**R intensive using the snail dynamics dataset – Day 2 Worksheet**

Instructor: Naima Starkloff

September 12, 2022

**Expectations of participants in this Workshop:**

1. Learn computational skills that allow you to process a dataset and answer questions.
2. Participate in pairs to complete worksheets.
3. Come up with a scientific question in pairs that could be pursued as a publication.
4. Feel welcome to raise your hand and ask a question at any point.
5. Raise your hand to answer questions asked by instructor.

**The science: Snails & aestivation**

As you all know, many of the water bodies that we scoop our snails from become dry in the dry season. During this time, the snails go below the surface of the soil. This process is called **aestivation**. And snails are able to survive and avoid desiccation because they undergo physiological changes such as lowered metabolism.

We know that:

1. Infected snails are prone to mortality due to the added physiological stress of infection.
2. Water bodies are dry for a large range of time (1-7 months).

Therefore, I ask: Do longer periods of water body desiccation lead to smaller snail populations and fewer schistosome infections?

**Today we learn about data management and graphical representation using new R packages.**

**Installing and using packages in R**

So far, we have only been using base functions in R, that is the functions that come with the original R program. We can add additional **packages** to the base program in order to perform more specialized tasks. You can either use code or buttons to install and use packages. Once you have enough practice with R, the code-based approach will be very comfortable. You only need to install a package once, but you will have to load the package every time you reopen R or every time you clean the environment.

Installing packages using button:

1. In the bottom right corner, press the “Packages” button.
2. In the packages box, type the name of the package you want to install and press “install”.
3. To load the package, find it in the list in the Packages window in the bottom right and click the checkbox next to it.

Installing packages using code:

You need one line of code to install and one line of code to load packages. Here is an example with the package named ‘dplyr’, which we will be using today:

*> install.packages(‘dyplr’)*

*> library(dyplr)*

**\*CAUTION: The installing function requires the package be in quotations marks, and the loading function library() does not.**

Diagram

Description automatically generated

**What is Tidyverse?**

Tidyverse is a collection of packages that allows you to clean, manage, manipulate, analyze, and visualize data. The coding structure is a little different from base R and could be a bit challenging at first. But it has much more flexibility and options. I use a combination of both base R and tidyverse for our snail dynamics data. We will be primarily utilizing four packages:

1. dplyr
2. tidyr
3. stringr
4. ggplot2

We can install all packages in tidyverse at once:

*> install.packages('tidyverse')*

*> library(tidyverse)*

Let’s start open the script called “RWorkshop\_Day2\_Script\_forParticipants.R”.

**EXERCISES**

**EXERCISE 1: Recap of subsetting + introduction to Piping**

We will start by opening the same dataset as yesterday. We have to set our working directory and read in the .csv file.

*> setwd()*

*> Year\_data=read.csv("YearSurvey\_Aug21Jul22.csv")*

We are going to focus on just Lambo la Mwabasabi to start. This time you will isolate just surveys from Lambo la Mwabasabi using the subset() function and saving it as its own datafame:

*> Mwaba\_data=subset()*

**Question 1: How many months are snails aestivating for in Lambo la Mwabasabi?**

How can we approach this? What information about the water body would tell you that snails are aestivating?

**Question 2: How many months is the water body dry for?**

*> subset(Mwaba\_data, Water\_level=="Dry")*

Always feel free to go back to yesterday’s script to help you construct the code.

So, we find it is dry for 0 months. Does this mean that snails are not aestivating in Lambo la Mwabasabi?

**Question 3: How many months is Lambo la Mwabasabi low and with no snails?**

*> subset(Mwaba\_data, Water\_level=="Low" & Bulinus\_Number\_Collected==0)*

We can also write code that accounts for both cases (dry or low with 0 snails).

*> subset(Mwaba\_data, Water\_level=="Dry"|Water\_level=="Low" & Bulinus\_Number\_Collected==0)*

We now have code to answer question 1: How many months are snails aestivating for in Lambo la Mwabasabi?

But what about all water bodies in Busega?

