Stats with Sparrows - 01

Julia Schroeder

2022

Introduction to Statistics with Sparrows and Welcome!

Learning aims

Welcome to Stats with Sparrows. Make sure you attend the welcome session, or watch the recorded version, to get up to speed to what this week will be about. The learning aims of this week are to: - Compute and interpret descriptive statistics - Compute and interpret parametric tests, in particular t-test and linear models - Understand how to interpret fixed effect interactions - Get an understanding for how to match your hypothesis with an appropriate model - Know how to report methods and results in a scientific paper

If you read through this, you might have noticed that the focus is on interpreting - not mathematics. As you are, or aspire to become, biologists, it is of utmost importance to know how to choose, and interpret statistics that are fitting to your biological question. Oftentimes, either people do not understand the statistics and simply report back numbers without ever having engaged with the values, what they really mean. On the other end of the spectrum, some people do statistics because they like the complexity, and the math, but then do not link the results back to the biology. Both approaches are not right - you need to know what statistic to use for what question, why, and how to interpret the result. You will be surprised to learn that you do not need to remember, or know, the math behind it to be a good biologist.

Following this philosophy, this week will be on applied statistics, and sometimes we will brush over the mathematical details. If you are interested in more, the recommended reading for this week will get you on the way there. Other times, however, it is helpful to know the mathematical logic behind a certain test. In that case, I will try to break it down as simple as possible. Thus, in short, this course aims at providing you with the tools, knowledge, and knowledge where to ask for help, to use the appropriate statistics for your biological question.

How to work through this course

The course is based on lectures, hand-out practicals (HOs), and group learning. Attend the lectures before you embark on the HO, - some information is only in one but not the other, - to graduate from the course you need both. Some lectures have no HO associated. Some, but not all, HOs have exercises that you are supposed to work on. These might require you to write a short methods or results section, or to write and annotate an R script, or to prepare a presentation, either every student by themselves, or in a group. Some exercises also ask for you to discuss things in a group – chat with the student sitting at your table or

with your neighbours. Many of these exercises are like model exam questions and are good practice.

Hand-outs and help sessions

After attending the lecture, return to your HO and work through the rest of it. Make sure to create a new R script (.R) for each day, and do not simple copy-paste from this handout but rather try to think about what the code does. The GTAs and myself to be able to help you. We expect you to write such code by yourself every day.

Often code builts on itself, it is imperative that you solve a problem before you can continue, therefore, I suggest you work with your GTA through issues during the synchronous help sessions if you feel your skills require a lot of help.

Here is a list of things you can do to solve problems. Here is a list for you to attempt to problem solve (and if this doesn't do it, don't hesitate to wave to any of our staff for help!):

1) Most errors are simply typos, and sometimes they are hard to find. To solve it, try to re-type it and this time pay good attention. Sometimes re-typing a line makes it easier to find the typo.
2) When something does not work, run

```
rm(list=ls())
```

   at least once. Often we forget cleaning the workspace before a session. Sometimes we've done this but have loaded up variables with the same name and that is what messes things up. Even if you have no clue what could be in the global workspace, do clean your environment when you run across an error.

3) Check your objects. The variable you are trying to change, or run a model with. Look at the variable with various functions - dir(), summary(), table(), ect. Sometimes we don't even notice that we are handling, e.g. an empty vector. Double and triple check all data involved in the code that doesn't work.
4) Google the function. This is the most important advice we can give you. There are tons of information and help available online. It is common procedure for anyone using R to regularly google functions one has never used before or forgot how to use.
5) If you use complex code with more than one operation in one go - break it down into smaller parts and run each separate part as stand-alone. (
6) For running algorithms using multiple iterations, loops ect: Use a break point or print to print out the current value.

Only when the above do not apply, ask for help in one of our Q&A sessions. And then provide information to the script that you are asking about and what it is named, so I can find it on R Studio Cloud.

Linearity

The HOs built on each other. Even if you think you know a topic, I advise you to work through the respective HO - you can test your knowledge using the exercise section. HOs

and tasks also give you good guidelines for how to write the statistics part of your thesis, how to choose what data to collect and exam practise.

BYO

At the end of this course, we might have a BYO session. This is typically where students send me data they have worked on in the past, e.g. for their undergrad degree, and that they would like to see re-analysed. We will use these data and distribute it among groups, with the biological question, and the groups then will find different ways of analyzing it. I will also provide a solution. If you think you have appropriate data that you do not mind sharing, please email me until latest Wednesday noon. In the past, students found these BYO sessions particularly helpful, so please come forward – if nobody comes forward we cannot do these.

Revisiting what R can do for you

In this practical, you will practice using the R command line interface to do some basic stats look at the data structure. First, we revisit what you learned last week to make sure you can use these skills.

