Basic Computing — Introduction to R*

Peter Carbonetto *University of Chicago*

The aim of this tutorial is to introduce R, and to use R to analyze data interactively. We will focus on one important data structure, the *data frame.* We will learn how to import tabular data into a data frame, and we will learn how to inspect, manipulate and analyze the data frame. In addition to learning how to program in R, we will also discuss what are the key aspects of a good data analysis, and how to analyze data in the world. This workshop is intended for biologists with little to no background in programming. **This tutorial is also a Google doc.**

How this tutorial is organized

This tutorial is divided into two parts:

1. First, we will analyze a small data set in R. Our focus will be learning about data frames: what they are, how to use them, and why they are important.
2. Second, we will apply the skills we developed in the first part to a much larger data set. You will (hopefully) find that analyzing a very large data set in R is not much different than analyzing a small data set (and, indeed, both are data frames, just one data frame is much larger than the other).

What is this document, and how should I use it?

This is a text document (a "Google Doc") containing text and R code. The R code is inside the gray boxes.

You can run the code by copying & pasting it into your favorite R IDE ("integrated development environment"); e.g., R, RStudio, Posit Cloud, Google Colab. For some practice, try running these two lines of code:

```
x <- rnorm(200)
hist(x,n = 32)
```

You may notice that your histogram is not the same as mine. *Why is that?*

Make this document your own by making a copy of it in your Google Drive, then add your own notes and code. I've also given you all permission to add comments to this Google doc. Feel free to use comments to post questions or suggest improvements. I'll periodically look for your comments.

Note: If you are running your code in a Jupyter notebook or in Google Colab, I recommend running this line of code so that the outputs look the same as they do in RStudio:

*This document is included as part of the Basic Computing—Introduction to R tutorial packet for the BSD qBio Bootcamp, University of Chicago, 2024. **Current version**: August 14, 2024; **Corresponding author**: pcarbo@uchicago.edu. Thanks to Stefano Allesina, John Novembre, Stephanie Palmer and Matthew Stephens for their guidance.

```
options(jupyter.rich_display = FALSE)
```

My data analysis

Here is my data analysis. It is an analysis of data from a 2008 Genetics article on the genetics of dog breeds:

```
dogs <- read.csv("dogs.csv",stringsAsFactors = FALSE)
fit <- lm(aod ~ weight,dogs)
print(fit)
```

You will notice that the code is very short! There's nothing wrong with that—successful analyses in R do not need to be long or complicated! Still, we will spend quite some time understanding this code and what it does, and in the process we will learn about R. But before we try to understand what the code is doing, let's start by trying to run the code in RStudio to reproduce the result.

Use this space to add your own notes and code.

A mini-course on data frames

A data frame is R's main data structure for *storing tabular data.* The data frame is one of the most important data structures in R, and is important enough that we will spend much of this tutorial seeking to understand how to work with and analyze data in data frames. (Not all data of course is tabular data, but because so many things in R work well with data frames, *it can be helpful to find ways to rework your data so that it fits into a data frame.*)

Before we get to more interesting things, we need to first get comfortable with some basic syntax for data frames. Let's run these lines of code and add a comment to the right of each line describing what this code did. To start you off, I've added comments next to the first three lines of code.

```
print(dogs) # Print the contents of the entire data frame.
head(dogs) # Print the first few rows of the data frame.
tail(dogs) # Print the last few rows of the data frame.
```

```
summary(dogs)
class(dogs)
nrow(dogs)
ncol(dogs)
names(dogs)
dogs$breed
dogs$aod
dogs$height
dogs$weight
dogs[,"aod"]
dogs[1,]
dogs[,"weight"]
dogs[1,"weight"]
x <- dogs[1,"weight"]
x <- dogs[,"weight"]
x <- dogs[1,]
```

You may have noticed that many of these lines of code *do the exact same thing*. This is a common theme in R (and in programming more generally): *there are often many different ways of accomplishing the same thing, and there is rarely one way that is "best"* (although sometimes R programmers get into passionate debates about this).

