1 Introduction to R, libraries, vectors, data frames, and functions

R is a computer language that is open-source and free, and in the last decade it has been widely adopted by scientists as a way to perform statistical analyses ranging from a basic t-test, to incredibly complex multivariate Bayesian modeling. One of the great advantages of R is that, unlike some other expensive statistical software, it is not a 'black box' that simply swallows your data and spits out an answer. It is a transparent coding procedure that can be shared and replicated, leaving a 'paper trail' in the form of code that allows you to see exactly how you get from dataset to statistical result.

The purpose of this workshop is to familiarize you with the basic format and capabilities of R, and point you in the right direction for future self-study[1]. This is not an exhaustive R course, and undoubtedly you will still have many questions at the end. However, it is my hope that learning some of the syntax, frequently used commands, and shortcuts will allow you to better determine what type of approach to coding in R will work best for you. It really is true that if you give 20 people a problem to work out in R, you'll end up with 20 different solutions.

- Advantages to using R: *free, flexible, well-maintained, transparent, easy to share, used in many fields*
- Disadvantages to using R: *steep initial learning curve, intimidating, unyieldingly precise*

1.1 Preparation

Before the workshop please download and install R and R Studio for your computer. Both are available for Windows, Mac, and Linux, and some OS even come with R preinstalled.

R is the architecture and language framework that does all the heavy lifting for your analyses. You can download it here:

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

For Windows follow the link and instructions for Windows. For Mac, follow the Mac link and download the file with the '.pkg' suffix on the left side to install. For Linux (Ubuntu) you should already have R preinstalled, if not, download it from the Ubuntu Software Center.

R Studio is a graphical user interface (GUI) that allows you to easily interact with, run, and save R coding scripts. You can download it here:

[1]A note: some of the data and text of this workshop is derived from a Biostatistics Course taught by Dr. Edd Hammill at the Bamfield Marine Sciences Centre in 2012.

https://www.rstudio.com/products/rstudio/download/#download

Just choose your OS and it should install with no problems. Congratulations!! YouâĂŹve just acquired the most powerful and versatile stats package in the world.

1.2 Goals of this Workshop

By the end of this workshop, you should have a basic understanding of the following procedures and techniques:

- loading datasets into R
- downloading and installing statistical/manipulative/graphical packages for R
- producing elegant(ish), well-annotated code
- manipulating datasets in R
- identifying the most parsimonious ways to enter data into spreadsheets for use in R
- performing basic statistical tests of data including t-tests and ANOVAs
- making simple visualization of data and results
- understanding what version control is and why it's important
- know where to look for help when you're stuck

1.3 R Interface Basics

Open R Studio. Your first reaction will most likely be overwhelming disappointment and a sense of âĂIJis that it?âĂİ when youâĂŹre presented withâĂę

R version 3.4.4 (2018-03-15) – ”Someone to Lean On” Copyright (C) 2018 The R Foundation for Statistical Computing Platform: x86_64-pc-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY. You are welcome to redistribute it under certain conditions. Type 'license()' or 'licence()' for distribution details.

Natural language support but running in an English locale

R is a collaborative project with many contributors. Type 'contributors()' for more information and 'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or 'help.start()' for an HTML browser interface to help. Type 'q()' to quit R.

>

Yes, that is it. You are currently looking at the âĂIJR consoleâĂİ. R is a programming package, you have to âĂIJtell it what to doâĂİ. Graphical user interfaces (GUIs) are available and are much more like the drop-down menu, point-and-click interface youâĂŹre used to, but they tend to be cumbersome and a pain. R works more like a very powerful calculator, you have to actually program the commands into it. As part of this, R has itâĂŹs own language, and during the rest of this course we will learn aspects of that language.

Another note from Ross: DOWNLOAD THE TIDYVERSE! https://www.tidyverse.org/ (the link above is primarily informative, you can download the packges in R by entering the command: install.packages('tidyverse') If this command throws you an error, download the most useful packages one at a time: install.packages('dplyr') install.packages('tidyr') install.packages('ggplot2')

Inputting commands into R There are 2 ways to input into R, these areâĂę. 1. Type things directly into the console, try this now, type âĂIJ2*4âĂİ and press return, or anything else you fancy. YouâĂŹll see it works a bit like a calculator. 2. Copy and paste from another window. This is a much better way of using R, and the habit I want you to get into. It means you have a record of everything youâĂŹve done that you can easily edit.