Let’s start by isolating just water body surveys in Busega:

*> busega\_data=subset(Year\_data, District=="Busega")*

Then we can isolate surveys where water bodies are in Busega and are dry:

*> subset(busega\_data, Water\_level=="Dry")*

But we also want the surveys where water bodies are low and have 0 snails. We are going to save this subset as its own object called “busega\_aestdata”:

*> busega\_aestdata=subset(busega\_data, Water\_level=="Dry"| Water\_level=="Low" & Bulinus\_Number\_Collected==0)*

**ADVANCED CODING:**

You could also get the same dataset by subsetting from Year\_data:

*> busega\_aestdata=subset(Year\_data, District=="Busega" & Water\_level=="Dry"|District=="Busega" & Water\_level=="Low" & Bulinus\_Number\_Collected==0)*

So, now we have a list of rows of water body surveys where snails are aestivating. But in order to find the number of aestivationmonths for each water body, we would need to count the number of surveys for each water body. This would take a while!

So, we are going to try a new tool using the new packages we installed. We are going to try the technique called piping.

**Piping**

This allows us to create a sequence of multiple functions. This will make your code shorter and allow you to use more intuitive functions.

For example, you can go through this sequence in one line of code:

1. subset your data to only surveys in Busega and those that are dry or low with 0 snails
2. Group surveys by water body
3. Count the number of surveys

We make use of the piping symbol “%>%” between each of these functions. Following a pipe, the next line of code can go to the next script line. This makes it easier for us to work with it.

*> Busega\_dry=subset(busega\_data, Water\_level=="Dry"| Water\_level=="Low" & Bulinus\_Number\_Collected==0)%>%*

*group\_by(Waterbody) %>%*

*count(Waterbody)*

We will want to rename the new column that contains the number of dry months as it was automatically saved as “n”. Here are three ways of renaming columns. The first two are with base R and the last using tidyverse.

> col*names(Busega\_dry)[colnames(Busega\_dry)=="n"] = "DryMonths"*

*> colnames(Busega\_dry)[2]="DryMonths"*

*> Busega\_dry=rename(Busega\_dry, DryMonths=n)*

We can look in the variation in the number of months that a water body is dry for in a year using a histogram:

*> hist(Busega\_dry$DryMonths)*

**EXERCISE 2: Introduction to joining**

As you all know, the data that you collect from the field gets entered into three sheets:

1. Snail measurement sheet
2. Snail shedding sheet
3. Water body sheet

To do our analyses, sometimes we want to combine these sheets, so that we end up with the correct measurements, shedding and water body data for each snail. This can be done quickly by using join functions.

QUESTION: Do the number of months a water body has aestivation affect the number of snails collected in a year? Are there more snails in waterbodies with fewer months of aestivation?

QUESTION: Do the number of months a water body has aestivation affect the number infected snails? Are there more infected snails in waterbodies with fewer months of aestivation?

In the last exercise you created a dataset with only water bodies in Busega and the number of aestivation months. I am also giving you a dataset with all the water bodies in Busega and the total number of snails and infected snails identified across the 12 survey months.

We will read in this dataset and take a look at it:

*> Bus\_Snails=read.csv("Busega\_snails.csv")*

*> Bus\_Snails*

Question: Why does the Bus\_Snails dataset have 21 rows and the Busega\_Aest have 16 rows?

We can combine two data sets by using the full\_join() function. The function will find all the unique water bodies across both data sets as rows and then add all columns from both datasets. If some column data is missing for a row, it will insert an “NA”:

> *all\_busega1=full\_join(Bus\_Snails, Busega\_Aest, by="Waterbody")*

Water bodies with NA in the AestMonths column did not have aestivation, that is they always had water. This is the same as 0 months of aestivation. So, we can replace the NA with 0:

*> all\_busega1$AestMonths[is.na(all\_busega1$AestMonths)] <- 0*

We can use the anti\_join() function to find out which water bodies are present in one data set and not the other. The first data set you enter is the reference data set, i.e., R will see what water bodies are present in the reference data set but not in the other data set. So, let’s start with Bus\_snails first as we know there are more water bodies in this dataset:

*> anti\_join(Bus\_Snails, Busega\_Aest, by="Waterbody")*

We can see that 5 water bodies were in Bus\_Snails, but not in Busega\_Aest. These 5 water bodies do not have aestivation. They always have water.

*> anti\_join(Busega\_Aest,Bus\_Snails, by="Waterbody")*

When we start with Busega\_Aest first, there is an error message. This is because all water bodies in this dataset are also in Bus\_Snails.

**ADVANCED CODING:**

You may only want to create a joined dataset that only has water bodies from one dataset, and to ignore any extra water bodies in the second dataset. For example, maybe you only want water bodies that have aestivation but also add the data on infection. If so, you can use the left\_join() function and enter the dataset of interest first:

*> all\_busega2=left\_join(Busega\_Aest, Bus\_Snails,by="Waterbody")*

Let’s see now if the number of months of aestivation affects the number of snails and number of infected snails. The plot() function creates a scatter plot and the abline(lm()) function (this is a complex double function) adds a line of best fit.