A reminder at this point that the advantages of analyzing data in R may be less obvious when working with a small a data set. Later when we work with a large data set the benefits of R will become more clear.

Checking for mistakes

All researchers, no matter how much experience they have, make mistakes. And most mistakes are stupid mistakes! (I have made my fair share of embarrassing mistakes.) Occasionally, very good researchers make mistakes in their papers. (See also here for a discussion of this mistake years later.) Therefore, *try to catch your mistakes early.*

So how can you find your mistakes? The good news is that R catches many of your mistakes: most methods in R have their own internal error-checking. Discussing your results with your labmates and advisors is also another way to catch mistakes. But ultimately you will need to develop some skills and strategies for performing your own checks.

A relatively simple but nonetheless helpful check is a "sanity check": it doesn't tell you the result is right, but at least it reassures you that the result is not horribly wrong. A good sanity check is simple—one that you can do on "the back of an envelope". Here we will perform a simple sanity check for our analysis of the dogs data (and in the process we will learn more about R).

Here our "sanity check" will be to hand-check that our model, $y = ax + b$, where x = weight and y = aod, is making sensible predictions of y given x. This will involve some basic arithmetic so in the process we will learn about how to do arithmetic in R. Also, plots are another powerful way to perform checks, and we will write some simple code to visualize our results.

Your code for checking your results.

Activity: Re-run this analysis on a different data set

Let's now try something *a bit audacious.* Let's try to re-run the same analysis as before, but on a different data set from the American Kennel Club. Will this new analysis produce a similar result, or not?

Using the skills you have developed so far—and a bit of creativity—I believe you can adapt the code you ran on the "dogs" CSV file to analyze the AKC data (the file is akc_data.csv). This exercise will involve making some judgments about how to analyze the data and therefore I do not expect everyone to get the same result. It will also involve some initial explorations of the data to understand what this data set contains, and how it differs from the first data set. *Do not be afraid to make mistakes!*

Note that you may encounter a challenge that you have not yet dealt with. When you encounter this challenge, try to figure out what is the issue, and we will discuss as a group how to overcome it.

Your code for analyzing the AKC data.

Tending to your gaRden

(Almost) every line of code in R acts on your R environment: it takes objects that exist in your environment, then generate new objects or overwrites existing objects. Therefore, to understand any line of code, you need to understand not only the code itself, but also what is the state of your environment (your "garden") the moment before you run your code; that is, the objects that are in your environment and what they represent. However, this is easier said than done when your environment (garden) is messy (not tended to). *Therefore an important skill as a coder is to tend to your gaRden.*

Your code for tending to your gaRden.

What are all these objects? Do we need to keep all of these objects? Could we have named some of these objects better to remind us what they are?

S. cerevisiae

Programming Challenge: Facts about dog breeds

The data frame is R's way of storing tabular data. It is one of the most important and more powerful data structures in R. In this Programming Challenge, we will take some time to understand data frames and how to work with them. Although our focus here is on data frames, many of the ideas you will pick up in this Programming Challenge are quite general.

Let's start this part of the tutorial with a clean environment. Then go ahead and import the dogs data set and print out the first few rows of the table:

```
rm(list = ls())
dogs <- read.csv("dogs.csv",stringsAsFactors = FALSE)
head(dogs)
```

The data frame is a data structure for storing tabular data. The data are actually stored in a very specific way: the data frame is a set of columns, and each column is a vector of the same length. Let's run some code to convince ourselves of this fact.

First, for convenience, make a copy of the first column, and call it "x":

```
x <- dogs$breed
```

Important note: This makes a copy of the original data, so if you were to modify or delete x, this leaves "dogs" unchanged.

This is a "character" data type. It is R's way of storing text data:

```
class(x)
length(x)
x
```

Note we could have also copied the first column this way:

```
x <- dogs[,"breed"]
```

And here's another way!

```
x <- dogs[,1]
```

Which way do you prefer?

