Intro To R

Ross Whippo

2023/09/21

# 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 (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). 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

## 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:

* <https://posit.co/download/rstudio-desktop/>

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

## 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

## R Interface Basics

Open R Studio. Your first reaction will most likely be overwhelming disappointment and a sense of ‘is that it?’ when you’re presented with:

R version 4.3.1 (2023-06-16 ucrt) -- "Beagle Scouts"  
 Copyright (C) 2023 The R Foundation for Statistical Computing  
 Platform: x86\_64-w64-mingw32/x64 (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.  
  
 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 ‘R console’. R is a programming package, you have to ‘tell it what to do’. R works more like a very powerful calculator, you have to actually program the commands into it. As part of this, R has it’s own language, and during the rest of this course we will learn aspects of that language.

***A Note on ‘Tidyness’…***

*The current community standard among R programmers for formatting, manipulating, and visualizing data are found in the ‘Tidyverse’ package:* [*https://www.tidyverse.org/*](https://www.tidyverse.org/) *which is itself a collection of various packages.*

*I recommend becoming familiar with the Tidyverse from the very beginning of your R journey, as it provides not only tools, but a coherent and robust philosophy for how we should be working with data. It highlights and supports code clarity, reproducibility, and sharing, among other things*

*You can download the package in R by entering the following command in your R Studio console:*

``install.packages('tidyverse')``

*It may take a while to fully download and install, so do this when you have some free time. If the download ends up throwing you an error code, you can also download the most useful (in my opinion) packages one at a time:*

install.packages('readr')

install.packages('tibble')

install.packages('dplyr')

install.packages('tidyr')

install.packages('ggplot2')

*OK, back to the lesson…*

### Inputting commands into R

There are 2 ways to input into R, these are:

1. Type things directly into the console, try this now, type:  
   2\*4  
   and press return. You’ll see it works a bit like a calculator.
2. Create a new ‘R Script’ and run your code from there.

Creating an ‘R Script’ is a much better way of using R, and a habit you should get into. It allows you to save your work as you go, and keep a record of everything you’ve done that you can easily edit. You can create a new R Script by going to the upper left hand corner of R Studio and clicking on the white square with the ‘plus’ sign in a green circle. A dropdown menu should appear and you can click on ‘R Script’. You now have a what is essentially a ‘text box’ that you can save and use to create and run your code!

You can type code into this editor, and easily run the code with the R Console by pressing ‘Command + Enter’ on a Mac, or ‘Control + Enter’ on a PC.

Just try it with another simple command. Type

4\*5

into your new R Script and with your cursor on the same line as the code press ’Command (or) Control + Return. You should see the output:

> 4\*5 [1] 20

Believe it or not you just wrote and ran a program. That text you just wrote in is now technically a piece of software known as a ‘script’. This one contains a very simple piece of code. Congratulations, you’ve taken the first step.

### Functions and naming things

Not to take away from your achievement, but inputting calculations one at a time into R isn’t 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 ‘x.values’. R doesn’t like spaces in named objects, so we would input either:

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

Or:

x.values <- c(1:10)

You should see text pop up in the upper right corner of your R Studio console that looks like:

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

This means you have saved that list of values. The ‘<-’ in R basically means ‘is’, it’s naming something in a way that R can understand and find later. The ‘c’ means ‘is a list of..’, and is a very useful command. The colon indicates a range of values. You can name vectors, data frames, and functions (more on these later). You can also type ‘x.values’ in your script or console 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 we’re starting from the beginning (number 1) of the ‘x.values’. 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 ‘^’ symbol means ‘to the power of’

y.values <- x.values^2

You will now see new text pop up in the upper right corner under your x.values object that looks like this:

y.values num [1:10] 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 wasn’t 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(). There’s 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 customizable elements called ‘arguments’, to find them, we can start by using the incredibly useful ‘help’ function.

help(plot)

There’s all sorts to play with in there, don’t worry if you don’t understand it all, but as an example, lets make our plot a big red dashed line:

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

It’s important to note here that we have been using ‘base R’, that is, the packages that come pre-installed with every R version. Now that you have a general idea of how R syntax works, we will begin using commands from the Tidyverse package, which are more robust (for the reasons listed near the beginning of this document).