*> plot(all\_busega1$SnailNumber~all\_busega1$AestMonths)*

*> abline(lm(all\_busega1$SnailNumber~all\_busega1$AestMonths))*

*> plot(all\_busega1$InfectedSnails~all\_busega1$AestMonths)*

*> abline(lm(all\_busega1$InfectedSnails~all\_busega1$AestMonths))*

**EXERCISE 3: Summary statistics**

Summary statistics are used to quickly summarize and get information about the dataset. This is greatly assisted by using the piping tool you already learned. We will make use of two summary statistics functions today:

1. count()
2. summarise()

The count() function allows you to count the number of rows that belong to a certain category. We are going back to the Year\_data dataset. Remember, in the Year\_data data set, each row refers to one survey. So, we can count the number of rows in each district and that tells use the number of surveys done per district:

*> Year\_data%>%count(District)*

Or we could count the number of surveys per water body. This should be 12 to ensure that there was a full year of surveys for each water body:

*> Year\_data%>%count(Waterbody)*

Uh oh! There are two water bodies that do not have 12 surveys. One was 10 and another has 2. But they are the same water body, and one seems to have a typo in its name. We need to fix this. We will fix it when we get to EXERCISE 4, which is for data cleaning and management.

**ADVANCED CODING:**

You can also use the group\_by() function and extra pipe before the count() function. This would group water bodies by a category first and then count the number of rows. This can be used, for example, to count the number of water body surveys that are dry for each water body. You would group the data set by the Water\_level column:

*> WB\_waterlevel=Year\_data%>%group\_by(Water\_level)%>%count(Waterbody)*

It is best to save the output of complicated piped data as an object.

The summarise() function allows you to perform mathematics such as addition or finding the average based on certain criteria. This method makes use of the group\_by() function and piping. It is best to save the output of complicated piped data as an object.

We can find the total number of snails per district. We first group the data by district and then use summarise() for the sum:

*> sum1=Year\_data%>%group\_by(District)%>%*

*summarise(totalsnails=sum(Bulinus\_Number\_Collected))*

We can find the mean number of snails per survey for each district. We first group the data by district and then use summarise() for the mean:

*> mean1=Year\_data%>%group\_by(District)%>%*

*summarise(mean=mean(Bulinus\_Number\_Collected))*

You can also group by two columns. So, then we get the sum of snails per district and per phase:

*> sum2=Year\_data%>%group\_by(District, Phase)%>%*

*summarise(totalsnails=sum(Bulinus\_Number\_Collected))*

Or the average snail number per district and per phase:

*> mean2=Year\_data%>%group\_by(District, Phase)*

*%>%summarise(mean=mean(Bulinus\_Number\_Collected))*

**ADVANCED CODING:**

You can also pipe in a subset, if you want to find out a summary for just a certain group of surveys. For example, if you want to know the number of snails per phase of only water bodies in the district of Kwimba. Remember there are many ways to do things in R, so rather than subsetting, you can also use a function called filter() to only isolate surveys of interest.

*> Kwimba\_sum=subset(Year\_data,District="Kwimba")%>%*

*group\_by(Phase)%>%summarise(totalsnails=sum(Bulinus\_Number\_Collected))*

*> Kwimba\_sum=Year\_data%>%filter(District=="Kwimba")%>%*

*group\_by(Phase)%>%summarise(totalsnails=sum(Bulinus\_Number\_Collected))*

**EXERCISE 4: Data Cleaning & Management**

As we saw from the error with a water body being misspelled, often we must correct errors in the data. So, let’s go back and check on what was incorrect:

*> Year\_data%>%count(Waterbody)*

We can see that Lambo la Ng’wiza has two entries, and one has an additional space at the end. Also, there are a total of 110 water bodies. Is this the correct number?

This sort of error happens often because human error. That’s alright, there are easy ways to fix it. We can use the str\_replace() function to find a string of characters and replace it with a new string of characters:

*> Year\_data$Waterbody=str\_replace(Year\_data$Waterbody, "Lambo la Ng'wiza ", "Lambo la Ng'wiza")*

So, we replace the water body name with a space with one without a space. And it will do this in all cells where this is occurring. Remember there were two incorrectly named water bodies. You can now check if this is corrected:

*> Year\_data%>%count(Waterbody)*

There are now only 109 water bodies and no doubled name for Lambo la Ng’wiza.