By storing tabular data in this way, we can *divide and conquer:* since each column is also an object in its own right, if the data are too complicated to understand all at once, we can make a copy of the columns we want to look at more closely, and run code on the copy. *This is a useful strategy for dealing with complex data sets.*

To drive home this idea of a data frame as a collection of vectors, the way to create a data frame is in fact to join together a bunch of vectors of the same length. For example:

```
mydogs <- data.frame(
  breed = dogs$breed,
  lbs = dogs$weight,
  years = dogs$aod)
head(mydogs)
```

Each data structure in R has its own features and its own techniques for working with them. In time, you will learn to work with other types of data structures: some are used widely (e.g., an "lm" object), and some are very specialized (e.g., a GRanges object).

Now that we have some basic understanding of what is a data frame, let's jump into the Programming Challenge: the goal is to write code to answer some basic questions about dog breeds from the data. The challenges will get progressively more difficult, and will build on each other, so try not to rush through them. Sometimes the code will be given to you, other times you will be given hints for writing the code to answer the questions. Now, this data set is small enough that you can answer many of these questions by eye, but please don't do that. (That being said, you are welcome to look at the data to verify your answers.)

Before diving deeply into the Programming Challenge questions, first discuss a collaboration strategy with your teammates. How will you work on the problems together? How will you share and discuss solutions? (Maybe do this in a shared Google doc?) How will you make sure that everyone is included in the problem solving? How will you address conflicts, e.g., when your team comes up with more than one solution?

Warmup: Smallest and largest dog breeds

This should find the average height (in inches) of the largest dog breed:

```
x <- dogs$height
max(x)
```

Now write code to find the (average) height of the smallest dog breed:

Add your code here.

This didn't tell us which breeds were the smallest and largest. For example, to find the largest breed, we can do this:

```
y <- dogs$breed
i <- which.max(x)
y[i]
```

The output of which.max() was stored in object "i". How kind of object is "i"?

What is the smallest breed?

Add your code here.

What objects did you create to answer this question?

Another warmup: inspecting data about specific dog breeds

Suppose you wanted to look more closely at the height, weight and other statistics of the Alaskan Malamute, which is stored in the fifth row of the data frame. This is easily done by selecting the fifth row with the square brackets:

```
dogs[5,]
```

You can also select several rows at once, e.g.,

```
dogs[c(43,46),]
```

Practice a few times selecting different combinations of rows. What happens if you select a row number that is larger than the height of the table?

Facts about dogs' BMI

The body-mass index (BMI) is a standard quantity—and sometimes misused quantity!—in science and medicine. In R, the BMI is easily calculated:

```
w <- dogs$weight
h <- dogs$height
bmi <- 703*w/h^2
```

Based on this code, what is the mathematical formula for BMI?

Next, write some code to find the largest, smallest, mean and median BMI. What are the dog breeds with the largest and smallest BMI?

Add your code here.

If you would like to use the BMI data later on, you can insert these data in the data frame:

```
dogs$bmi <- bmi
```

(What is the benefit of adding these data to the data frame as opposed to storing the BMI data in a separate object?)

The longest-living dog breeds

In the dogs data frame we encountered two types of data: numeric data and text data. Another type of data that is very important is *logical data.* Although the data frame does not contain logical data, we can easily create logical data using *logical operators.*

You might create logical data in the process of answering questions about the data. For example, suppose you would like to know how many dogs have an expected longevity of 16 years or greater. Here is some code to answer this question:

```
x <- dogs$aod >= 16
summary(x)
```

What is "x" here?

```
class(x)
x
```

To get the indices that are "TRUE", use which():

```
i <- which(x)
length(i)
i
dogs[i,]
```

What does "i" contain?

Other logical operators include equals (==), and (&), or (|) and not (!). Running help(Logic) will give you a longer list.

