Bio395

Jameson Hinkle

2023-05-01

Contents

Biology 395, Advanced Bioassessment | Everything will be in R for your reference! I will have this presentation, as well as a PDF version you'll be able to take with you.

Biology 395

Welcome to Biology 395, Advanced Bioassessment. In this course we'll:

- Review the basis of Bioassessment (Think conservation biology and genetics)
- Review Run off sources
- Review Water Quality monitoring its uses, and how to gather these data.
- Review Biological Monitoring methods.
- Review the basis of Indices of Biological Integrity
- Gather and analyze VSCI data from a local Appomattox tributary.
- eDNA now and in the future.

How will this class be formatted?

Mainly, you'll be taking a look at and reviewing my lectures outside of class in Canvas. Once you review those, you'll come to class and we'll practice what you've looked at in the lectures prior to class. There will be two exams, one lab report, and lots of small assignments based in R and that will total up your grade.

General outcomes:

-

How will this class be formatted? | Continued

Specific outcomes:

- Be able to import and manipulate data in R
- Assess a stream for physical habitat
- Assess a stream for water quality
- Perform Biological Monitoring
- Analyze these data in R
- Discuss the use case(s) of eDNA

Course Theory

This course is meant to be modeled after how a professional biologist's year. It will include a field season toward the beginning of the semester and will analyze data once the weather isn't so conducive to acquiring data.

August 21 | Preview of Course

August 21 | Preview of Course

```
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.2     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.0
## v ggplot2   3.4.2     v tibble    3.2.1
## v lubridate 1.9.2     v tidyr     1.3.0
## v purrr     1.0.1
## -- Conflicts ---------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
library(ggpubr)
library("FactoMineR")
library("factoextra")
```

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

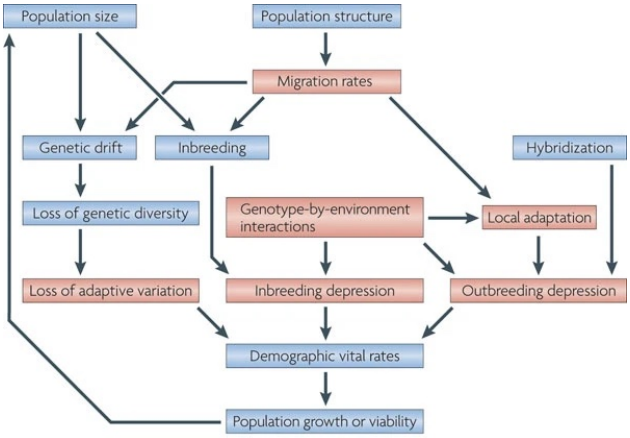
August 21 | Preview of Course

You'll see "chunks" like these throughout the course. This is actual R code, and is meant to be a guide/reference to you so you can take it with you and use as the basis of your own code in the future. These HTML slides are what you'll see in class, but I will give you a PDF document of all the code at the end of the semester. I can also give you particular lectures in PDF format upon your request.

August 21 | Preview of Course

Today, we're going to take a look at a brief overview of what's in the course starting with conservation biology/genetics.

```
knitr::include_graphics("Conservationgenetics_model.jpeg")
```

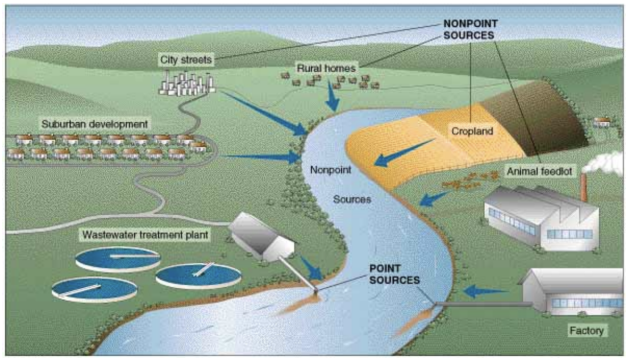



```
# ![](/Users/jamesonhinkle/Desktop/folders/Longwood/Bio395/Conservationgenetics_model.jpeg)
```

August 21 | Preview of Course

We'll look into the details of run off sources and how they're monitored, mainly in Virginia via the USGS and DEQ.

```
knitr::include_graphics("Runoff_sources.png")
```



August 21 | Preview of Course

We'll also talk in a lot of depth about biological monitoring. This is really at the center of the course as this completely relies on our ability to generate indices of biological integrity.

August 21 | Preview of Course

Speaking of Indices of Biological Integrity (IBI), we'll talk in depth about how these are generated, and how they're applied. Indices of biological integrity have been around now for decades. These include but are not limited to:

- Shannon Diversity Index
- Simpson Diversity Index

The links above are really just references for you. We'll talk in more detail about these later.

August 21 | Preview of Course

As indices developed, multimetric indices of biological integrity were developed and requested by the EPA so that states could have their own metrics and define what is impaired streams under the 303(d) section of the Clean Water Act.