In R, click âĂIJFileâĂİ, and âĂIJNew DocumentâĂİ. YouâĂŹll now be presented with a basic text editor. You can type code into this editor, and easily copy it across to the R Console by pressing âĂIJCommand + ReturnâĂİ in a Mac, or âĂIJControl + RâĂİ in a PC.

Just try it with another simple command e.g. 4 times 5, then press enter, you should getâĂę

> 4*5 [1] 20

Believe it or not you just wrote and ran a program. That text file you just wrote in is now technically a piece of software known as a âĂIJscriptâĂİ. This one contains a very simple piece of code. Congratulations, youâĂŹve overcome the first step.

Functions and naming things Not to take away from your achievement, but inputting calculations one at a time into R isnâĂŹt really that useful. However, we can name things in R, and then use them again later. For example, say we want a vector of numbers between 1 and 10, and will call it âĂIJx.valuesâĂİ. R doesnâĂŹt like spaces, so we would input either..

> x.values<-c(1,2,3,4,5,6,7,8,9,10)

OrâĂę

> x.values<-c(1:10)

The âĂIJ<-âĂIJ in R basically means âĂIJisâĂİ, itâĂŹs naming something in a way that R can understand and find later. The âĂIJcâĂİ means âĂIJis a list of..âĂİ, and is a very useful command. You can name vectors, data frames, and functions (more on these later). Now type âĂIJx.valuesâĂİ and press enter, you should get

> x.values [1] 1 2 3 4 5 6 7 8 9 10

The [1] at the beginning is just telling us that weâĂŹre starting from the beginning (number 1) of the âĂIJx.valuesâĂİ. If the list goes over one line, R will give you another number in brackets telling what number the start of that line is.

Naming things in this way is useful as you can use them for other calculations.

For example, we could also produce the squares of our x.values by typing the following, the âĂIJâĂİ symbol means âĂIJto the power ofâĂİ

```
> y.values<-x.values2̂ > y.values [1] 1 4 9 16 25 36 49 64 81 100
```

We now have 2 vectors that R can understand. Of course what we named them wasnâĂŹt important, we specified that ourselves. These names are just a way for us to communicate with R.

If we want to plot the two vectors against each other, we can use the inbuilt function plot(). ThereâĂŹs a huge array of functions in R, in each case the function is carried out on whatever is in the parenthesis.

```
> plot(x.values,y.values)
```

This will produce a very basic plot of our 2 vectors. Within the plot function are many customisable elements called âĂIJargumentsâĂİ, to find them, we can start by using the incredibly useful âĂIJhelpâĂİ function.

```
> help(plot)
```

ThereâĂŹs all sorts to play with in there, donâĂŹt worry if you donâĂŹt understand it all, but as an example, lets make our plot a big red dashed lineâĂę

```
> plot(x.values,y.values,col=”red”,type=”l”,lty=3,lwd=3)
```

If we wanted to combine our 2 vectors into a data frame (similar to an excel spreadsheet in format) we simply use the âĂIJdata.frameâĂİ command.

```
> x.and.y<-data.frame(x.values,y.values) > x.and.y x.values y.values 1 1 1
2 2 4 3 3 9 4 4 16 5 5 25 6 6 36 7 7 49 8 8 64 9 9 81 10 10 100
```

These data frames are how we will be using most of our own data in R.

As well as the functions built within R, we can make our own. Say for example we wanted to make a function that worked out the square root of something, and then added 2. We would program it, and have to name it somethingâĂę

```
> my.function<-function(X)sqrt(X)+2
```

The parenthesis () are the thing to which the function is applied to, the curly brackets are the actual function. We can then apply this to a numberâĂę

```
> my.function(9) [1] 5
```

Or a vector, such as our âĂIJx.valuesâĂİ from before.

```
> my.function(x.values) [1] 3.000000 3.414214 3.732051 4.000000 4.236068
[6] 4.449490 4.645751 4.828427 5.000000 5.162278
```

You may need to produce your own functions later down the line in order to perform transformations in data. If you donâĂŹt ever need to make one, itâĂŹs worth knowing how theyâĂŹre used as it will give you a better understanding of how they come to be in R.