Now write similar code to find the breeds with the following characteristics:

1. Expected age of death (AOD) at least 15 and average weight greater than 20 lbs.
2. AOD greater than 15 and "shortcoat" value of 1. (Later we will learn what the "shortcoat" column represents.)

Add your code here.

A QTL for weight (also, dealing with missing data)

In the *Genetics* paper, the strongest QTL ("quantitative trait locus") for weight was a QTL on chromosome 7. The "cfa7_46696633bp" column stores the breeds' allele frequencies for these QTL. What happens when you try to calculate the correlation between the allele frequencies and weights?

```
x <- dogs$cfa7_46696633bp
y <- dogs$weight
cor(x,y)
```

In turns out that the allele frequencies were not available for some of the breeds, so a special value "NA" was entered for those breeds. *Why "NA"?*

Use the is.na() function, and other functions you have used before, to find the missing entries, and determine:

1. How many allele frequencies are missing?
2. Which breeds are missing allele frequencies?

Add your code here.

The designers of R, appreciating that missing data is widespread in statistics, made sure that missing values were an integral part of the R programming language. Most statistical functions in R can deal with missing data. Read the documentation for "mean" and "cor", then use the guidance provided in the documentation to compute the average allele frequency at the QTL (that is, averaged across all dog breeds), and the correlation between body weight and allele frequency.

Add your code here.

What is a "factor"?

So far, we have seen three basic data types: character, numeric and logical. There is a fourth important atomic data type in R: *factor.* What is unusual about factors is that—last I checked—there is no equivalent in other popular programming languages, at least not as a primitive data type. And yet you will find that they are extremely useful.

None of the columns in the dogs data frame are a factor. But, like logical data, *we can create a factor from other data.*

The "shortcoat" column contains numeric data. As it turns out, *it may be more useful to analyze the data as a factor.*

In this part of the Programming Challenge, you will be given the code, and your task will be to run the code and interpret the outputs, with the aim of gaining some intuition for factors, how to use them, and when they may be useful.

Let's first run a few lines of code to inspect the shortcoat data:

```
x <- dogs$shortcoat
class(x)
x
summary(x)
unique(x)
```

Now run the following lines of code to create a factor:

```
x <- factor(x)
class(x)
x
summary(x)
```

Notice that although the data have stayed the same, the two summaries are different. This is because although the data haven't changed, the *data representation* or *encoding* has: *R treats the numeric encoding differently from the factor encoding*.

What does the second summary tell us? Judging by these outputs, what do you think a factor is?

These observations suggest that we can improve the data representation further; the data should be as easy to interpret as possible ("human readable"). Representing the data as zeros and ones may be convenient for the computer, but it is confusing when representing the data as a factor because it "looks" like numeric data, when in fact it is not.

Fortunately, now that the data are stored as a factor, this is easily fixed. To fix this, we modify a property ("attribute") of the object. Since this is the first time we are using an object's attributes, let's first take stock of the object's attributes:

```
attributes(x)
```

What does the "levels" attribute keep track of?

We can modify the levels attribute. For example, this replaces the zeros with "no" and ones with "yes":

```
levels(x) <- c("no","yes")
summary(x)
```

Having made these improvements to the shortcoat data, let's store the improved data in the data frame:

```
dogs$shortcoat <- x
summary(dogs)
```

To illustrate the power of factors, let's see how easily it can be incorporated into a linear regression analysis.

In one of my analyses, I found that dogs with short coats tended not to live quite as long as dogs with longer coats:

```
sx <- dogs$shortcoat
y <- dogs$aod
i <- which(x == "no")
j <- which(x == "yes")
mean(y[i])
mean(y[j])
```

Is this difference significant? We can check this with lm():

```
fit <- lm(aod ~ shortcoat,dogs)
coef(fit)
summary(fit)
```

What does the "shortcoatyes" output from coef(fit) represent? Is the difference significant?