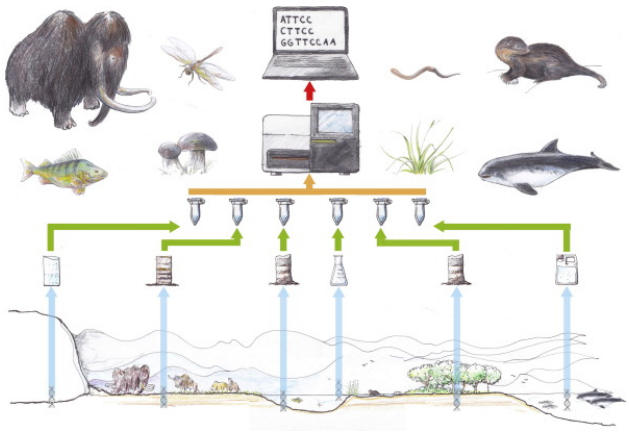
- Virginia developed it's own IBI known as the Virginia Stream Condition Index VSCI that we'll discuss in depth.
- A need was also identified for a multimetric index for streams that are not of a "higher gradient." Research to develop this index through the mid-2000s produced the Coastal Plain Macroinvertebrate Index CPMI

We will talk about both of these in detail and learn how to analyze basic VSCI and CPMI datasets in R.

August 21 | Preview of Course eDNA

eDNA or environmental DNA is a tool used in lieu of doing physical biologial assessment. It can be used to detect cryptic, invasive, or endangered species without harming them with any sort capture method. The method isolates DNA from a number of "matrices" namely: - water - soil - bone/excrement

```
knitr::include_graphics("edna_image.jpg")
```



August 21 | Preview of Course eDNA cont'd

Now, eDNA includes things a broad as metagenomics where we isolate DNA from many species in one substrate (mostly commonly, water) to look at what's present in entire ecological communities.

Good review examples:

- Willerslev et al 2015
- Willserlev et al 2015 (again)
- Lodge et al 2012

August 23 | Phab - Physical habitat assessment

August 23 | Physical habitat assessment

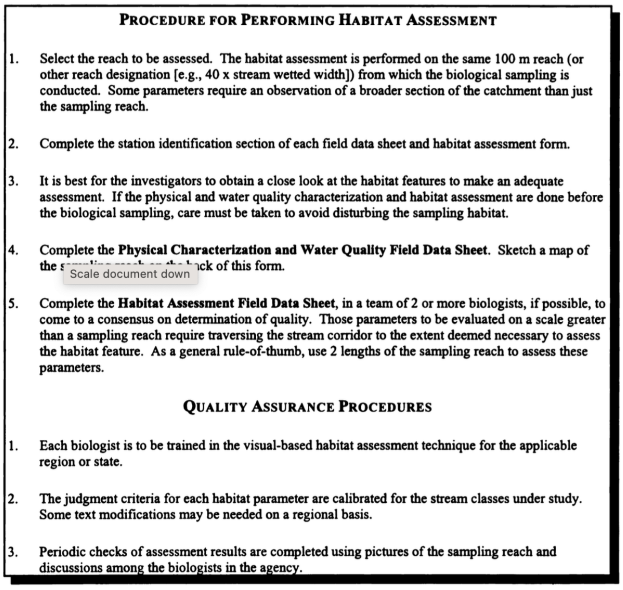
One of the first assessments necessary when working on a field site for any biomonitoring effort, is to assess physical habitat. The EPA developed rapid bioassessment tools that include physical habitat monitoring.

In normal site visits, you would look at each aspect of biomonitoring (Phab, fish, macroinvertebrates, water quality) all at the same time, but since we're learning we're mostly going to take it one at a time, starting with physical habitat.

August 23 | Physical habitat assessment

Subjectivity is a real issue here. As each scientist could assess physical habitat differently. The EPA as a result made a standarized protocol in an effort to decrease variability.

```
knitr::include_graphics("EPA_physicalhabitatprotocol.png")
```



PROCEDURE FOR PERFORMING HABITAT ASSESSMENT

1. Select the reach to be assessed. The habitat assessment is performed on the same 100 m reach (or other reach designation [e.g., 40 x stream wetted width]) from which the biological sampling is conducted. Some parameters require an observation of a broader section of the catchment than just the sampling reach.

2. Complete the station identification section of each field data sheet and habitat assessment form.

3. It is best for the investigators to obtain a close look at the habitat features to make an adequate assessment. If the physical and water quality characterization and habitat assessment are done before the biological sampling, care must be taken to avoid disturbing the sampling habitat.

4. Complete the **Physical Characterization and Water Quality Field Data Sheet**. Sketch a map of the s[Scale document down]ck of this form.

5. Complete the **Habitat Assessment Field Data Sheet**, in a team of 2 or more biologists, if possible, to come to a consensus on determination of quality. Those parameters to be evaluated on a scale greater than a sampling reach require traversing the stream corridor to the extent deemed necessary to assess the habitat feature. As a general rule-of-thumb, use 2 lengths of the sampling reach to assess these parameters.

QUALITY ASSURANCE PROCEDURES

1. Each biologist is to be trained in the visual-based habitat assessment technique for the applicable region or state.

2. The judgment criteria for each habitat parameter are calibrated for the stream classes under study. Some text modifications may be needed on a regional basis.