A lot of this is revisiting last week. Start up R or RStudio, or whatever GUI you wish. Remember, you can download R from the Comprehensive R Archive Network (http://cran.r-project.org/).

So let's start now!

Some good house-keeping: Cleaning your environment!

```
rm(list=ls())
```

Checking and re-setting your working directory:

```
getwd()
setwd("/cloud/project/SwS01Workspace")
getwd()
```

Remember to always comment on your code:

```
getwd()                    # check which working directory we're in
setwd("/cloud/project/SwS01Workspace")# re-set workign directory to
getwd()                    # check that what we did worked
```

Remember you can use basic calculator commands in R:

```
2*2+1
```

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

```
2*(2+1)
```

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

```
12/2^3
```

```
## [1] 1.5
```

```
(12/2)^3
```

```
## [1] 216
```

Remember you can assign values to variables:

```
x <- 5
x
```

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

```
y <- 2
y
```

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

```
x2 <- x^2
x2
```

```
## [1] 25
```

```
x
```

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

```
a<-x2+x
a
```

```
## [1] 30
```

```
y2 <- y^2
z2 <- x2 + y2
z <- sqrt(z2)
print(z)
```

```
## [1] 5.385165
```

There are logical tests:

```
3>2
```

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

```
3 >= 3
```

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

```
4<2
```

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

You can create vectors of different formats:

```
myNumericVector <- c(1.3,2.5,1.9,3.4,5.6,1.4,3.1,2.9)
myCharacterVector <- c("low","low","low","low","high","high","high","high")
myLogicalVector <- c(TRUE,TRUE,FALSE,FALSE,TRUE,TRUE,FALSE,FALSE)
```

We will use the str() function to get the structure of any variable.

```
str(myNumericVector)

##  num [1:8] 1.3 2.5 1.9 3.4 5.6 1.4 3.1 2.9

str(myCharacterVector)

##  chr [1:8] "low" "low" "low" "low" "high" "high" "high" "high"

str(myLogicalVector)

##  logi [1:8] TRUE TRUE FALSE FALSE TRUE TRUE ...
```

It is important for statistical reasons, to always know what kind of data your variable is - numeric, vector, ect. If you add values of different categories to one variable, you will create a mixed variable and R will coerce (force) it into one of the most basic mode that covers all:

```
myMixedVector <-c(1, TRUE, FALSE, 3, "help", 1.2, TRUE, "notwhatIplanned")
str(myMixedVector)

##  chr [1:8] "1" "TRUE" "FALSE" "3" "help" "1.2" "TRUE" "notwhatIplanned"
```

We install packages, and load them. Can you find out what the difference is between the library and the require command?

```
install.packages("lme4")
library(lme4)
require(lme4)
```

When we are lost, we ask R for help:

```
help(getwd)

help(log)
```

There are some special notation in R.

```
sqrt(4); 4^0.5; log(0); log(1); log(10); log(Inf)

## [1] 2

## [1] 2

## [1] -Inf

## [1] 0
```

```
## [1] 2.302585

## [1] Inf
```

Inf means infinity. You can specify if it's negative or positive. In R, *e* is typed in using an R function exp and we should type in exp(1). For π, just type in pi.

```
exp(1)

## [1] 2.718282

pi

## [1] 3.141593
```

R shows up to 6 decimal places, the default is to show 7 digits. To what precision should you report decimal places in your reports?

Clear your workspace at the beginning of each session, and between different parts of a project.

```
rm(list=ls())
```