Task 1âĂę Combining some functions to do a basic test

The function âĂIrnormâĂİ produces a normal distribution, we can use it to make up simulate some data. For example if we wanted to simulate 10 data points, with a mean of 8 and a standard deviation of 3, we would input

```
> data.1<-rnorm(n=10,mean=8,sd=3) > data.1 [1] 11.694046 11.956468
11.174013 6.000238 1.365149 [5] 12.945742 6.089448 10.645392 11.408550 12.508761
```

DonâĂŹt worry if you donâĂŹt get the exact same numbers, R is simulating the data and so it will be different each time.

Now, a little test. I want you to produce 2 vectors with rnorm, call one âĂIJdata.1âĂİ and the other âĂIJdata.2âĂİ. The first will be 20 numbers long, have a mean of 15 and a standard deviation of 3. The second will be 15 numbers long, have a mean of 9, and a standard deviation of 2. Have look at your 2 vectors, calculate the means and standard deviations using the functions âĂIJmean()âĂİ and âĂIJsd()âĂİ.

Running a t-test Using âĂIJt.testâĂİ, see if there is a difference between your vectors ($p < 0.05$)? If you have trouble with âĂIJt.testâĂİ, try the help function then ask me. The t.test you are running is asking if there is a difference between the 2 groups.

The t-test tests the null hypothesis (H0) that there is no difference between the groups

The Alternative hypothesis (Ha) is that there is a difference between the groups.

Hints and tips for a better R life.

When producing R scripts, most people like âĂIJto annotateâĂİ their scripts to keep track of what theyâĂŹre doing. R doesnâĂŹt read anything in a line after a âĂIJ#âĂİ symbol, so you can use this feature to add in notes for future reference.

E.g. in your script could write

x.values<-c(2:15) ## Making a vector of x values

## now to make a list of y.values, these will be the ## x.value$\hat{3}$

y.values<-x.values$\hat{3}$

## Now lets plot that..

plot(x.values,y.values)

You can copy all of that into R in one go, and R will ignore all the stuff behind the #âĂŹs. This is very useful later on, and a good habit to get into.

Bringing a spreadsheet from excel to R Despite how wonderful R is for analysis, the fact remains that actually inputting data into a computer is best done in excel. This is the way in which most of you will be used to inputting data, and will do so in the future. Inputting into R is often a real hurdle and fraught with difficulty for a lot of people. There are multiple ways to achieve it, IâĂŹm teaching you the one I most often use. The key point is that the syntax, and file name have to be exactly right. There are a number of other extra things that can also go wrong, donâĂŹt worry if it doesnâĂŹt work immediately. WeâĂŹll go through it step by step.

WeâĂŹre going to use a small data set I collected in Costa Rica. The place we were staying in was full of scorpions, I decided to see if they were bigger in the kitchen or the bedroom. Each time we caught one in either room, we measured it then threw it the hell outside. The lengths are in âĂIJscorpion lengths.xlsâĂİ. WeâĂŹre going to import this data, obtain descriptive statistics (mean, standard deviation, standard error) then do a t.test again.

Important point 1 - In excel, save the spreadsheet as a .csv file. Excel asks you if you want to do this in the âĂIJsave asâĂİ menu.

The procedure is now different if youâĂŹre using a PC or a Mac, WeâĂŹll do PCs first.

PC procedure

In excel, save the file into a folder where you can easily find it, and with a name youâĂŹll recognise, e.g. âĂIJR stuffâĂİ in Documents, you will save all future data here.

In R, go to âĂIJFileâĂİ, click âĂIJChange directoryâĂİ, find the folder you just saved the file in, and choose this as your directory.

Now input the following line of code

> scorpion.data<-read.csv(âĂIJscorpion lengths.csvâĂİ,header=TRUE)

âĂIJread.csvâĂİ is a function like weâĂŹve used before, the âĂIJheader=TRUEâĂİ argument just tells R that the top row is the column names. What weâĂŹve done is brought the data in using the read.csv function, and named it scorpion.data. Type its name to have a look at it.