3. Periodic checks of assessment results are completed using pictures of the sampling reach and discussions among the biologists in the agency.

August 23 | Physical habitat assessment

I also HIGHLY recommend taking a look at at the Rapid Bioassessment Protocol from the EPA as well

August 23 | Physical habitat assessment

As a result, our aims in this presentation will be three fold:

- Review Physical habitat assessment methods
- Go into field and perform Physical habitat survey on our field site for the semester
- Analyze Physical habitat results from empirical state data.

This lecture is meant to review Physical habitat methods, and next week, we'll go into the field. When we introduce R, I'll show you how to import data, and then we'll do a brief "analysis."

August 23 | Physical habitat assessment

There are 10 Physical characteristics of streams we will look at. Each slide will represent one of the ten aspects, and we'll briefly discuss before going into the field. -Epifaunal Substrate and Available Cover.

```
knitr::include_graphics("epifaunalsubstrate:availablecover.png")
```

Habitat Parameter	Condition Category			
	Optimal	Suboptimal	Marginal	Poor
1. Epifaunal Substrate/ Available Cover **(high and low gradient)**	Greater than 70% (50% for low gradient streams) of substrate favorable for epifaunal colonization and fish cover; mix of snags, submerged logs, undercut banks, cobble or other stable habitat and at stage to allow full colonization potential (i.e., logs/snags that are not new fall and not transient).	40-70% (30-50% for low gradient streams) mix of stable habitat; well-suited for full colonization potential; adequate habitat for maintenance of populations; presence of additional substrate in the form of newfall, but not yet prepared for colonization (may rate at high end of scale).	20-40% (10-30% for low gradient streams) mix of stable habitat; habitat availability less than desirable; substrate frequently disturbed or removed.	Less than 20% (10% for low gradient streams) stable habitat; lack of habitat is obvious; substrate unstable or lacking.
SCORE	20 19 18 17 16	15 14 13 12 11	10 9 8 7 6	5 4 3 2 1 0

August 23 | Physical habitat assessment

- Embeddedness

```
knitr::include_graphics("embeddedness.png")
```

Habitat Parameter	Condition Category			
	Optimal	Suboptimal	Marginal	Poor
2.a Embeddedness **(high gradient)**	Gravel, cobble, and boulder particles are 0-25% surrounded by fine sediment. Layering of cobble provides diversity of niche space.	Gravel, cobble, and boulder particles are 25-50% surrounded by fine sediment.	Gravel, cobble, and boulder particles are 50-75% surrounded by fine sediment.	Gravel, cobble, and boulder particles are more than 75% surrounded by fine sediment.
SCORE	20 19 18 17 16	15 14 13 12 11	10 9 8 7 6	5 4 3 2 1 0

August 23 | Physical habitat assessment

- Pool Substrate Characterization

```
knitr::include_graphics("PoolSubstarte.png")
```

Habitat Parameter	Condition Category			
	Optimal	Suboptimal	Marginal	Poor
2b. Pool Substrate Characterization **(low gradient)**	Mixture of substrate materials, with gravel and firm sand prevalent; root mats and submerged vegetation common.	Mixture of soft sand, mud, or clay; mud may be dominant; some root mats and submerged vegetation present.	All mud or clay or sand bottom; little or no root mat; no submerged vegetation.	Hard-pan clay or bedrock; no root mat or submerged vegetation.
SCORE	20 19 18 17 16	15 14 13 12 11	10 9 8 7 6	5 4 3 2 1 0

August 23 | Physical habitat assessment

- Velocity Depth Combinations

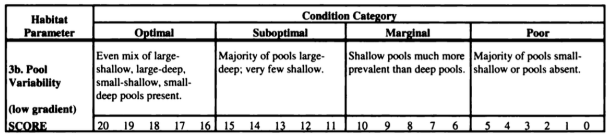
```
knitr::include_graphics("VelocityDepth.png")
```

Habitat Parameter	Condition Category			
	Optimal	Suboptimal	Marginal	Poor
3a. Velocity/ Depth Regimes **(high gradient)**	All 4 velocity/depth regimes present (slow-deep, slow-shallow, fast-deep, fast-shallow). (slow is <0.3 m/s, deep is >0.5 m)	Only 3 of the 4 regimes present (if fast-shallow is missing, score lower than if missing other regimes).	Only 2 of the 4 habitat regimes present (if fast-shallow or slow-shallow are missing, score low).	Dominated by 1 velocity/ depth regime (usually slow-deep).
SCORE	20 19 18 17 16	15 14 13 12 11	10 9 8 7 6	5 4 3 2 1 0

