Lab 1: R Basics

STAT218

This lab is intended to introduce you to the basics in R that you will need for this class. Most of our analyses will consist of just a few steps:

1. load a dataset
2. identify and select variable(s) of interest
3. perform one or more calculations using variable(s) of interest as inputs

We will illustrate this process so that you can get used to the mechanics and familiarize yourself with how different data types appear in R.

### How to do this lab

I’ve provided you with a project on Posit Cloud containing data files and a script (a script is a plain text file containing R commands). The script contains all commands shown in this document, and some blank areas for you to fill in, with comment lines (the ones starting with #) to help you navigate.

You should refer back to this document for instructions and context, and fill in the script as you go:

* run the codes provided as you read through the narrative in this document and inspect the results
* in the ‘your turn’ sections, refer to the prompt in this document and use the example commands provided immediately beforehand to determine which command to write
* write in your commands **in the script** the space below the corresponding comment, **not in the console** (otherwise you’ll have a hard time keeping track of your work)

Your goal is to complete all of the “your turn” parts in the script. Two practice problems are given at the end of the lab as homework for you to complete on your own before next class.

### How to use this lab

This lab (and the lab activities in general) are designed to provide you with a set of examples to learn initially in class and then follow on your own later when doing the homework problems given at the end of the lab.

If you can do the examples and ‘your turn’ activities in class, all you’ll need to do to complete the homeworks is copy commands from those examples and activities and adjust some small details (variable names, dataset names, etc.).

If you later need to figure out how to do something in R for a homework problem or test, all you’ll need to do is refer back to the labs.

### Packages in R

A “package” is a bundle of functions, datasets, and other objects that can be imported into R for use in your working environment. Many scripts begin by loading packages that will be used throughout the script. Packages are loaded using the command library(<PACKAGE NAME>) where <PACKAGE NAME> is replaced by the actual name of the package. For example:

library(tidyverse)

Packages do need to be installed before they can be loaded. One of the nice things about using Posit Cloud is that I can manage all of these installs for you. However, if you ever wish to install and use a package that’s not available (or if you use R on your own machine), you can install a package using the command install.packages("<PACKAGE NAME>") after replacing <PACKAGE NAME> with the actual name of the package (but keeping the quotation marks!).

### Loading a dataset

There are several ways to load datasets in R. The strategy we’ll use most often is to load an .RData file, but you will encounter a few others here and there.

# load nhanes data  
load('data/nhanes.RData')

This command looks for a file called nhanes.RData in a directory folder named data and reads the file.

Notice that once you run the command, an object called nhanes appears in the “Environment” tab in the upper right hand panel of your RStudio window.

If you click the little blue carrot next to nhanes in the environment tab, you will then see a list of variables contained in the dataset. You can also see the first few rows of the dataset using head(...).

# first few rows  
head(nhanes)

# A tibble: 6 × 9  
 subj.id gender age poverty pulse bpsys1 bpdia1 totchol sleephrsnight  
 <int> <fct> <int> <dbl> <int> <int> <int> <dbl> <int>  
1 1 male 34 1.36 70 114 88 3.49 4  
2 2 male 34 1.36 70 114 88 3.49 4  
3 3 male 34 1.36 70 114 88 3.49 4  
4 5 female 49 1.91 86 118 82 6.7 8  
5 8 female 45 5 62 106 62 5.82 8  
6 9 female 45 5 62 106 62 5.82 8

This kind of object in R is called a **data frame**. Data frames are displayed in a tabular layout, like a spreadsheet. While data frames *should* be arranged so that observations are shown in rows and variables in columns, this is not guaranteed, so you should be in the habit of checking to make sure the layout is sensible; otherwise, you might accidentally perform bogus calculations and analyses.

Beyond providing a sanity check, inspecting the data frame will show you three key pieces of information besides the values of the first few observations of each variable.

1. Data dimensions: how many observations (rows) and how many variables (columns)
2. Variable names: subj.id, gender, age, etc.
3. Data types:
   * int for integer (numerical data type)
   * fct for factor (categorical data type)
   * num for numeric (numerical data type)
   * chr for character (categorical data type)

So, for example, seeing that pulse is of data type int tells you that pulse is a discrete numerical variable. It also tells you what name to use to refer to the variable in subsequent R commands.

|  |
| --- |
| Your turn |
| There is another data file in the data directory called famuss.RData. Load this into the environment, preview the first few observations, and check the variable names and data types.  # load famuss dataset  # preview first few rows  To check your understanding:   * how many observations and variables? * identify a categorical variable * what kind of variable is bmi? |

### Selecting variables

The variable names in a dataset can be used to retrieve or refer to specific variables. For example, try running this command:

# extract total cholesterol  
total.cholesterol <- nhanes$totchol  
  
# preview first few values  
head(total.cholesterol)

[1] 3.49 3.49 3.49 6.70 5.82 5.82

That command did the following:

* extracted the totchol column of nhanes (the nhanes$totchol part)
* *assigned* the result a new name total.cholesterol (the <- part)

Assignment (<-) is a very important concept in R – you can store the result of any calculation as an object with a name of your choosing.

You’ll notice that total.cholesterol looks a bit different than the data frame in terms of its appearance. This is because it’s not a data frame but rather a different kind of object called a **vector**: a collection of values of the same data type.

|  |
| --- |
| Your turn |
| Extract the body mass index variable from the FAMuSS dataset, and store it as a vector called bmi.  # store the change in arm strength variable as a vector called 'strength'  # preview the first few values |

### Performing calculations

Extracting and storing variables as vectors isn’t strictly necessary, but does make it easier to perform many calculations. While you’re a beginner, I’d recommend using this strategy.

