Intro to R in 15 Steps

The goal of this tutorial is to introduce you to one of the most commonly used programming languages in computational biology: **R**. Throughout this tutorial, you can think of learning **R** as learning how to use a specialized smartphone, and learning the specific **R packages** as the various apps designed to help you do different tasks computational biologists do.

Step 1: Create an account with RStudio Cloud

R & R Studio are free! For this tutorial, I recommend you use R Studio Cloud. To create a free account just access this website *https://login.rstudio.cloud* and create a new account using your preferred email address.

Step 1b: Download and install R + R studio alternative R is free and will always be! If after this workshop, you would like to go one step further, you can download R by accessing this website: *http://www.r-project.org/* The download link will take you to a list of CRAN mirrors – pick a mirror: e.g. South Africa (for a list of the mirrors *https://cran.r-project.org/mirrors.html*).

Download R studio - free option; *http://www.rstudio.com/ide/download/*.

R studio is composed by **4 panels**: **1**=Top Left, **2**=Top Right, **3**=Bottom Left, and **4**=Bottom right. Note taking happens in **Panel 1**. **Panel 2** has "your environment", every time you open R, you read data or create any kind of object in R it will be listed in your ***Environment*** tab. Any code you run in **Panel 3** will appear also in Panel 2, under ***History*** and anything that you visualize and more will be in **Panel 4**. We will delve more into the details soon.

You will then need to set your working directory in your local computer if you chose the option to download R.

A working directory is the folder in your computer where all available data are & results will be saved.To find out the current working directory, type getwd() and hit ENTER.To set a new working directory, you will specify the computer path to a folder, for example *setwd("~/Dropbox/USB")*. You can also use the non-programming option by following the drop-down menu on R Studio: click on the tab called *Session -> Set Working Directory -> Choose Directory* and just select the folder of your choice in your computer.

Step 2: Learn the Help Command

Practice makes perfect A way to learn is by deciphering error messages! So, if you get an error message, which is the output showing on red font, don't panic! Computational biologists are constantly figuring out what error messages mean and why do we get them. An effective way to do this is to **google** your error message. Often someone has already found a solution and there is a ton of help online! Now since we are learning how to code, we can ask R for help using a specific function in a package by calling in R's built-in help function, or by directly typing a question mark before the function you are trying to use.

```
#You want to learn what the function "help()" does in R
?help()
help()
```

Question 1: How would you access built-in help about the R package *dplyr*, this is a commonly used package in data wrangling, management or rearranging (*https://dplyr.tidyverse.org/*).
A/

Step 3: Do basic calculations

R is like a smartphone, and its built-in package called **base** can do everything calculators do & much more! Let's explore using R as a calculator.

Let's use R to do some simple calculations:

```
#Sum
3 + 3

#Exponential
2^-7

#log function
?log() #add a question mark before a function to learn about what it does.
log(2)
```

Question 2: What type of logarithm (e.g. natural log, common log-base 10, or binary logarithm) does the log function computes?
A/

But then why using R and not a calculator? In R, anyone can create so called **functions** to create plots, maps, figures, organize data bases, conduct statistical analyses, make presentations, build websites, analyze whole genome sequences, among others.

Step 4: Familiarize yourself with packages and versions

Functions are compiled into **packages** (remember to think of these as the apps in a smartphone) published into the Comprehensive R Archive Network (CRAN), R's central software repository hosted by the R foundation. In March 2022, there were 18,948 R packages made available through CRAN. (*https://cran.r-project.org/web/packages/*)

When using R or R studio, you can find all the built-in packages under **Packages** in Panel 4. To search for a specific package you can select "Install", this opens a new window in which you may select the CRAN repository (recommended option) and search the name of the package by typing the name of it under **Packages (separate multiple with space or comma)**. Make sure that under "Install to Library" -> you use the default location to install R packages and double check that the box for "install dependencies" is ticked. We can also install packages by coding in the console (**Panel 3**).

For instance search the package called **ggplot2** used to create elegant graphs/plots using your data. The amount of freely available tutorials varies a lot among packages, some plotting packages like ggplot have lots of tutorials online whereas more specific packages will have sparse info. Yet, almost all packages have a user guide available online or by simply clicking on the name of the package in Panel 4, under the **Packages** tab, or the **help()** function we examined earlier.