August 23 | Physical habitat assessment

- Pool Variability

```
knitr::include_graphics("Poolvariability.png")
```

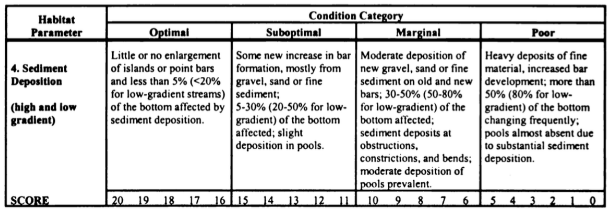


Habitat Parameter	Condition Category			
	Optimal	Suboptimal	Marginal	Poor
3b. Pool Variability (low gradient)	Even mix of large-shallow, large-deep, small-shallow, small-deep pools present.	Majority of pools large-deep; very few shallow.	Shallow pools much more prevalent than deep pools.	Majority of pools small-shallow or pools absent.
SCORE	20 19 18 17 16	15 14 13 12 11	10 9 8 7 6	5 4 3 2 1 0

August 23 | Physical habitat assessment

- Sediment Deposition

```
knitr::include_graphics("Sedimentdeposition.png")
```

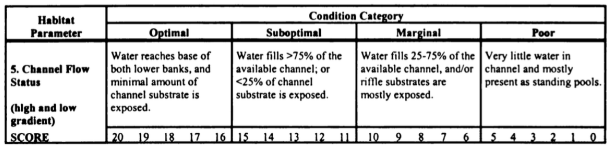


Habitat Parameter	Condition Category			
	Optimal	Suboptimal	Marginal	Poor
4. Sediment Deposition (high and low gradient)	Little or no enlargement of islands or point bars and less than 5% (<20% for low-gradient streams) of the bottom affected by sediment deposition.	Some new increase in bar formation, mostly from gravel, sand or fine sediment; 5-30% (20-50% for low-gradient) of the bottom affected; slight deposition in pools.	Moderate deposition of new gravel, sand or fine sediment on old and new bars; 30-50% (50-80% for low-gradient) of the bottom affected; sediment deposits at obstructions, constrictions, and bends; moderate deposition of pools prevalent.	Heavy deposits of fine material, increased bar development; more than 50% (80% for low-gradient) of the bottom changing frequently; pools almost absent due to substantial sediment deposition.
SCORE	20 19 18 17 16	15 14 13 12 11	10 9 8 7 6	5 4 3 2 1 0

August 23 | Physical habitat assessment

- Channel Flow Status

```
knitr::include_graphics("ChannelFlowStatus.png")
```

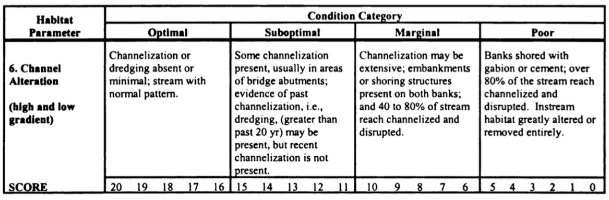


Habitat Parameter	Condition Category			
	Optimal	Suboptimal	Marginal	Poor
5. Channel Flow Status (high and low gradient)	Water reaches base of both lower banks, and minimal amount of channel substrate is exposed.	Water fills >75% of the available channel; or <25% of channel substrate is exposed.	Water fills 25-75% of the available channel, and/or riffle substrates are mostly exposed.	Very little water in channel and mostly present as standing pools.
SCORE	20 19 18 17 16	15 14 13 12 11	10 9 8 7 6	5 4 3 2 1 0

August 23 | Physical habitat assessment

- Channel Alteration

```
knitr::include_graphics("ChannelAlteration.png")
```



Habitat Parameter	Condition Category			
	Optimal	Suboptimal	Marginal	Poor
6. Channel Alteration (high and low gradient)	Channelization or dredging absent or minimal; stream with normal pattern.	Some channelization present, usually in areas of bridge abutments; evidence of past channelization, i.e., dredging, (greater than past 20 yr) may be present, but recent channelization is not present.	Channelization may be extensive; embankments or shoring structures present on both banks; and 40 to 80% of stream reach channelized and disrupted.	Banks shored with gabion or cement; over 80% of the stream reach channelized and disrupted. Instream habitat greatly altered or removed entirely.
SCORE	20 19 18 17 16	15 14 13 12 11	10 9 8 7 6	5 4 3 2 1 0

August 23 | Physical habitat assessment

- Frequency of Riffles

```
knitr::include_graphics("Frequencyriffles_bends.png")
```

Habitat Parameter	Condition Category			
	Optimal	Suboptimal	Marginal	Poor
7a. Frequency of Riffles (or bends) **(high gradient)**	Occurrence of riffles relatively frequent; ratio of distance between riffles divided by width of the stream <7:1 (generally 5 to 7); variety of habitat is key. In streams where riffles are continuous, placement of boulders or other large, natural obstruction is important.	Occurrence of riffles infrequent; distance between riffles divided by the width of the stream is between 7 to 15.	Occasional riffle or bend; bottom contours provide some habitat; distance between riffles divided by the width of the stream is between 15 to 25.	Generally all flat water or shallow riffles; poor habitat; distance between riffles divided by the width of the stream is a ratio of >25.
SCORE	20 19 18 17 16	15 14 13 12 11	10 9 8 7 6	5 4 3 2 1 0