Although you have the Tidyverse package installed in R, you must initialize it every time you start a new R session. To do this you use the library() command. For Tidyverse it looks like this:

library(tidyverse)

You should see an output that looks something like this:

── Attaching core tidyverse packages─────────────────────────────tidyverse 2.0.0 ── ✔ dplyr 1.1.2 ✔ readr 2.1.4 ✔ forcats 1.0.0 ✔ stringr 1.5.0 ✔ ggplot2 3.4.3 ✔ tibble 3.2.1 ✔ lubridate 1.9.2 ✔ tidyr 1.3.0 ✔ purrr 1.0.2 ── Conflicts ────────────────────────────────────────── tidyverse\_conflicts() ──``✖ dplyr::filter() masks stats::filter() ✖ dplyr::lag() masks stats::lag() ℹ Use the conflicted package to force all conflicts to become errors

Don’t worry, you did it right!

Now, If we wanted to combine our 2 vectors into a data frame (similar to a spreadsheet in format) using tidyverse we simply use the tibble() command.

x.and.y <- tibble(x.values, y.values)

You will now see text pop up in the upper right hand corner under a new ‘Data’ heading. This indicates that you have created an object that has both rows and columns (as opposed to just a list of numbers). Click on the x.and.y object in the Data window to view it. 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 something:

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 number:

my.function(9)

Or a vector, such as our ‘x.values’ from before.

my.function(x.values)

You may need to produce your own functions later down the line in order to perform transformations in data. If you don’t ever need to make one, it’s worth knowing how they’re used as it will give you a better understanding of how they come to be in R.

#### Task 1: Combining some functions to do a basic test

The function ‘rnorm’ 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)

To view your results type

data.1

into your script and press Command (or) Control + Enter. You should see a list of your numbers appear in the Console.

Don’t worry if you don’t get the exact same numbers if you run the rnorm command multiple times with the same parameters, 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 ‘data.1’ and the other ‘data.2’. 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 ‘mean()’ and ‘sd()’.

Running a t-test Using ‘t.test’, see if there is a difference between your vectors (p < 0.05)? If you have trouble with ‘t.test’, 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 ‘to annotate’ their scripts to keep track of what they’re doing. R doesn’t read anything in a line after a ‘#’ 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 making a list of y values which will be x.value^3y.values <- x.values^3# now plot the resultsplot(x.values, y.values)

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

### Working with imported data

Despite how wonderful R is for analysis, the fact remains that actually inputting data into a computer is best done in a spreadsheet. 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, I’m 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, don’t worry if it doesn’t work immediately. We’ll go through it step by step.

We’re going to use a small data set my friend Edd collected in Costa Rica. The place he was staying in was full of scorpions, he decided to see if they were bigger in the kitchen or the bedroom. Each time he caught one in either room, he measured it then threw it the hell outside. The lengths are in ‘scorpion\_lengths.csv’ (*Note: a ‘.csv’ file is a ‘comma separated file’. It is generally readable without specific spreadsheet software like excel, though excel can read it.* ) We’re going to import this data, obtain descriptive statistics (mean, standard deviation, standard error) then do a t.test again.

#### Import procedure

In your spreadsheet software, save the file into a folder where you can easily find it, and with a name you’ll recognize, e.g. ‘R stuff’ in Documents, you will save all future data here.

In R, go to Session>Set Working Directory>Choose Directory and navigate to 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('scorpion\_lengths.csv')

‘read\_csv’ is a function like we’ve used before. What we’ve done is brought the data in using the read\_csv function, and named it scorpion.data. Click on it in your Data window to view it.