It will become tricky to manage packages if these require different R versions (just like some apps in your smartphones that stopped being supported in old smartphones). It is important to keep the packages you often need organized and only load those that you will use, to avoid any problems. Also, since the community is actively developing **free** packages, there ways to install these are getting easier and easier.

Step 5: Familiarize yourself with objects in R

R programming language recognizes a handful of fundamental *units* used to perform different actions, we call those *objects*. The main types of objects in r are:

Functions: In R, a function is an object that we will *code*. Think of this as a task you want the computer to do for you. We will use functions by specifying instructions using *arguments* (inside of the parentheses) that may be necessary for the function to accomplish the actions. The basic syntax of a function can look as simple as this: *help()* or *log(x,base=y)*.

Vectors: Vectors are variables, or series of values of the same class, they can be continuous ("numeric"), categorical ("factor")", etc.

Data Frames and Tibbles: A data frame is a matrix composed by variables, but formatting variables as a data frame allows handling versatility in your R studio environment. Many functions require formatting your data as data frame. We often need to try out different ways to re-organize our data, so to do this we will take advantage of a new, simpler and versatile version of the *data frame* object called the *tibble* to be used along with the package *dplyr*.

List: A list can comprise all sorts of object types. Few functions require data as lists.

Other: Other less frequently used objects include: scalars (a single value), matrices (similar to data frame, but allow objects with varying lengths).

Let's explore these in R:

```
#Object
x <- 3 + 5
x

#Vector
y <- c(1, 2, 3, 4, 5)
y

seq(1,10, by = 1)#Creates a sequence following specific rules

vector1<-seq(1, 10,  by = 1)
vector2<-seq(1, 20, by = 2)
vector3<-seq(1, 30,  by = 3)
vector4<-c("A", "B", "C", "D", "E", "F","G", "H", "I", "J")

#Data frame
df1<-data.frame(vector1,vector2,vector3, vector4)

#Tibble
tibble1<-tibble::tibble(df1)

#List
list1<- list(df1, vector1, vector2, vector3, vector4, x, y)

#Note that giving a name (e.g, "df1") will have downstream implications.
#It is important to give your objects unique, intuitive, short & simple names.
```

Question 3: What is an advantage and a disadvantage of using a list?
A/

Step 6: Familiarize yourself with functions in R

Here try to find a few more of the many built-in functions in r, for example:

```
#Calculate mean
?mean
mean(vector1)
#Sum values in a vector using
?sum
sum(vector2)

#If we want to build a data frame we *need* to have a set of vectors all of the same length
```

```
#Figure out length of a vector:
length(vector3)
#Retrieve unique values in vector
unique(vector4)
#Sort values in a vector
df1$vector5<-sort(vector4, decreasing = TRUE)
#Sample 6 values at random from a numeric vector
sample(vector3, 3)
#Sample 4 letters at random from a character vector
sample(vector4, 4)
```

Question 4: How would you modify the *sample()* function to sample only three values at random, allowing a given number or character to appear more than once?
A/

You can also use R to plot functions just as most scientific calculators do. But before coding plots, we will learn a few more R commands to extract information from data frames or similar objects.

Step 7: Practice using data frames

Very often we want to summarize, visualize, simulate or analyze existing data, for that, one can start by using the data frame format in R. Here we are going to explore functions using very basic data frames, to learn more about how the data look like.

Note that in R studio, going to **Panel 2** and simply clicking on a given data frame, will retrieve all its contents in a new tab in **Panel 1**.

Question 5: How would you modify one of this functions (e.g. *mean()*) to deal with missing values?
A/

Step 8: Practice exporting data frame

If you want to save/export the data frame you created in R, you just need to call a function named *write.csv*. To export your data frame, just name the file. I recommend you save your traditional data files (aka. *excel-like* tables) to use in R as a ***.csv*** (**c**omma-**s**eparated **v**alues) file format.

```
#Exporting data
write.csv(df1, "df_2022jul2.csv")
```

Question 6: Where is the file saved? Use the getwd() function to find out. How can you export text or a list that was created using R?
A/

Step 9: Practice importing data in R

Very often we have data that we want to import into R, there are several formats that are compatible with R. Data tables can be generated in R or other software, for example: excel, statistical software or google sheets (free). When data is not generated in R, data, can still easily be imported but there are distinct ways to import files depending the format of the data to be imported. Common formats include data as: text or *.txt*, comma separated values or ***.csv*** and/or as excel *.xlsx* (only the latter requires installing a package called *.openxlsx*. I recommend that you always save or import the data as ***.csv***.