August 23 | Physical habitat assessment

- Channel Sinuosity

```
knitr::include_graphics("ChannelSinusosity.png")
```

Habitat Parameter	Condition Category			
	Optimal	Suboptimal	Marginal	Poor
7b. Channel Sinuosity **(low gradient)**	The bends in the stream increase the stream length 3 to 4 times longer than if it was in a straight line. (Note - channel braiding is considered normal in coastal plains and other low-lying areas. This parameter is not easily rated in these areas.)	The bends in the stream increase the stream length 2 to 3 times longer than if it was in a straight line.	The bends in the stream increase the stream length 1 to 2 times longer than if it was in a straight line.	Channel straight; waterway has been channelized for a long distance.
SCORE	20 19 18 17 16	15 14 13 12 11	10 9 8 7 6	5 4 3 2 1 0

August 23 | Physical habitat assessment

- Bank Stability

```
knitr::include_graphics("BankStability.png")
```

Habitat Parameter	Condition Category			
	Optimal	Suboptimal	Marginal	Poor
8. Bank Stability (score each bank) **Note: determine left or right side by facing downstream** **(high and low gradient)**	Banks stable; evidence of erosion or bank failure absent or minimal; little potential for future problems. <5% of bank affected.	Moderately stable; infrequent, small areas of erosion mostly healed over. 5-30% of bank in reach has areas of erosion.	Moderately unstable; 30-60% of bank in reach has areas of erosion; high erosion potential during floods.	Unstable; many eroded areas; "raw" areas frequent along straight sections and bends; obvious bank sloughing; 60-100% of bank has erosional scars.
SCORE ___ (LB)	Left Bank 10 9	8 7 6	5 4 3	2 1 0
SCORE ___ (RB)	Right Bank 10 9	8 7 6	5 4 3	2 1 0

August 23 | Physical habitat assessment

- Bank Vegetatitive Protection

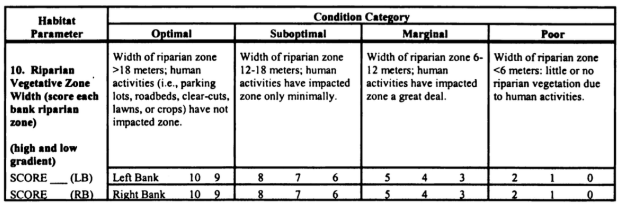
```
knitr::include_graphics("BankVegetativeProtection.png")
```

Habitat Parameter	Condition Category			
	Optimal	Suboptimal	Marginal	Poor
9. Vegetative Protection (score each bank) **Note: determine left or right side by facing downstream.** **(high and low gradient)**	More than 90% of the streambank surfaces and immediate riparian zones covered by native vegetation, including trees, understory shrubs, or nonwoody macrophytes; vegetative disruption through grazing or mowing minimal or not evident; almost all plants allowed to grow naturally.	70-90% of the streambank surfaces covered by native vegetation, but one class of plants is not well-represented; disruption evident but not affecting full plant growth potential to any great extent; more than one-half of the potential plant stubble height remaining.	50-70% of the streambank surfaces covered by vegetation; disruption obvious; patches of bare soil or closely cropped vegetation common; less than one-half of the potential plant stubble height remaining.	Less than 50% of the streambank surfaces covered by vegetation; disruption of streambank vegetation is very high; vegetation has been removed to 5 centimeters or less in average stubble height.
SCORE ___ (LB)	Left Bank 10 9	8 7 6	5 4 3	2 1 0
SCORE ___ (RB)	Right Bank 10 9	8 7 6	5 4 3	2 1 0

August 23 | Physical habitat assessment

- Riparian Vegetative Zone Width

```
knitr::include_graphics("RiparianBufferZone.png")
```



Habitat Parameter	Condition Category											
	Optimal			Suboptimal			Marginal			Poor		
10. Riparian Vegetative Zone Width (score each bank riparian zone) **(high and low gradient)**	Width of riparian zone >18 meters; human activities (i.e., parking lots, roadbeds, clear-cuts, lawns, or crops) have not impacted zone.			Width of riparian zone 12-18 meters; human activities have impacted zone only minimally.			Width of riparian zone 6-12 meters; human activities have impacted zone a great deal.			Width of riparian zone <6 meters: little or no riparian vegetation due to human activities.		
SCORE ___ (LB)	Left Bank	10	9	8	7	6	5	4	3	2	1	0
SCORE ___ (RB)	Right Bank	10	9	8	7	6	5	4	3	2	1	0

August 23 | Physical habitat assessment

The next steps will be actually acquiring these data at our field site (Buffal Creek - check). We'll gather the data for the 10 characteristics we discussed and talk about what we think is good and/or bad for QA/QC purposes.

After that, during our R introduction, I will attempt to have you perform a small analysis of Physical habitat data from Virginia DEQ to see if we can look at some basic statistics and perceive interesting about the data.

August 30 | Introduction to R

August 30 | Introduction to R

- R is both a language and an interface for statistical analysis, programming, and graphics. R has become a standard interface for statistical analysis in biological sciences due in part to its openness, ability to be extended by users, and it vibrant user base. As a statistical analysis platform, R has its own grammar and in this activity you will begin to understand how to use and interpret R.