#### Numeric summaries

Most simple summary statistics can be calculated using simple functions in R that take a single vector argument. For example, to calculate the average, minimum, and maximum total cholesterol among the respondents in the sample:

# average total cholesterol  
mean(total.cholesterol)

[1] 5.042938

# minimum  
min(total.cholesterol)

[1] 2.33

# maximum  
max(total.cholesterol)

[1] 13.65

|  |
| --- |
| Your turn |
| Find the average percent change in nondominant arm strength of participants in the FAMuSS study sample using the strength vector you created before.  # compute mean change in nondominant arm strength |

#### Categorical summaries

Most data summaries for categorical variables proceed from counts of the number of observations in each category. These counts can be obtained by passing a vector of observations to table(...):

# retreive sex variable  
sex <- nhanes$gender  
  
# counts  
table(sex)

sex  
female male   
 1588 1591

To obtain the proportion of observations in each category – the counts divided by the total number of observations – pass the table to the proportions(...) function:

# proportions  
table(sex) |> proportions()

sex  
 female male   
0.4995282 0.5004718

The character string |> is a bit of syntax that you could read verbally as ‘then’: first make a table, *then* obtain proportions. It’s known as the pipe operator, because it ‘pipes’ the result of the command on its left into the command on its right.

To see another example of the pipe operator in action, you could rewrite the previous command as a chain of three steps:

# same as above  
sex |> table() |> proportions()

sex  
 female male   
0.4995282 0.5004718

You could interpret this as follows: start with sex, pass that to table(), then pass the result to proportions.

|  |
| --- |
| Your turn |
| Using the FAMuSS dataset, calculate the genotype frequencies in the sample (*i.e.*, find the proportion of observations of each genotype).  # retrieve genotype  # counts  # proportions |

While the analyses you’ll learn will get more complex than computing summary statistics, the mechanics of performing the computations in R will be analogous to what you just did: executing a one-line command with a vector input.

### Putting together the pieces

Reflect for a moment on what you just did: you wrote a few lines of code to import a dataset, extract a variable, and compute a statistic. If you filled in the script as instructed, you now have a record of the commands you executed that you can use to retrace your steps.

In fact, anyone with your script and the data files (including future you) could easily reproduce your work. Reproducibility is a pillar of data-driven science; by storing analyses in the form of executable scripts, researchers can easily create and share records of their work.

We could put the steps above together in just a few lines as if it were a short script. Typical style is to provide line-by-line comments explaining what the commands do.

# import nhanes data  
load('data/nhanes.RData')  
  
# inspect data  
head(nhanes)  
  
# extract total cholesterol  
total.cholesterol <- nhanes$totchol  
  
# compute average total cholesterol  
mean(total.cholesterol)  
  
# extract sex  
sex <- nhanes$gender  
  
# proportions of men and women in sample  
table(sex) |> proportions()

|  |
| --- |
| Your turn |
| Follow the example above and combine the previous exercises into a few lines of code with appropriate line comments.  # load famuss dataset  # inspect data  # extract nondominant change in arm strength  # compute average change in strength  # extract genotype  # compute genotype frequencies (proportions) |

If this was all entirely new to you, congratulations on writing your first lines of code!

### Extras

#### Reading CSV files

Often data are stored in spreadsheets, which can be easily converted to comma-separated values or CSV files (extension .csv). These are plain-text files that are a bit more lightweight than an Excel spreadsheet.

R can read CSV (as well as other) files. The read.csv(...) function will parse the file and produce a data frame. The result can be assigned a name and stored as an object in the environment.

# parse a csv file  
read.csv('data/gss.csv')  
  
# store the result in the environment  
gss <- read.csv('data/gss.csv')

Most of the time in class we’ll load .RData files or obtain datasets through packages (more on this later), but if you use R outside of class you may find it more common to manage data input via .csv files.

#### More about R

While you will learn new commands going forward, we won’t go much more in depth with R than what you just saw. However, if you’re interested in understanding the above concepts in greater detail, or learning about R as a programming environment, see [An Introduction to R](https://intro2r.com).

### Practice problems

Due before the next class meeting.

1. The census dataset contains a sample of data for 377 individuals included in the 2000 U.S. census. Load and inspect the dataset, and determine:
   1. the youngest and oldest individual in the sample
   2. the average total personal income
   3. the average total family income
   4. how many variables are in the dataset, not including census year and FIPS code
   5. how many *categorical* variables are in the dataset, not including FIPS code
2. The cdc.samp dataset in the oibiostat package contains a sample of data for 60 individuals surveyed by the CDC’s Behavioral Risk Factors Surveillance System (BRFSS). Use the provided commands to load the dataset, and then inspect it the usual way. Notice that several of the variables are 1’s and 0’s. Use the command ?cdc.samp to view the data documentation.
   1. What do the values (1’s and 0’s) mean in the exerany variable?
   2. What proportion of the sample are men? What proportion are women?
   3. For each general health category, find the proportion of respondents who rated themselves in that category.
   4. How many of the respondents have health coverage? (*Hint:* sum(x) will add up the values in a vector x; adding up a collection of 1’s and 0’s is equivalent to counting the number of 1’s.) What percentage of the respondents have health coverage?