The difference in expected lifespan might be explained better by differences in the body weights between dogs with short and long coats. Is the AOD difference explained by "shortcoat" still significant when weight is included as an explanatory variable for AOD?

```
fit <- lm(aod ~ weight + shortcoat,dogs)
summary(fit)
```

An index of dog breeds

There are many situations in biology research in which your data are text data (consider that DNA sequences are a type of text data). The stringi and stringr packages are popular packages for performing more complex analyses of text data. (In computer science, a "string" means a sequence of characters, so one can think of text data as a collection of strings.) In this last question, we will practice some simple manipulation of text data ("strings") to organize the dog breeds by the first letter of the breed name.

The first step is to extract the data we want. This can be done using substr():

```
x <- dogs$breed
d <- substr(x,start = 1,stop = 1)
d
```

Which is the most common first letter for a dog breed? And how many breeds start with this letter? To answer these questions, try creating a factor.

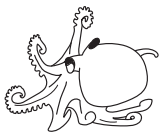
Add your code here.

The factor() function automatically determined which letters appeared in the data. But it could be useful to include all the letters in the alphabet, not just the ones that appear in the data. Modify your factor call above to include the unused letters as well. See help(factor) and help(LETTERS) for guidance.

Add your code here.

Once you have the new factor using all the letters, write code to determine which letters are not used for the first letter of any dog breed:

Add your code here.



O. bimaculoides

A "random" exercise

Here is a short piece of code that demonstrates the use of (1) a "for loop", and (b) if-else statements. This code simulates a simple "random walk" in which you decide to go up and down at the flip of a coin. The plot at the end shows the random walk over time.

```
n <- 100
sim <- data.frame(t = seq(1,n), x = rep(0,n))
x <- 0
for (i in 1:n) {
  coin <- sample(2,1)
  if (coin == 2) {
    x <- x + 1
  } else {
    x <- x - 1
  }
  sim[i,"x"] <- x
}
plot(sim$t,sim$x,type = "l")
```

Now I want you to adapt this code to simulate a 2-d random walk: instead of a coin, roll a 4-sided die, and the outcome determines the four moves (up, down, left or right). Then try simulating random walks of different lengths.

Add your code here.

C. jacchus

Final Programming Challenge: The Tornado Super Outbreak of 1974

Now we will practice our R skills on a much larger data set.

From the *University of Chicago Magazine*, Fall, 2020:

> Fujita published his proposed tornado scale in 1971, but it needed a high-profile event to take root. On April 3, 1974, a tornado touched down in Morris, Illinois, around noon. Over the next 17 hours, 148 confirmed tornadoes tore through 13 states and Ontario, Canada. Following the 1974 Super Outbreak—one of the worst tornado outbreaks on record—Fujita and his team took a whirlwind airplane tour of more than 10,000 miles, surveying the ruins.

Fujita's scale is now known the "F-scale", and it scores tornadoes from F0 to F5 based on wind speeds and ensuing damage.

Analysis aims

Our main analysis aim is to uncover evidence for the 1974 Tornado Super Outbreak in data from NOAA's Severe Weather Data Inventory (SWDI). In this programming challenge, you will be given some suggestions for how to proceed, *but you will (mostly) not be given the code.*

Import the data

The SWDI data are stored in a file "StormEvents_details-ftp_v1.0_d1974_c20220425.csv.gz" which is included in the GitHub repository.

Since the SWDI data are stored as a CSV file, you should now know what to do to import the data into R.

Add your code here.

Examine the data

Now that we have the data in a data frame, write some code to get an overview of the data frame: e.g., number of rows and columns, the names of the columns, and what types of data are contained in the columns.

Add your code here.

You will find that these seven data columns are most useful: EVENT_TYPE, BEGIN_DAY, MONTH_NAME, STATE, BEGIN_LON, BEGIN_LAT and TOR_F_SCALE. Write some code to examine these columns. **Hint:** You might find that the table() function is useful for this.