Adding a space after a water body name is actually a very common error, so you can set up a line of code just in case other mistakes may happen. You can use str\_replace() to look for a space at the end of a water body name and replace it with no space:

*> Year\_data$Waterbody=str\_replace(Year\_data$Waterbody, " $", "")*

You can also create new columns of data from the columns that already exist. Three functions to do this are:

1. unite()
2. separate()
3. mutate()

The unite() function can be used to merge the information in two columns. For example, you may want the name of the water body to also include the district. You enter the new column name first in "" and the names of the existing columns that you want to merge. You also provide the symbol that will separate the two names (for example: sep= "\_") and whether you want to remove (remove=T) or keep (remove=F) the two original columns in the dataset. You then save the new object in place of the old Year\_Data dataset.

*> Year\_data= Year\_data %>%*

*unite("District\_Waterbody", District, Waterbody, sep="\_", remove=F)*

Open the Year\_data dataset and see where the new column is. The unite() function usually puts it close to the old columns.

The separate() function can be used to separate the information in one column into two columns. For example, you may want to separate the month and year in the phase column. You enter the old column name that you want to separate first and then names of the new columns in "". You will provide a symbol that tells us where to separate the data in the column (for example: sep= "\_") and whether you want to remove (remove=T) or keep (remove=F) the original column in the dataset.

*> Year\_data= Year\_data %>%*

*separate(Phase, c("Month", "Year"),sep="\_", remove=F)*

Open the Year\_data dataset and see where the new columns are. The separate() function usually puts them close to the old column.

The mutate() function can be used to create new columns with new information based on other columns in the dataset. For example, you can create a column that is the result of the mathematical outcome of another column or columns. And you can have different outcomes for different row types, for example, one mathematical function for surveys that are dry and a different mathematical function for those that are not dry. The coding here gets advanced so we will not cover it today, but please explore the advanced coding section below on your own.

**ADVANCED CODING:**

Yesterday, we calculated prevalence for surveys in Lambo la Mwabasabi and looked at how prevalence changed over time. We made a new column using this code:

*> Mwaba\_data$SchistoPrev=Mwaba\_data$InfectedSnails/Mwaba\_data$SnailNumber*

We make use of the mutate() function to do this in a similar way for the Year\_data data set:

*> Year\_data=Year\_data%>%*

*mutate(SchistoPrev=Infected\_Bulinus\_Number/Bulinus\_Number\_Collected)*

Open the Year\_data dataset and see where the new column is. The mutate() function usually puts them at the end of the dataset.

You have created a new column named “SchistoPrev” which has the outcome of the infected snail number divided by the total snail number for each row. You might recall that last time we calculated prevalence, “NaN” was put into the new rows when there were no snails scooped in that survey. We can also see “NaN” in the new column we just created in Year\_data.

The mutate function is useful because you can do different mathematical functions for different rows by incorporating the case\_when() function into the mutate() function. This way we can tell R that in cases where the total snail number is 0, that the prevalence should be 0. And cases where the snail number is not 0, then the calculation should happen as normal.

*> Year\_data=Year\_data%>%*

*mutate(SchistoPrev=case\_when(Bulinus\_Number\_Collected!=0~ Infected\_Bulinus\_Number/Bulinus\_Number\_Collected,*

*Bulinus\_Number\_Collected==0~0))*

Here is another example of using case\_when() within the mutate() function. We can create a new column where surveys with infected snails are marked as “Infected” and surveys without infected snails are marked “Uninfected”.

*> Year\_data=Year\_data%>%*

*mutate(InfectedSurvey=case\_when(Infected\_Bulinus\_Number>0~"Infected",*

*Infected\_Bulinus\_Number==0~"Uninfected"))*

Here is an example that uses the str\_detect() and case\_when() functions to detect the presence of a string of characters and if it is present carries out a certain outcome in a new column. Here, we are looking for the string “2021” in the Phase column and if present putting the number 2021 in the new column named “Year2”. Alternatively, we are looking for the string “2022” in the Phase column and if present putting the number 2022 in the new column named “Year2”.