We previouly saved our data as a *.csv*, to import these data, we can just use the **read.csv()** function ***-as long as the file is contained in our current folder used as working directory.***

```
#Import data as csv
imported_df1<-read.csv("df_2022jul2.csv", header=T)
```

Step 10: Create plots

We can use base R to create quick plots and identify patterns. We then can export these as a ***.pdf*** file format. Plotting is one of the biggest strengths of R. The possibilities are unlimited and there is a universe of tutorials out there to show you how to customize plots. Over the years I have moved on from base R into using the package *ggplot2* which we will explore next.

```
#First: Plotting with base R
#Plot 1 continuous variable
hist(sample(seq(1:100),200, replace=TRUE), breaks=25, main="") #Note Question 7

#Plot 2 continuous variables
plot(df1$Cont1~df1$Cont2)

#Plot 1 continuous variable and 1 Binary/Categorical
boxplot(df1$Cont1~df1$Binary1)

#To export a pdf file with multiple figures:
pdf("plotsA_2022jul2.pdf")
hist(sample(seq(1:100),200, replace=TRUE), breaks=25, main="")
plot(df1$Cont1~df1$Cont2)
boxplot(df1$Cont1~df1$Binary1)
dev.off()
```

Question 7: What happens when we remove *main=" "*. How would you go about to rename your y axis label?
A/

Next we will use data on the survival status of individual passengers on board of the ***Titanic*** ship. Visit my google drive to download the titanic data set **.csv** format: *https://drive.google.com/file/d/1jhGvXkYSrtsgmEnZ2GBR6AzBy1BUGbNg/view?usp=sharing*

After successfully downloading *trains.csv* file you would be able to use the ***Upload*** option in **Panel 4**. Simply choose the file to upload, and hit ***Ok***. Once you do this, it will be available in your environment.

Question 8: Did you get an error trying to upload this data file? How did you go about to fix this error?
A/

```
#Import the data set
titanic <- read.csv('trains.csv',stringsAsFactors = F, header = T)

#Explore the data set using the functions we learned before
head(titanic) #to see first 10 rows and column names
summary(titanic) #to get a summary of each column
#***Note that R describes missing data as "NA"

#Plot to summarize passenger age with base R
hist(na.omit(titanic$age),col=rainbow(length(1:8)),main="",xlab="age", ylab="count")
```

Step 11: Practice using ggplot2

Base R plots can be modified to look as pretty/detailed as you would like them to look. However, packages like **ggplot2** make all the heavy-lifting for you. Next we will learn how to make a few simple, yet publication-level, *nice looking plots* using **ggplot2** and the data management package **dplyr**. This tutorial does not cover data wrangling using dplyr, but there are many out there that do!

We just created a simple histogram using base R to use as comparison with the same plot created using *ggplot2* and associated packages.

```
#Now let's use dplyr and ggplot2
#Part 1
#Note that the code gets increasingly complex, as you try to make
#better-looking, customized figures in R
titanic %>% head(10) #Overview of the first 10 rows.
titanic %>% describe_tbl() #Describes data.

#Plot to summarize passenger age with ggplot
p1<-titanic %>% drop_na() %>% ggplot(aes(x=age))+
  geom_histogram(fill=rainbow(length(1:8)),bins=8,show.legend = FALSE)+
  labs(x = "age")
p1
```

The histogram created using this simple ggplot Rcode is more attractive, but we can do a lot more to customize this plot. For instance you can try adding different themes (e.g., classic, bw, minimal). Simply type ***theme*** followed by tab on the console to learn about the different themes and how to add them to an existing ggplot.