Add your code here.

Prepare the data

Your initial examinations should suggest a few improvements to the seven columns of the data frame we are interested in. Write code to make those improvements.

Hints: "month.name" is a built-in constant than may be useful. Also, " " produces the empty text value (a "string" of length zero).

Add your code here.

Since the focus is a particular type of storm event—tornadoes—extract the rows of the table about tornadoes.

Add your code here.

When and where did the tornadoes occur?

What calendar date (month + day) saw the most tornadoes? Let's call this date "the day of the Super Outbreak." **Hint:** Since there is no data column for "day of year", to answer this question you could create a new column (say, "dayofyear") from other columns using the paste() function.

Add your code here.

Which two states had the most tornadoes in 1974? **Hint:** The sort() function might be useful here.

Add your code here.

To understand the geography of the tornadoes in more detail, use the lat-long coordinates to plot the tornadoes on a map. First, this can be done very simply using the plot() function. (What geographical structures emerge from this plot?)

Add your code here.

For a better map, I have written a *custom function* that takes a data frame "latlongs" as input and outputs a map of the US with the geographic locations projected onto it.

The inputs are two numeric vectors of the same length. The output is a ggplot object.

```
map_usa_latlongs <- function (lats, longs) {
  dat <- data.frame(lat = lats,long = longs)
  return(ggplot(dat,aes(x = long,y = lat)) +
         geom_path(data = map_data("state"),
                   aes(x = long,y = lat,group = group),
                   color = "gray") +
         geom_point(shape = 20,size = 1) +
         theme_classic())
}
```

```
library(ggplot2)
# Add the rest of your code here.
```

Remove the outliers

Plotting the tornadoes by geographic location revealed some strange "outliers". Write some code to understand what these "outliers" are, remove them from the data frame, then create a new map of the tornadoes without these strange outliers (reusing map_usa_latlongs).

Add your code here.

Map the Super Outbreak

Now use map_usa_latlong() again to create a map of the tornadoes that occurred on the single day of the tornado Super Outbreak. Compare your map to https://en.wikipedia.org/wiki/1974_Super_Outbreak.

Add your code here.

E. coli

Additional R resources

There is of course much more to learn about R. If R ends up being important to your work, then you will need to do more to improve your R programming skills. Fortunately, there are many good resources out there. Here are a few that I've discovered over the years (including many that have been recommended to me by others).

Paul Torf's and Claudia Brauer's (very) short introduction to R. This is a great place to go next after this tutorial.

Software Carpentry provides introductory lessons on R.

Data Carpentry provides additional introductory lessons on R. Although these lessons overlap quite a bit with the Software Carpentry lessons, they are tailored to specific research disciplines and therefore may be more interesting.

R in a Nutshell is currently available through the U of C library and I think can be downloaded as a PDF. It probably isn't great for learning about R, but it can be helpful if you want to look up a specific topic. For example, I often use it to check what are the (many) different options for the "plot" function.

R for Data Science is a book available for free online that focusses on some of the more popular R packages developed by Hadley Wickham that provide additional tools for analyzing data. We used one of these packages—ggplot2—but widely used packages include tibble, dplyr and magrittr. Also, the book covers R Markdown in detail. There is also the ggplot2 book that covers ggplot2 in much more detail.

Introductory Statistics with R by Peter Dalgaard is another book available for free from the U of C library. I found it particularly helpful for learning how to use the various basic functions for statistical analysis such as lm(), glm() and anova().

workflowr is an R package that we—that is, a bunch of us at U of C—developed specifically for researchers to help them organize their data analyses and make it easier to share with others. workflowr = organized + reproducible + shareable data science in R.

The R packages book is another book that is freely available online. It is for people interested in developing their own R package. It is quite advanced, but I mention it because a few of you may end up creating an R package as part of your research project. Also, for more advanced usage—say, if you are analyzing very large or very complex data sets—I recommend the Advanced R book.