> scorpion.data bedroom kitchen 1 74 46 2 67 41 3 85 65 4 56 45 5 75 37 6 71 NA 7 67 NA

Mac procedure In a Mac the key initial step is to save the file in a folder in your home directory, detailed instructions are below.

In excel, open the scorpion lengths.xls file. Click âĂIJsave asâĂİ, thenâĂę.. 1. On the left of the screen, under âĂIJPLACESâĂİ click your home directory (the little house). 2. Click âĂIJNew FolderâĂİ, call it something like âĂIJr stuffâĂİ, you will save all future data in here. 3. Change the file format to âĂIJComma Separated Values (.csv) 4. Save the file

In R, input the following line of code.

> scorpion.data<-read.csv(" /r stuff/scorpion lengths.csv",header=TRUE)

Each of the âĂIJ/âĂİs means âĂIJopen thisâĂİ, and what you just typed in is known as a âĂIJfile pathâĂİ. Now just have a look at it to make sure itâĂŹs there.

> scorpion.data bedroom kitchen 1 74 46 2 67 41 3 85 65 4 56 45 5 75 37 6 71 NA 7 67 NA

Everyone back together again! Now run descriptive statistics and do the t-test

The problem is the things we want to test this time arenâĂŹt separate vectors as they were before, but columns in a data frame. We need to find away to tell R to look within the data frame. There are two basic ways to do this, each with advantages and disadvantages.

1. Telling R directly to look in the data frame

We can do this using âĂIJ, *thissymbolmeanstellsRtolookinside.Forexample*, *ifwewantedRtoshowusthelen*
> scorpion.data*bedroom*[1]74678556757167 ThatâĂŹs telling R that weâĂŹre interested in the âĂIJbedroomâĂİ column, which is inside the âĂIJscorpion.dataâĂİ data frame. You can the use these vectors as we did before for things like âĂIJ-plotâĂİ and all the rest of it.

You can now run descriptive statistics on these data by using commands such as

mean() # Calculates the mean of whatever is in the # brackets

sd() # Calculates the standard deviation of whateverâĂŹs in # the brackets length() # tells you the âĂIJlengthâĂİ of whateverâĂŹs in the # brackets, in this case the number of values we # have (n)

Task! Use the âĂIJ*methodtocalculatemeansandstandarddeviationsofthelengthsofscorpionsfrombothloc*
Generally, I prefer the âĂIJ$âĂİ method, although it requires more typing it avoids attaching data frames together, which can cause problems down the line.

2. Using the âĂIJattachâĂİ function

Rather than selecting columns, we can attach the data frame together, letting R know that weâĂŹre interested in using for now.

> attach(scorpion.data) ## Attaching the frame together

We can now run all the same commands as before, but donâĂŹt need to use the âĂIJ > t.test(kitchen,bedroom) Welch Two Sample t-test

data: kitchen and bedroom t = -4.0651, df = 7.65, p-value = 0.003964

The issue with this technique is that if we donâĂŹt âĂIJdetachâĂİ the data frame afterwards, it languishes around inside R and gets in the way. If we later try to refer to something with the same name as one of our columns, R gets confused and can start to compare the wrong things. This can lead to all kinds of frustration. So always detach when youâĂŹve finished with something.

> detach(scorpion.data) ## Make sure to do this!

When youâĂŹve got to this stage, youâĂŹve made some serious progress. YouâĂŹve overcome the first two big challenges of R, talking to it, and getting it to bring in data. At some point down the line, you will struggle to get R to bring in data frames. It happens to everyone, normally when someoneâĂŹs watching you type. DonâĂŹt worry, just work through it slowly, check the command and file path syntax, and it will all fall into place.

Before/after data and paired t-tests A wonderful feature of t-tests is their ability to deal with paired data, for example looking at before/after the application of a treatment, or looking at the same locations at different times.