Now run descriptive statistics and do the t-test

(Hint: the $ operator is a way to separate out columns into individual lists or vectors of numbers. It essentially means ‘in this column’. For example, type out

x.and.y$x.values

and observe the output.

There are many different ways to formulate t-test within R, but I will show a method that I use to make sure my code is as clear as possible. It starts with a reformatting of the data (it’s worth thinking about how you enter you data into your spreadsheet before you start, in order to minimize manipulations you’ll have to do in R.

For this, I will introduce what is called the ‘pipe’. It is typically represented by the symbols %>% and is a unique operator within the Tidyverse. It allows you to rearrange the arguments within a function and to combine functions so they are more ‘readable’ by human eyes. For example, it’s possible to find the mean and standard deviation of your x.and.y values with the commands:

mean(x.and.y$x.values)mean(x.and.y$y.values)

and

sd(x.and.y$x.values)sd(x.and.y$y.values

But this is rather wordy. With the %>% operator, (and properly formatted data) you can put all these things together in an intuitive way:

xy\_long <- x.and.y %>% pivot\_longer(everything())

Now the names of the vectors that the numbers came from are all in one column, and the values associated with each vector are in another. From here, we can summarise the dataset quickly:

xy\_long %>% group\_by(name) %>% summarise(mean(value), sd(value))

Likewise, we can now run a more clean looking t-test:

xy\_long %>% t.test(value ~ name, data = .)

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 whatever's in # the brackets

length() # tells you the 'length' of whatever's in the # brackets, in this case the number of values we # have (n)

##### Task!

Use the ‘$’ method to calculate means and standard deviations of the lengths of scorpions from both locations. Then, do a t.test comparing the lengths of the scorpions. If you manage this easily, write a function to calculate the standard error, and run it on the scorpion data.

THEN…

make the same calculations above using the pipe %>% operator.

When you’ve got to this stage, you’ve made some serious progress. You’ve 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 someone’s watching you type. Don’t 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 data:

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(daphniawinter)

Welch Two Sample t-test

data: daphniawinter t = 1.8354, df = 17.921, p-value = 0.0831

From this is appears as though there’s 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 we’re sampling from the same ponds multiple times, and we’d 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 we’re essentially saying there was a difference between the summer and winter sample.

Fortunately, we don’t need to do this by hand, as R can do the whole thing automatically. All we have to do is set the ‘paired’ argument in the t-test to TRUE.

t.test(daphniawinter,paired=TRUE)

Paired t-test

data: daphniawinter t = 9.9611, df = 9, p-value = 3.696e-06

You can see that now the result is significant. We don’t have as many degrees of freedom as before, as technically we’ve 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.

‘Long form’ data in R So far we’ve just looked at comparing 2 lists of data. It is however possible to use a slightly different method. Whenever we’re performing a statistical test, we have at least 2 variables, a response variable (the thing we’re 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, ‘height’ would be our response variable, and ‘sex’ would be our descriptive variable. If we had the data in a data frame named ‘heights’, which contained 2 columns; ‘height’ - the height in cm, and ‘sex’ - either male or female, overall the data would look like:

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 test:.

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 ‘~’ inside the t.test literally translates as ‘by’, so with the command line we’re saying ‘do a t.test looking at height differentiated by sex’. This method of using the ‘~’ (called a ‘tilda’) 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 ‘~’ method for yourself. Import the dataset ‘scorpion long’ into R, and run the t.test, check you got the same result as when you ran the t.test before. You should do, you’re using the same data, it’s just organised differently.

T-test notes If you’re interested in how the t-test works, extensive information is available online. Essentially, it uses the following formula

Where the x’s 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 â 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 you’ve managed to bring in data sets and communicate with R, have a go at doing the following..

1. Bring in the data set 'bromeliad.xls' (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 'species' 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 we’d 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 ‘the Analysis of Variance’, 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 tests:

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) is:. A>B>C, or A>B B=C but Aâ C, or Aâ B A=C Câ B or Aâ B Aâ C but B=C etc::.