To get R and Rstudio (you'll need both), you can go here

Please have this by next class (Aug 30). This will be one of your assignments.

August 30 | Introduction to R

- R itself consists of an underlying engine that takes commands and provides feedback on these commands. Each command you give the R engine is either an:
- Expression An expression is a statement that you give the R engine. R will evaluate the expression, give you the answer and not keep any reference to it for future use. Some examples include:

```
2 + 6
```

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

```
sqrt(5)
```

```
## [1] 2.236068
```

```
3 * (pi/2) - 1
```

```
## [1] 3.712389
```

August 30 | Introduction to R

-Assignment

An assignment causes R to evaluate the expression and stores the result in a variable. This is important because you can use the variable in the future. An example of an assignment is:

```r
x <- 2 + 6
myCoolVariable <- sqrt(5)
another_one_number23 <- 3 * (pi/2) - 1
x
```

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

```r
myCoolVariable
```

```
## [1] 2.236068
```

```r
another_one_number23
```

```
## [1] 3.712389
```

August 30 | Introduction to R

- Functions

There are thousands of potential functions in R and its associated packages. To use these functions, you need to understand the basic taxonomy of a function. A function has two parts: - A unique name, and - The stuff (e.g., variables) passed to it within the parentheses.

August 30 | Introduction to R

Not all functions need any additional variables. For example, the function ls() shows which variables R currently has in memory and does not require any parameters. If you forget to put the parentheses on the function and only use its name, by default R will show you the code that is inside the function (unless it is a compiled function). This is because each function is also a variable. This is why you should not use function names for your variable names (see below for more on naming).

August 30 | Introduction to R

To find the definition of a function, the arguments passed to it, details of the implementation, and some examples, you can use the ? shortcut. To find the definition for the sqrt() function type ?sqrt and R will provide you the documentation for that function.

August 30 | Introduction to R

Functions may have more than one parameter passed to it. Often if there are a lot of parameters given then there will be some default values provided. For example, the log() function provides logarithms. The definition of the log function show log(x, base=exp(1)) (say from ?log). Playing around with the function shows:

```r
log(2)
```

```
## [1] 0.6931472
```

```r
log(2, base = 2)
```

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

```r
log(2, base = 10)
```

```
## [1] 0.30103
```

August 30 | Introduction to R

R recognizes over a dozen different types of data. All of the data types are characterized by what R calls classes. To determine the type of any variable you can use the built-in function class(x). This will tell you what kind of variable x is. What follows are some of the more common data types.

August 30 | Introduction to R

- Numeric

Numeric types represent the majority of numerical valued items you will deal with. When you assign a number to a variable in R it will most likely be a numeric type. Numeric data types can either be displayed with or without decimal places depending if the value(s) include a decimal portion. In fact, R will make any assignment of a numerical value a numeric by default. For example:

August 30 | Introduction to R

```
x <- 4
class(x)
```

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

```
x
```

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

```
x <- numeric(4)
x
```

```
## [1] 0 0 0 0
```

August 30 | Introduction to R

```
x[1] = 2.4
x
```

```
## [1] 2.4 0.0 0.0 0.0
```

August 30 | Introduction to R

Notice this is an all or nothing deal here, each element of a vector must be the same type and the de- fault value for a numeric data types is zero. Also notice (especially those who have some experience in programming other languages) that dimensions in vectors (and matrices) start at 1 rather than 0. Operations on numeric types proceed as you would expect but since the numeric type is the default type, you don't really have to go around using the as.numeric(x) function. For example:

August 30 | Introduction to R

```
is.numeric(2.4)
```

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

```
as.numeric(2) + 0.4
```

```
## [1] 2.4
```

```
2 + 0.4
```

```
## [1] 2.4
```

August 30 | Introduction to R

- Numeric

Word of Caution, It is important to point out here that you need to be rather careful when dealing with floating point numbers due in part to the way in which computers store these numbers and how they are presented to us in the R interface as well as when we need to perform logical operations on them. Consider the following case. The ancient Egyptians had an approach to calculating pie as the ratio of 256/81.

```
e.pi <- 256/81
e.pi
```

```
## [1] 3.160494
```

August 30 | Introduction to R

- Numeric

Word of Caution cont'd: Very nice and apparently pretty close to 3.1416 so that they could get work done. Now, as we all know, the value of pie is the ratio of a circle's circumference to its diameter. We also know that it is a transcendeental number (e.g., on that cannot be produced using finite algebraic operations) and its decimal values never repeat.

```
print(e.pi, digits = 20)
```

```
## [1] 3.1604938271604936517
```

August 30 | Introduction to R

- Numeric

Word of Caution cont'd: There is another issue that you need to be careful with. You need to be considerate of how a computer stores numerical values. Consider the following:

```
x <- 0.3/3
x
```

```
## [1] 0.1
```

```
print(x, digits = 20)
```

```
## [1] 0.099999999999999991673
```

August 30 | Introduction to R

- Numeric

Why the difference? A computer deals in binary (0/1) representations and as such has a limited ability for precision, particularly for very large or very small numbers. Usually this does not cause much of a problem, but when you begin to work at crafting analyses, you should be aware of this drawback.