Entering data

We can enter data in many ways, and you will have learned last week how to do that. Here, we will mostly simply read tables with headers:

```
d<-read.table("SparrowSize.txt", header=TRUE)
str(d)

## what you see below might differ from what you will see in your r console -
## it differs by R version. Don't be distracted by these.

## 'data.frame':    1770 obs. of  8 variables:
##  $ BirdID: int  1 2 2 2 2 2 2 2 2 2 ...
##  $ Year  : int  2002 2001 2002 2003 2004 2004 2004 2004 2004 2005 ...
##  $ Tarsus: num  16.9 16.8 17.2 17.5 17.8 ...
##  $ Bill  : num  NA NA NA 13.5 13.4 ...
##  $ Wing  : num  76 76 76 76 77 78 77 77 77 77 ...
##  $ Mass  : num  23.6 27.5 28.1 27.8 26.5 ...
##  $ Sex   : int  0 1 1 1 1 1 1 1 1 1 ...
##  $ Sex.1 : chr  "female" "male" "male" "male" ...

head(d)

##   BirdID Year Tarsus Bill Wing  Mass Sex  Sex.1
## 1      1 2002   16.9   NA   76 23.60   0 female
## 2      2 2001   16.8   NA   76 27.50   1   male
## 3      2 2002   17.2   NA   76 28.10   1   male
## 4      2 2003   17.5 13.5   76 27.75   1   male
## 5      2 2004   17.8 13.4   77 26.50   1   male
## 6      2 2004   17.7 13.1   78 26.00   1   male
```

```
summary(d)
```

```
##      BirdID           Year          Tarsus           Bill           Wing
##  Min.   :  1.0   Min.   :2000   Min.   :15.00   Min.   : 9.7   Min.   :60.
0
##  1st Qu.:117.0   1st Qu.:2003   1st Qu.:18.00   1st Qu.:13.0   1st Qu.:76.
0
##  Median :259.5   Median :2004   Median :18.60   Median :13.3   Median :77.
0
##  Mean   :290.3   Mean   :2004   Mean   :18.52   Mean   :13.3   Mean   :77.
4
##  3rd Qu.:477.0   3rd Qu.:2005   3rd Qu.:19.10   3rd Qu.:13.7   3rd Qu.:79.
0
##  Max.   :636.0   Max.   :2010   Max.   :21.10   Max.   :16.0   Max.   :84.
0
##                                 NA's   :85      NA's   :630    NA's   :75
##       Mass            Sex           Sex.1
##  Min.   :18.60   Min.   :0.000   Length:1770
##  1st Qu.:26.40   1st Qu.:0.000   Class :character
##  Median :27.60   Median :1.000   Mode  :character
##  Mean   :27.76   Mean   :0.513
##  3rd Qu.:29.10   3rd Qu.:1.000
##  Max.   :36.20   Max.   :1.000
##  NA's   :66
```

We can also look at the data structure. Data structure is incredibly important, but people often ignore it. The way data is structured will determine how you can analyse it, so you need to know what your data structure is. The first thing you do is find out what is the identifying feature of a single row entry. With identifying feature, I mean how is a single row differentiated from other rows. Let's have a look at the sparrow dataset:

```
head(d)
```

```
##   BirdID Year Tarsus Bill Wing  Mass Sex  Sex.1
## 1      1 2002   16.9   NA   76 23.60   0 female
## 2      2 2001   16.8   NA   76 27.50   1   male
## 3      2 2002   17.2   NA   76 28.10   1   male
## 4      2 2003   17.5 13.5   76 27.75   1   male
## 5      2 2004   17.8 13.4   77 26.50   1   male
## 6      2 2004   17.7 13.1   78 26.00   1   male
```

There is a BirdID number, but it is not unique. The same bird can be caught more than once, e.g. bird 2 has been caught in multiple years. It even has been caught twice in 2004. The measurements for Tarsus, Bill, Wing, Mass however differ slightly, even between the recaptures within years. From this we can conclude that our data structure is repeated measures of birds within years. Now I want to know, how many years, and how many bird capture per year?

```
table(d$Year)
```

```
##
## 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010
##  236   98   66  177  468  458  123  107   11   23    3
```

This gives us a nice overview of how many capture per year. Now we want to know how many captures per bird. The table() is a bit difficult to yield here:

```
table(d$BirdID)
```

We really want to summarize this. Now we can see how often each bird is captured, but we really want to go one level up - how many birds have been captured once, twice, three times ect. ? It's a bit like inception. We want to do a table of a table:

```
table(table(d$BirdID))
```

```
##
##   1   2   3   4   5   6   7   8   9  10  11  12
## 225 147  98  49  45  28  15  12   8   6   1   2
```

There are two birds that have been caught 12 times!

There are other ways of doing this - you can also use the dplyr version:

```
require(dplyr)
```

```
## Loading required package: dplyr
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
BirdIDCount <- d %>% count(BirdID,BirdID, sort=TRUE)
BirdIDCount %>% count(n)
```

```
## # A tibble: 12 x 2
##        n    nn
##    <int> <int>
##  1     1   225
##  2     2   147
##  3     3    98
##  4     4    49
##  5     5    45
##  6     6    28
##  7     7    15
##  8     8    12
##  9     9     8
```

```
## 10    10     6
## 11    11     1
## 12    12     2
```

When people talk about sample size, it is often not simply the total number of observations. While the total number is crucial and needs to be given, it is equally crucial to give sample sizes of sub-group levels - individuals per treatment group, observations per national park, per year, per family.

Exercises:

1. How many repeats are there per bird per year?

2. How many individuals did we capture per year for each sex? Compute the numbers, devise a useful table format, and fill it in.

3. Think about how you can communicate (1) and (2) best in tables, and how you can visualise (1) and (2) using plots. Produce several solutions, and discuss with GTA and your peers which the pros and cons for each solution to communicate and visualize the data structure for (1) and (2).

4. Write two results sections for (1) and (2), and ask your GTA for feedback. Each result section should use different means of communicating the results, visually and in a table.