Henry Hagg Lake	45.49	-123.2

and a "samples" table

sampleID	siteID	date	sampler	temp_C	fish
1	1	2008-02-22	BPS	9.0	0
2	1	2008-02-26	BPS	9.2	3
3	1	2008-03-01	BPS	10.0	0
4	2	2008-02-22	BPS	8.9	11
5	2	2008-02-26	BPS	9.1	14

Now the site related information is only recorded once and we use a "site_id" to related the samples table to the sites table.

We can also join these two tables back up based on the "site_id" if for example we need the latitude and longitude where a sample was collected.

Meta-data

Metadata is "data about data".

Who collected this data?
Purpose of the data?
How the data was created?
When the data was created?
Descriptions of all tables and data fields.

The more information you can record the better.

Citations

[1] Stodden V, Guo P, Ma Z (2013) Toward Reproducible Computational Research: An Empirical Analysis of Data and Code Policy Adoption by Journals. PLoS ONE 8(6): e67111. doi:10.1371/journal.pone.0067111 [link] (http://www.plosone.org/article/info%3Adoi%2F10.1371%2Fjournal.pone.0067111)

Homework

1. Find at least one dataset you might be interested in working with over the next couple classes. If you're stumped, try the following data repository...

http://datadryad.org

2. Find or create the metadata required to describe this found dataset. Include the who, what, where, when, and how of the dataset as well as a description of the various data fields themselves.