August 30 | Introduction to R

- Character

The character data type is the one that handles letters and letter-like representations of numbers. For example, observe the following:

```
x <- "If you can read this, you are beginning to take a step into a larger world."
class(x)
```

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

```
length(x)
```

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

August 30 | Introduction to R

- Character

Notice here how the variable x has a length of one, even though there are 37 characters within that string. If you want to know the number of characters, you need to use the nchar() function, otherwise it will tell you the 'vector length' (see below) of the variable.

```
y <- 23
class(y)
```

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

```
z <- as.character(y)
z
```

```
## [1] "23"
```

```
class(z)
```

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

August 30 | Introduction to R

- Character

Notice how the variable y was initially designated as a numeric type but if we use the as.character(y) function, we can coerce it into a non-numeric representation of the number. Combining character variables can be done using the paste() function to 'paste together' a string of char- acters (n.b., notice the optional sep argument).

August 30 | Introduction to R

```
w = "cannot"
x = "I"
y = "can"
z = "code in R"

paste(x, w, z)
```

```
## [1] "I cannot code in R"
```

```
paste(x, y, z)
```

```
## [1] "I can code in R"
```

August 30 | Introduction to R

- Constants

Constants are variables that have a particular value associated with them that cannot be changed. They are mostly here for convenience so that we do not have to go look up values for common things. Below are listed some common constants that you will probably encounter as you play with R.

August 30 | Introduction to R

- Constants

```
knitr::include_graphics("Constants_table.png")
```

Table 1: Common constants you will run across in R

Constant	Description
pi	The mathematical constant, π representing the ratio of a circles circumference to its diameter.
NULL	The absence of a type. This is the oubliette, complete nothingness, `/dev/null` Richmond on a Wednesday night... This is commonly used by functions that return undefined responses.
nan	Not a number.
Inf	Infinity (∞) as well as -Inf for $-\infty$.
NA	Typically used to represent something that is not there or missing. You can use it for missing data if you like.

August 30 | Introduction to R

- Constants

For the non-numerical constants, there are commands such as is.NULL(), is.nan(), is.infinite (and its cousin is.finite()), and is.na() to help you figure out if particular items are of that constant type if you like. At times this can be handy such when you have missing data and you want to set it to some meaningful value (e.g, is.na(X) <- 32 will set all N A values in X to 32). We'll get into this more in depth at a later time.

August 30 | Introduction to R

- Logical

Logical data types are boolean variables with a value of TRUE or FALSE. Obviously, these two values are the opposites of each other (e.g., not TRUE is FALSE, etc.). You will encounter logical data types in two primary situations;

- When you are writing a conditional statement that requires you to know the truth about something (e.g., if x == 0 you probably shouldn't try to divide by x because for some reason mathematicians haven't figured out how to divide by zero yet. . .), or
- If you are tying to select some subset of your data by using a particular condition (e.g.,select all entries where color == "blue").

The interesting thing about logical variables is that numbers can be coerced into a logical variable. For example the number zero, as an integer, numeric, complex, or raw data type, is considered to be FALSE whereas any non-zero value is considered TRUE.

August 30 | Introduction to R

- Factors

Factors are a particular kind of data that is used in statistics associated with treatments. You can think of a factor as a categorical treatment type that you are using in your experiments (e.g., Male vs. Female or Treatment A vs. Treatment B vs. Treatment C). Factors can be ordered or unordered depending upon how you are setting up you experiment. Most factors are given in as characters so that naming isn't a problem. Below is an example of five observa- tions where the categorical variable sex of the organism is recorded.

August 30 | Introduction to R

- Factors

```
sex <- factor(c("Male", "Male", "Female", "Female", "Unknown"))
levels(sex)
```

```
## [1] "Female"  "Male"    "Unknown"
```

```
table(sex)
```

```
## sex
##  Female    Male Unknown
##       2       2       1
```

```
sex[5] <- "Male"
```

August 30 | Introduction to R

- Factors

Here the table() function takes the vector of factors and makes a summary table from it. Also notice that the levels() function tells us that there is still an "Unknown" level for the variable even though there is no longer a sample that has been classified as "Unknown" (it just currently has zero of them in the data set).

August 30 | Introduction to R

- Collections

While alluded to previously, working with single numbers, factors, or local types are OK, but we often work with collections of data and R has some built-in objects that handle different assortments of basic data types.

August 30 | Introduction to R

A collection of several values, all of the same type, are held in a vector. In fact, all base data types can be created as vectors using the c() function (c for combine).