Step 12: Create publication-ready plots using ggplot2 and dplyr

```
#Part 2
#We can combine multiple categorical and continuous variables
p2<-ggplot(data=na.omit(titanic),
           aes(x=factor(sex),y=age,fill=factor(survived)))+
  geom_boxplot()+
  labs(title="Survival of passengers by sex, age and class",
       subtitle="Not everyone had equal chances of surviving",
       caption="Plot by J.Golcher") +
  xlab("Passenger class")+
  ylab("age")+
  facet_wrap(~factor(pclass))+
  scale_fill_manual(values=c("mediumvioletred","cornflowerblue"),
                    name="",breaks=c("0", "1"),labels=c("died", "survived"))+
  theme(plot.title = element_text(face = "bold"))

p2
```

Using ggplot, we can quickly make clear figures that help data interpretation. Multiple data visualization R tutorials and cheat sheets exist in the internet. If you are interested in how to go from beginner to pro, again ***practice makes perfect***. There are also new packages that make it easier to explore patterns from data, one of them is the package ***explore***. This part of the tutorial is adapted from a 2022 tutorial written by Roland Crasser that can be accessed here: *https://cran.r-project.org/web/packages/explore/vignettes/explore_titanic.html*

```
#Part 3
#Let's explore functions and plots in explore
#
#First let's make age a categorical variable:
titanic$age_cat <- cut(titanic$age,breaks=c(0,19,60,90),
                       labels=c("children","adult","senior"))

#This is perhaps unnecessary but just to be sure
titanic2 <- tibble(titanic) #data frame into tibble
```

```
#We would like to reorganize our data as counts per categories
titan_sum<-titanic2 %>% group_by(pclass, survived,age_cat,sex) %>% dplyr::count()

titan_sum %>% describe_tbl(n=n) #describes data and NA

#Creating plots using explore
p3<-titan_sum %>% explore(age_cat, n = n) #passenger class distrib.
p4<-titan_sum %>% explore(age_cat, target = survived, n = n)
p5<-titan_sum %>% explore_all(n = n) #To see counts for all vars.
p6<-titan_sum %>% explore_all(target = survived, n = n, split = TRUE)

p3
p4
p5
p6
```

Step 13: Export ggplot2 charts

In this example we went beyond simply plotting using base R to practice data re-organization with advanced visualization. Lastly we will learn how to arrange and export ggplot figures using the package *patchwork*.

```
#Part 4
#A package to save ggplots
library(patchwork)
p<-(p1)/(p2)
#Save plots
ggsave("plotsB_2022jul2.pdf", p)
```

Question 9: How would you modify this code to save the plots next to one another using *patchwork* (instead of one on top ot the other).
A/

Now we have explored how to make a few plots using base R and some of the newest plotting options available associated to the package *ggplot2*. Still, remember that this is the very tiny tip of the iceberg. It will be up to you to take it from here and make significant progress in your R journey coding and making plots. There are almost too many resources online to learn how to visualize data using R, but as a start visit: *https://www.r-graph-gallery.com/* to get inspired with a gallery of different types of charts.

Step 14: Run a simple statistical test.

As a biologist, one of the first statistical tests we learn is how to calculate a Chi-squared value. For instance, such tests are used in the context of transmission genetics, together with the Chi-squared value distribution table to test for significance rejecting a null hypothesis of observations deviating or not from expected. As you already guessed it, R can do this for you and much more. We will start by learning how to do this test using the console, and review how to interpret results of the *chisq.test()* function in base R using the Titanic toy data set.

Question 10: What would be **1)**the null and **2)**the alternative hypotheses for a Chi-squared test looking at passenger survival by class? After running the code below: **3)**Was the p-value significant? **4)** What does this mean?
A/

```
#Chi-squared test using R
res1<-chisq.test(titanic$pclass, titanic$survived)
res1
res1$observed
```

```
res1$expected
```

Step 15: Cite R and Rpackages

It is key to give credit to the generous people who develop packages for free, often with public funds from universities and other research institutions. Providing authors with the credit they deserve also encourages funding agencies to support their work, which ultimately benefits the entire community of R users.

```
citation() #To cite R
R.Version() #Figure out R version
RStudio.Version() #To cite R studio
#To cite all the packages  used in this tutorial:
citation("ggplot2")
citation("dplyr")
citation("tibble")
citation("explore")
citation("patchwork")
```

This mini-workshop was modified from the 2018 Workshop notes Intro to R for fisheries researchers in Tanzania co-taught by Dr. Jessica Rick and Dr. Jimena Golcher-Benavides. I would like to thank the generous R & R learning community for all the high-quality learning resources freely available online.

For more learning resources on how to use R & R Studio, check out these guidelines written by Thomas Mock: *https://www.rstudio.com/resources/webinars/a-gentle-introduction-to-tidy-statistics-in-r/*.