In the t-test, the test relied on a t-statistic, which had a number of degrees of freedom (n-1). In the ANOVA, it uses a slightly different statistic, the f-statistic, which has it’s own tables. The number of degrees of freedom are slightly different as well because we have not only the number of samples (n), but also the number of groups (k). We have to take both these into account.

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 it:

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 ‘species’ to run a t-test, but say for example we wanted to see if the number of mosquitoes differed between the 3 locations, ‘De salva’, ‘pitilla’, and ‘Monte verde’. We now have to run an ANOVA as we have more than two groups, we do this using the ‘aov’ command in R.

ANOVAs run slightly differently than t-tests, and we have to actually name the test something, we’ll call it ‘anova.1’. As the data is in long form we have to use the ‘~’, because we’ve not attached it we also use the ‘$’. Remember, R won’t read anything after the ‘#’ $ anova.1<-aov(datalocation) ## test code

When you run the test, it doesn’t automatically give you the output, this is because you’ve just produced the test within R, to look at the results, you need to use the ‘summary’ command..

summary(anova.1)

This should then give you the following ANOVA table:

Df Sum Sq Mean Sq F value Pr(>F)

data$location 2 20923 10462 6.55 0.0048 **Residuals 27 43124 1597**  
**— Signif. codes: 0 â**\*’ 0.001 â\*\*’ 0.01 â\*’ 0.05 â.’ 0.1 â ’ 1 $ 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, ‘data$location’ was our treatment, and you can see from the degrees of freedom column the treatment df=2 (remember, treatment degrees of freedom equal number of levels minus one). It has given us our error degrees of freedom (Although R calls these ‘Residuals’, just to be difficult). So how do we report this, bearing in mind what our original question was; are there differences in mosquito abundance between the locations? $ To formally report this, we would write ‘We reject the null hypothesis of no difference between mosquito abundances among the three locations (f(2,27) = 6.55, p = 0.0048, ANOVA)’. This one sentence conveys all the information we want to get across. Whenever you report the f­-statistic you must report the degrees of freedom in the brackets in the following order; treatment df first, then error df.

Post-hoc testing Although ANOVA is wonderful and allows us to test multiple different groups, it doesn’t actually tell us which groups are different from each other. As this is useful and interesting, we need a way to do it.

DON’T 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 we’re looking to be 95% confident that there is a difference between the groups, we have a 5% chance of finding a difference when it isn’t there. Essentially, we’ve a 5% chance of being wrong. However, if we run three of tests together, we’re inflating the chance of finding at least one difference because due to probability theory our chances of being wrong at least once is:

1-(95%*95%*95%) = 1-85.7% = 14.3%

This means across our three tests, we have a 14.3% chance of finding at least one significant difference when it doesn’t exist. If we had 4 groups, and did all the pairwise t-tests (6 in total) our chance of being wrong increases to 26.5%, and it gets even worse every time you add another group.

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, it’s conservative, and very commonly used and accepted. In R, it’s also very easy to run, we just use the ‘TukeyHSD’ command on whatever we named our ANOVA test, in our case, ‘anova.1’

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

This gives the following results table:

Tukey multiple comparisons of means 95% family-wise confidence level

Fit: aov(formula = datalocation)

$`data$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. It’s fitted our model (in the Fit: ‘aov:.’ 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 didn’t differ from Monte verde.

summary(cars)

## speed dist   
## Min. : 4.0 Min. : 2.00   
## 1st Qu.:12.0 1st Qu.: 26.00   
## Median :15.0 Median : 36.00   
## Mean :15.4 Mean : 42.98   
## 3rd Qu.:19.0 3rd Qu.: 56.00   
## Max. :25.0 Max. :120.00

## Including Plots

You can also embed plots, for example:



Note that the echo = FALSE parameter was added to the code chunk to prevent printing of the R code that generated the plot.