August 30 | Introduction to R

-Vectors

```
x <- c(1, 2, 3)

y <- c(TRUE, TRUE, FALSE)
y
```

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

```
z <- c("I", "can", "code", "in", "R")
z
```

```
## [1] "I"    "can"  "code" "in"   "R"
```

August 30 | Introduction to R

To access an element in a vector, R uses square brackets ([]) as demonstrated here:

```
x
```

```
## [1] 1 2 3
```

```
x[1] <- 2
x[3] <- 1
x
```

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

```
x[2]
```

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

August 30 | Introduction to R

Sequences of numbers are so common in analyses that there are several helper functions that assist you.

```
x <- 1:6
x
```

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

```
y <- seq(1, 6)
y
```

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

```
z <- seq(1, 20, by = 2)
z
```

```
##  [1]  1  3  5  7  9 11 13 15 17 19
```

```
rep(6, 4)
```

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

August 30 | Introduction to R

The notion x : y provides a vector of whole numbers from x to y. In a similar fashion the function seq(x,y,by=z) provides a sequence of numbers from x to y but can also have the optional parameter by= to determine how the sequence is made (in this case the by 2s for all the odd numbers from 1 to 20). The function rep(x,y) repeats x a total of y times.

August 30 | Introduction to R

- Matrices

Matrices are 2-dimensional vectors and can be created using the default constructor matrix() function. However, since they have 2-dimensions, you must tell R the size of the matrix that you are interested in creating by passing it a number for nrow and ncol for the number of rows and columns.

August 30 | Introduction to R

- Matrices

```
matrix(nrow = 2, ncol = 2)
```

```
##      [,1] [,2]
## [1,]   NA   NA
## [2,]   NA   NA
```

```
matrix(23, nrow = 2, ncol = 2)
```

```
##      [,1] [,2]
## [1,]   23   23
## [2,]   23   23
```

August 30 | Introduction to R

- Matrices Matrices can be created from vectors as well.

August 30 | Introduction to R

```
x <- c(1, 2, 3, 4)
x
```

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

```
is.vector(x)
```

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

```
is.matrix(x)
```

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

```
matrix(x)
```

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

```
y <- matrix(x, nrow= 2)

is.matrix(y)
```

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

```
is.vector(y)
```

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

August 30 | Introduction to R

There is a slight gotcha here if you are not careful.

```
x <- 1:4
matrix(x, nrow = 4, ncol= 2)
```

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

```
matrix(x, nrow = 3)
```

```
## Warning in matrix(x, nrow = 3): data length [4] is not a sub-multiple or
## multiple of the number of rows [3]
```

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

```
matrix(seq(1, 8), nrow = 4)
```

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

```
## [3,]    3    7
## [4,]    4    8
```

August 30 | Introduction to R

Notice here that R added the values of x to the matrix until it got to the end. However, it did not fill the matrix so it started over again. In the first case the size of x was a multiple of the size of the matrix whereas in the second case it wasn't but it still assigned the values (and gave a warning). Finally, as shown in the last case, if they are perfect multiples, then it fills up the matrix in a column-wise fashion. To access values in a matrix you use the square brackets just as was done for the vector types. However, for matrices, you have to use two indices rather than one.

August 30 | Introduction to R

```
X <- matrix(c(1, 2, 3, 4, 5, 6), nrow = 2)
X
```

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

```
X[1, 3]
```

```
## [1] 5
```

```
X[2,2] <- 3.2
X
```

```
##      [,1] [,2] [,3]
## [1,]    1  3.0    5
## [2,]    2  3.2    6
```

```
X[1, ]
```

```
## [1] 1 3 5
```

```
X[ , 3]
```

```
## [1] 5 6
```

August 30 | Introduction to R

The last two operations provide a hint as to some of the power associated with manipulating matrices. These are slice operations where only one index is given (e.g., X[1,]) provide a vector as a result for the entire row or column.

August 30 | Introduction to R

It is at this point that we get into some compounded information. R no longer just works in the space you were looking at above. It use to be useful to simply manipulate data in base R as you just saw, but it gets more and more cumbersome as your data complexity intensifies.

On top of that, R also uses "packages" to work with increasingly complex data to make it more computationally effecient as well as make it easier to look at visually.

August 30 | Introduction to R

Given that we'll be looking at decently large datasets by the end of the semester, and for your future adventures in R, it will be necessary to get some packages to help with data manipulation as well as visualization.

In our case we'll start with the following:

- Tidyverse
- ggpubr

August 30 | Introduction to R

I highly encourage you to implement Tidyverse with all your data. Lists and regular data.frames are kind of a think of the past. data.frames in Tidyverse are called "tibbles". "Tibbles" are essentially data frames with a meta data frame that runs in the background so it is easy for you to call data. Instead of having to call the specific row or column or call the row or column by $and the name with the data frame name every time; because of the meta data frame of a tibble, you can just call the variable. This is super useful with trimming data in various forms (filter, select, mutate, etc.).

August 30 | Introduction to R

Ok, so now that we've gone through everything necessary to get started, your assignment coming into class is to get as class to having R and Rstudio installed on your computer. If you get past that, then load in Tidyverse and ggpubr onto R in Rstudio via the install.packages(" ") command.

In class we will: - Make sure you're caught up on your R things from above - Take a look at some sample data for Phab from VADEQ. - Take a look at some of our data we just collected as practice.

August 30 | Introduction to R

- develop sample analysis from VADEQ data frame you already have.