Lab_1R (R Vectors)

36-290 – Statistical Research Methodology

Week 1 Thursday – Fall 2021

Preliminaries

Goal

The point of this lab is not to teach you how to code `R`, per se, but rather to expose you to elements of `R` coding that you may be using repeatedly throughout this semester. Thus the focus is pretty narrow: this lab is about working with vectors. If you group (or "column bind" or `cbind()`) a bunch of equal-length vectors together, you get a data frame, and data frames are how data are stored for analyses. So as you learn how to manipulate vectors, you are also learning how to manipulate columns of data frames.

Instructions

In every question, you will see a code chunk. A code chunk begins witha line with ```{r} (three backquotes, a left curly bracket, an r, and a right curly bracket), (a) line(s) containing code, and a final line with ```. Here is an example that only truly makes sense if you are looking at the `R Markdown` file itself:

```
print("Hello, world.")
```

```
## [1] "Hello, world."
```

What you will see is an invitation to "FILL ME IN." You should delete that line and replace it with the code necessary to answer the question at hand. To run your edited chunk to see if it works, simply put your cursor *inside* the chunk and, e.g., select "Run Current Chunk" from the "Run"

Preliminaries

pulldown tab. Alternately, you can click on the green arrow at the upper right-most part of the chunk, or use "<cntl>-<return>" as a keyboard shortcut.

Note that in some questions you will also see a "verbatim" text chunk. This looks like a code chunk but lacks the {r}. Material written in such a chunk is rendered verbatim upon knitting, and thus this is where you place answers to qualitative (as opposed to coding) questions. For instance, in Q1 below, I ask what you see when you type `x[4]`; you would place your answer in the provided verbatim text chunk.

When you have finished (or as you are working through) the lab, click on the down-arrow next to the `Knit` button, and on the pull-down menu click on `Knit to PDF`. This should output an `pdf` file; if you cannot find that file, go to the console and type `getwd()` (i.e., "get working directory")…you may find that your working directory and the directory in which you've placed the `Rmd` file are not the same. The `pdf` file should be in the working directory. If your system is not set up properly to output the `pdf` files seamlessly, click on `Knit to HTML` instead and use a web site such as `html2pdf.com` to convert the `html` output to `pdf`.

`R Markdown` may prompt you to install packages for knitting to work; go ahead and install these.

Question 1

Let's start with some basic vector manipulation. Use the `c()` function to define a vector `x` with four values: 1, 4, 1, and 9. (You should replace the "# FILL ME IN" statement below with your answer.) Note that vectors are homogeneous (all of the same type), with the most common types being `double` (or `numeric`), `character`, and `logical`. The vector you define has type `double`. Check this by typing `typeof(x)` and noting the output. Then type `x[4]`. What do you see?

```
x <- c(1, 4, 1, 9)

typeof(x)
```

```
## [1] "double"
```

```
x[4]
```

```
## [1] 9
```

```
When I type x[4] I see that it returns the number 9.
```

If the value(s) inside the square bracket is/are numeric, then that/those elements of the vector are displayed. (Note: `R` counts from 1, not 0.) If the value(s) are logical, then only those elements with value `TRUE` are displayed. This will make more sense below.

Question 2

Use the `append()` function to extend the vector `x` with new values 2, 3, 4, and 5.

```
x <- c(1, 4, 1, 9)
x <- append(x, 2:5)
```

Question 3

Initialize a vector `y` with one logical value, one numeric value, and one character value, and determine the type of `y`. Write down what type it is.

```
y <- c(TRUE, 1, "character")

typeof(y)
```

```
## [1] "character"
```

```
It says it "character"
```

Preliminaries

Question 4

Relational operators are binary operators of the form "variable operator value," e.g., `x < 0`. The six basic relational operators are `==`, `!=`, `<`, `>`, `<=`, and `>=` (for "equals," "not equals," "less than," "greater than," "less than or equals," and "greater than or equals.") Relational operators return a vector of logicals, meaning a vector of `TRUE` and `FALSE` values. Below, display the output for `x == 1` and `x > 3`.

```
x == 1
```

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

```
x > 3
```

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

Question 5

Apply the `sum()` function with input `x == 1`. Does the output make sense to you?

```
sum(x == 1)
```

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

```
Yes, it seems to be referring to how many values in vector x are equal to 1.
```

Question 6

Via code (i.e., not by hand), determine the proportion of values of `x` greater than 2.

```
x > 2
```

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

```
sum(x > 2)
```

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

```
5/8
```

```
## [1] 0.625
```

Question 7

Using the `which()` function, display the indices of `x` for which the associated data are less than 4.

```
which(x < 4)
```

```
## [1] 1 3 5 6
```

Question 8

Relational operators may be combined with `&` (logical AND) or `|` (logical OR). Below, display the output for `x < 2 | x > 5`.

```
a = x < 2 | x > 5
x[a]
```

```
## [1] 1 1 9
```