*> Year\_data=Year\_data%>%*

*mutate(Year2 = case\_when(str\_detect(Phase, "2021") ~2021,*

*str\_detect(Phase, "2022") ~2022))*

**EXERCISE 5: graphing with ggplot2**

The last specific set of skills we are learning in this workshop is to visually represent our data, so it is easy to interpret. The ggplot2 package makes really nice graphs. We will return to our dataset this morning called “all\_busega1” that includes the number of months water bodies in Busega are undergoing aestivation and the number of snails in that water body through a year’s survey. Last time we graphed it using this format:

*> plot(all\_busega1$SnailNumber~all\_busega1$AestMonths)*

Where you first enter the dependent variable and then the independent variable.

In ggplot2 the coding structure is a bit different:

**ggplot format: ggplot(dataframename, aes(x= , y=,))+ graphtype()**

*> ggplot(all\_busega1, aes(x=AestMonths, y=SnailNumber))+ geom\_point()*

Similar to piping, you can hit enter to move the code to the next line if you use a “+” and R will include the next line of code when it runs it.

We can also look at the pattern of infected snails by number of aestivation months:

*> ggplot(all\_busega1, aes(x=AestMonths, y=InfectedSnails))+*

*geom\_point()*

The package ggplot2 is really nice to work with as you can just add more details to make a more complicated graph. It is useful to save the simplest version of the graph as an object that you can then add more to. We will just be working with the SnailNumber column for the rest of this section. Save the graph code as an object and open the object to see the graph:

*> graph= ggplot(all\_busega1, aes(x=AestMonths, y=SnailNumber))+ geom\_point()*

*> graph*

Thereafter, we can add aesthetic information such as a line of best fit using the geom\_smooth() function:

*> graph+geom\_smooth(method="lm")*

And we can edit the x and y axes of the graph using the xlim() and ylim() functions:

*> graph+geom\_smooth(method="lm") +*

*xlim(0,10)+ylim(0,1000)*

The changed axes are not very helpful so we will go back to the graph without them:

*> graph+geom\_smooth(method="lm")*

We can also change the names of the axes using the labs() function:

*> graph+geom\_smooth(method="lm")* *+*

*labs(x="Months of aestivation", y="Number of snails collected/year")*

We can also add a name for the graph within the labs() function:

*> graph+geom\_smooth(method="lm")* *+*

*labs(x="Months of aestivation", y="Number of snails collected/year",*

*title="Snail numbers and aestivation duration")*

Lastly, we can change the color and of the points shapes in the graph by adding extra information to the geom\_point() function:

*> ggplot(all\_busega1, aes(x=AestMonths, y=SnailNumber))+*

*geom\_point(col="red", size=2, shape=21)*

Let’s save this new graph as “graph2”

*> graph2 =ggplot(all\_busega1, aes(x=AestMonths, y=SnailNumber))+*

*geom\_point(col="red", size=2, shape=21)*

We can then add a line of best fit to graph2 suing the geom\_smooth() function and change the color and thickness of the line within the ():

*> graph2 +* *geom\_smooth(method="lm", col="black", size=0.5)*

**RESOURCES:**

join resources:

<https://stat545.com/join-cheatsheet.html>

Summary stats resources:

<https://davetang.org/muse/2016/10/13/using-dplyr-aggregate-r/>

<https://dplyr.tidyverse.org/reference/summarise.html>

Cleaning & management resources:

<https://raw.githubusercontent.com/rstudio/cheatsheets/main/data-transformation.pdf>

<https://raw.githubusercontent.com/rstudio/cheatsheets/main/tidyr.pdf>

<https://raw.githubusercontent.com/rstudio/cheatsheets/main/strings.pdf>

<https://raw.githubusercontent.com/rstudio/cheatsheets/main/lubridate.pdf>

ggplot2 resources:

<https://ggplot2.tidyverse.org/>

<http://r-statistics.co/Complete-Ggplot2-Tutorial-Part1-With-R-Code.html>

<https://r-graph-gallery.com/>

<http://www.stat.columbia.edu/~tzheng/files/Rcolor.pdf>

<http://www.sthda.com/english/wiki/ggplot2-point-shapes>

<http://www.sthda.com/english/wiki/be-awesome-in-ggplot2-a-practical-guide-to-be-highly-effective-r-software-and-data-visualization#:~:text=qplot()%3A%20Quick%20plot%20with%20ggplot2,-The%20qplot()&text=It%20can%20be%20used%20to,plots%2C%20histogram%20and%20density%20plots>.

<https://raw.githubusercontent.com/rstudio/cheatsheets/main/data-visualization.pdf>