For example, saw we wanted to ask whether the population density of Daphnia (a freshwater zooplankton species) differed between the summer and the winter. We could select 10 ponds, take a 1000ml sample of pond water, and count the number of individuals it contained. We may end up with the following dataâĂę

daphnia.csv

pond summer winter 1 30 13 2 54 45 3 63 51 4 51 43 5 32 21 6 36 30 7 42 35 8 48 37 9 39 31 10 61 51

We now have repeated measures of our 10 different locations, and could do a t-test as before

t.test(daphnia*summer*, *daphnia*winter)

Welch Two Sample t-test

data: daphnia*summeranddaphnia*winter t = 1.8354, df = 17.921, p-value = 0.0831

From this is appears as though thereâĂŹs no difference in Daphnia population sizes between the seasons. However, could we be missing something? Our data here are not technically a random sample, as weâĂŹre sampling from the same ponds multiple times, and weâĂŹd therefore expect two samples from the same pond to be more similar than two samples from different ponds. Ponds may differ in many important ways, for example some may contain fish that can consume Daphnia, some may contain more food for Daphnia than others. As a consequence, the differences between the ponds are so large, that the variance of the two samples becomes very large. The data does have a natural pairing however (each site was sampled twice), and we can use this pairing as a sort of natural control. What if, instead of asking if there was a difference between the samples, we took the difference between the summer and winter value for each ponds, and see if the overall difference is significantly different from 0. If it is different from 0, then weâĂŹre essentially saying there was a difference between the summer and winter sample.

Fortunately, we donâĂŹt need to do this by hand, as R can do the whole thing automatically. All we have to do is set the âĂIJpairedâĂİ argument in the t-test to TRUE.

t.test(daphnia*summer*, *daphnia*winter,paired=TRUE)

Paired t-test

data: daphnia*summeranddaphnia*winter t = 9.9611, df = 9, p-value = 3.696e-06

You can see that now the result is significant. We donâĂŹt have as many degrees of freedom as before, as technically weâĂŹve only got half as many data points (1 list of differences, instead of 2 lists of the actual values). Remember though, this technique can only be used when there is a natural pairing to the data.

âĂIJLong formâĂİ data in R So far weâĂŹve just looked at comparing 2 lists of data. It is however possible to use a slightly different method. Whenever weâĂŹre performing a statistical test, we have at least 2 variables, a response variable (the thing weâĂŹre interested in) and a descriptive variable (the thing we want to use to group the data). For example, say we wanted to see if there was a difference in height between men and women, âĂIJheightâĂİ would be our response variable, and âĂIJsexâĂİ would be our descriptive variable. If we had the data in a data frame named âĂIJheightsâĂİ, which contained 2 columns; âĂIJheightâĂİ âĂŞ the height in cm, and âĂIJsexâĂİ âĂŞ either male or female, overall the data would look likeâĂę

height sex 1 186 male 2 179 male 3 168 male 4 190 male 5 183 male 6 175 male 7 169 male 8 176 male 9 162 female 10 159 female 11 162 female 12 143 female 13 142 female 14 157 female 15 152 female

we could perform the following testâĂę.

t.test(height sex,data=height)

This would generate results identical in form to the ones we saw before..

Welch Two Sample t-test

data: height by sex t = -5.7667, df = 12.377, p-value = 7.909e-05

The âĂIJ âĂİ inside the t.test literally translates as âĂIJbyâĂİ, so with the command line weâĂŹre saying âĂIJdo a t.test looking at height differentiated by sexâĂİ. This method of using the âĂIJ âĂİ (called a âĂIJtildaâĂİ) is very common in R, and we shall be using it for the majority of the remaining tests.

It is very important as it allows you to perform tests using multiple descriptive variables, and have multiple levels within a variable. But more on that later.

Now try using the âĂIJ âĂİ method for yourself. Import the dataset âĂIJscorpion longâĂİ into R, and run the t.test, check you got the same result as when you ran the t.test before. You should do, youâĂŹre using the same data, itâĂŹs just organised differently.

T-test notes If youâĂŹre interested in how the t-test works, extensive information is available online. Essentially, it uses the following formula

Where the xâĂŹs are the means of the 2 samples, sx1x2 is the pooled standard deviation, and n is the total number of samples. We also need to know the number of degrees of freedom (the sample size n âĂŞ 1). We can then look up our value of t with the appropriate number of degrees of freedom in t-tables, and if it is greater than the critical value, we have a significant difference. For example, say we had 20 samples, and found a t value of 5.60, we would look up the critical value ($p < 0.05$) of t on 19 degrees of freedom (2.093). As ours is bigger, we have a significant difference. We would then report it in the following manner ($t(19) = 5.60$, $p < 0.05$). Of course, R does all this for us, and we just need to read the t-value, the degrees of freedom, and the p-value.