Question 9

A reason to learn relational operators is that they underpin the manipulation of vectors (and thus underpin the manipulation of, e.g., rows or columns of data frames). To display a subset of values of the vector `x`, you can for instance type `x[...]`, where you would replace `...` with a relational operator. What happens when you type `x[x==1]`?

```
x[x==1]
```

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

```
Displays the ones in the data set
```

Question 10

Apply the `length()` function to `x`, apply the `sort()` function to `x`, apply the `sort()` function to `x` with the additional argument `decreasing=TRUE`, apply the `unique()` function to `x`, and apply the `table()` function to `x`. Build intuition about what each does. (Note that `table()` is a handy function for doing exploratory data analysis of categorical variables.) To see the help page for each function, you can go to the `Console` pane in `RStudio` and at the prompt type a question mark immediately followed by the function name (e.g., `?sort`). You can use help pages to determine what optional arguments exist for each function and what their default values are. For instance, what is the default value for `decreasing` when sorting?

```
length(x)
```

```
## [1] 8
```

```
sort(x)
```

```
## [1] 1 1 2 3 4 4 5 9
```

```
sort(x, decreasing = TRUE)
```

```
## [1] 9 5 4 4 3 2 1 1
```

```
unique(x)
```

```
## [1] 1 4 9 2 3 5
```

```
table(x)
```

```
## x
## 1 2 3 4 5 9
## 2 1 1 2 1 1
```

```
False
```

Question 11

A *list* in `R` is a collection of vectors. Define a list below using `list()`, with the first entry having name `x` and values 1 and 2, and the second entry having name `y` and values “a”, “b”, and “c”. Display the list.

```
x <- c(1,2)

y <- c("a", "b", "c")
```

```
my_list <- list(x, y)

my_list
```

```
## [[1]]
## [1] 1 2
##
## [[2]]
## [1] "a" "b" "c"
```

The individual entries of a list are vectors, which are homogeneous, but the entries may each be of different type. A list whose entries are all of the same length is a data frame.

Let's import some data from the 36-290 GitHub site. These data are stored in `Rdata` format (which is equivalent to `rda`, if you've seen that file extension); such data are saved via `R`'s `save()` function and loaded via `R`'s `load()` function. One wrinkle here: the data are stored on the web, so we also have to apply the `url()` function.

```
rm(list=ls())   # This is generally good practice: remove all current variabl
es from the global environment.
file.path = "https://raw.githubusercontent.com/pefreeman/36-290/master/EXAMPL
E_DATASETS/PHOTO_MORPH/photo_morph.Rdata"
load(url(file.path))
rm(file.path)
objects()       # Shows the loaded variables. (Redundant with the Environment
pane.)
```

```
## [1] "predictors" "response"
```

If everything loaded correctly, you should see two variables in your global environment: `predictors` and `response` (loaded from the file). `response` is a vector of length 3419, and it represents *redshift*, which you can think of as a directly observable proxy for the distance of a galaxy from the Earth. (After all, tape measures aren't going to help us here.)

Since you'll hear me talk about redshift for the entire semester, I may as well explain the concept here.

The Universe is expanding, and has been since the Big Bang (some 13.8 billion years ago). The expansion was at one point decelerating, but the action of dark energy is now causing the expansion to accelerate. (Dark energy is an unknown source of "repulsive gravity" that comprises approximately 70% of the mass-energy budget of the Universe. We can infer its presence, and you can think of it as a "parameterization of ignorance," i.e., we can model it, but we don't know what it is…kinda like gravity between the times of Newton and Einstein.) The wavelengths of photons are tied to the expansion, so a photon emitted from a far-away galaxy will have a longer wavelength when it reaches our detectors. Redshift (generally denoted z) is related to emitted and observed wavelengths as follows:

$$\frac{\lambda_{\rm obs}}{\lambda_{\rm emit}} = 1 + z$$

Redshifts over the domain of galaxies range from zero (right here in the Milky Way) to approximately 10. We have a lot of galaxy data for redshifts $\leq$ 0.5 or so from the Sloan Digital Sky Survey and a fair bit for redshifts $0.5 - 3$ from other smaller surveys.

Question 12

Determine, via Google, how one would get a five- (or six-) number summary of the response vector (containing the minimum value, the median value, etc.). Why Google? Because Google is your documentation friend that often (not always, but often) leads you to that well-populated Internet paradise, StackOverflow. (That probably won't happen here…) While you are at it, get the interquartile range, and check to see if it matches what you would compute from a five- (or six-) number summary.

```
summary(response)
```

Preliminaries

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0001  0.7340  1.0210  1.3032  1.6101  5.7860
```

```
IQR(response)
```

```
## [1] 0.8761
```