Task! Now youâĂŹve managed to bring in data sets and communicate with R, have a go at doing the following..

1. Bring in the data set âĂIJbromeliad.xlsâĂİ (consult the instructions above for all the steps!) 2. Make a plot with max.vol on the x-axis, and mosquitoes on the y. 3. Perform a t-test to see if the number of mosquitoes differed between plant species (the âĂIJspeciesâĂİ column in the dataset).

Introduction to ANOVA

We previously looked at using a t-test to ask whether two different groups of data are significantly different from each other. We tested the null hypothesis of no difference between groups. In our case we had scorpion lengths from two different rooms on a research station. However, often we have data from more than two different groups (for example, if weâĂŹd found scorpions in the kitchen, bedroom AND bathroom, heaven forbid). In this situation we may want to ask if any of them differ from each other, for this we need a slightly different test, called âĂIJthe Analysis of VarianceâĂİ, or ANOVA.

ANOVA is a very powerful, useful and versatile statistical test. It is at the heart of most of the statistics currently used, including (but not limited to) regression, linear modelling, non-linear modelling, mixed effects, MANOVA, analysis of covariance. As a result, you really want to understand it and know how to use it.

The first thing to understand, is what an ANOVA actually testsâĂę

Null hypothesis (H0) = There are no differences between the groups

Alternative hypothesis (HA) = at least one group is different from one other.

It is important to understand that the alternative hypothesis is not saying ALL groups are different from ALL others, just that at least one is different from one other. For example, if we had 3 groups; A, B, and C the alternative hypothesis (HA) isâĂę. AâĽăBâĽăC, or AâĽăB B=C but AâĽăC, or AâĽăB

A=C CâĽăB or AâĽăB AâĽăC but B=C etcâĂęâĂę.

In the t-test, the test relied on a

Formally, when we run an ANOVA, our groups are known as our treatment, and each of our groups are formally known as levels of the treatment. The actual ANOVA test calculates an f-statistic, and then looks at whether this value is greater than the critical value of f with the correct treatment degrees of freedom (k-1), and our error degrees of freedom ((n-k)-1). Of course, being wonderful, R does all this for you and you just have to know where each value is, and what it means.

For example, lets take another look at the bromeliad data.

data<-read.csv(" /bromeliads.csv",header=TRUE) data ### Taking a look at itâĂę

leaf.number diameter max.vol well.vol mosquitoes species location 10 58.5000000 450 45.00000 12 guzmania pitilla 10 63.5000000 900 90.00000 67 guzmania Monte.verde 12 87.0000000 200 16.66667 3 guzmania pitilla 13 113.5000000 875 67.30769 16 guzmania pitilla 13 93.0000000 500 38.46154 35 guzmania Monte.verde 14 60.0000000 450 32.14286 24 guzmania Monte.verde 15 72.0000000 200 13.33333 15 guzmania pitilla 15 80.0000000 1250 83.33333 113 verasia De.salva 16 112.5000000 1250 78.12500 95 verasia Monte.verde 16 96.0000000 1000 62.50000 100 verasia De.salva 16 84.0000000 1000 62.50000 110 verasia De.salva 18 87.5000000 1500 83.33333 102 verasia De.salva 19 74.0000000 750 39.47368 12 guzmania pitilla 20 89.5000000 1140 57.00000 32 guzmania Monte.verde 20 0.5288462 500 25.00000 36 guzmania Monte.verde 21 95.5000000 500 23.80952 3 guzmania pitilla 21 80.0000000 300 14.28571 15 guzmania pitilla 21 85.0000000 625 29.76190 88 verasia Monte.verde 21 135.5000000 3000 142.85714 148 verasia pitilla 22 107.5000000 1500 68.18182 105 verasia De.salva 25 95.0000000 1250 50.00000 83 verasia Monte.verde 25 91.5000000 3000 120.00000 152 verasia pitilla 26 115.0000000 1100 42.30769 121 verasia De.salva 27 84.5000000 750 27.77778 34 guzmania De.salva 27 98.0000000 750 27.77778 38 guzmania Monte.verde 27 87.0000000 1125 41.66667 120 verasia De.salva 34 69.0000000 1500 44.11765 52 guzmania Monte.verde 36 102.5000000 1250 34.72222 100 verasia De.salva 40 94.5000000 500 12.50000 6 guzmania pitilla 42 107.5000000 1250 29.76190 102 verasia De.salva

Last time, we used the column âĂIJspeciesâĂİ to run a t-test, but say for example we wanted to see if the number of mosquitoes differed between the 3 locations, âĂIJDe salvaâĂİ, âĂIJpitillaâĂİ, and âĂIJMonte verdeâĂİ. We now have to run an ANOVA as we have more than two groups, we do this using the âĂIJaovâĂİ command in R.

ANOVAs run slightly differently than t-tests, and we have to actually name the test something, weâĂŹll call it âĂIJanova.1âĂİ. As the data is in long form we have to use the âĂIJ âĂİ, because weâĂŹve not attached it we also use the âĂIJ.$Remember, Rwontreadanythingafterthe$# anova.1<-aov(data$mosquitoes\ data$location) ## test code

When you run the test, it doesnâĂŹt automatically give you the output, this is because youâĂŹve just produced the test within R, to look at the results, you need to use the âĂIJsummaryâĂİ command..

summary(anova.1)

This should then give you the following ANOVA tableâĂę

Df Sum Sq Mean Sq F value Pr(>F) data*location*220923104626.550.0048 ∗ ∗*Residuals*27431241597 − − − *Signif.codes* : 0 ∗ ∗ ∗ 0.001 ∗ ∗0.01 ∗ 0.05.0.11 This contains everything you need to report your result, R has calculated the f-statistic, the p-value, and all the degrees of freedom (Df) for you, and even told you the significance level.

In our case, âĂIJdata*locationwasourtreatment*, *andyoucanseefromthedegreesoffreedomcolumnthetreat*
2(*remember*, *treatmentdegreesoffreedomequalnumberoflevelsminusone*).*Ithasgivenusourerrordegreesoff*
To formally report this, we would write âĂIJWe reject the null hypothesis of no difference between mosquito abundances among the three locations (f(2,27) = 6.55, p = 0.0048, ANOVA)âĂİ. This one sentence conveys all the information we want to get across. Whenever you report the f

Post-hoc testing Although ANOVA is wonderful and allows us to test multiple different groups, it doesnâĂŹt actually tell us which groups are different from each other. As this is useful and interesting, we need a way to do it.

DONâĂŹT JUST DO A BUNCH OF T-TESTS We cannot however just do a series of three pairwise t-tests, this is very naughty. With a t-test weâĂŹre looking to be 95

1-(95

This means across our three tests, we have a 14.3

There is a solution, the Tukey test! The Tukey test is a legitimate post-hoc test we can use to look for all the differences between our groups, itâĂŹs conservative, and very commonly used and accepted. In R, itâĂŹs also very easy to run, we just use the âĂIJTukeyHSDâĂİ command on whatever we named our ANOVA test, in our case, âĂIJanova.1âĂİ

TukeyHSD(anova.1) ### Running a post-hoc Tukey test

This gives the following results tableâĂę

Tukey multiple comparisons of means 95

Fit: aov(formula = data*mosquitoes data*location)

‘*data*location‘ diff lwr upr p adj Monte.verde-De.salva -45.7 -90.01399 -1.386014 0.0422571 pitilla-De.salva -62.5 -106.81399 -18.186014 0.0045414 pitilla-Monte.verde -16.8 -61.11399 27.513986 0.6202390

Beautiful, just look at it. ItâĂŹs fitted our model (in the Fit: âĂIJaovâĂę.âĂİ Line) and then given us p-values for all of our location comparisons in the last column of the table. We can now just read these off and report them. We see Monte verde differed from De salva, pitilla differed from De salva, bit Pitilla didnâĂŹt differ from Monte verde.