214B: Introduction to R

Winter 2020

TA: Melissa Gordon Wolf

Learning R

You should think of learning R as learning a new language. As such, a 50 minute lab will not be enough for you to learn a new language - it will require lots of practice. This lab is set up to give you the tools you'll need to begin learning a programming language.

Because everyone is in a different spot in terms of their R knowledge, I'm going to ask you to get in groups and work together on this lab. This material combines the material from the 214A labs, 214A R solution sets, an SPSS to R bookdown called Rosetta Stats that I have the pleasure of working on, and the *awesome* **R advent calendar** that I recommended during the last lab. In other words, some of this material is our original material but the rest is a bunch of existing resources compiled together for our purposes. I've tried my best to note when the material in this lab is not orginal.

Go forth, and raise your hand if you have any questions!

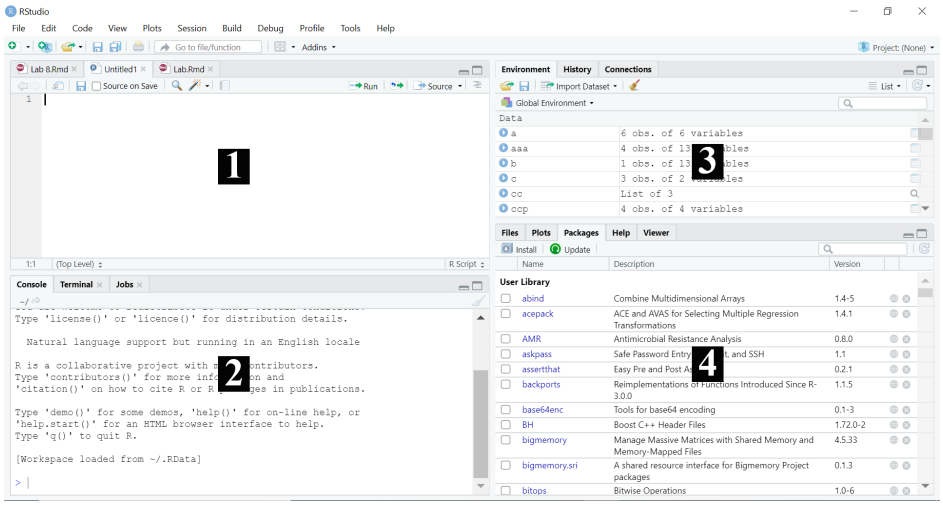
Getting R up and running

1. Download R and RStudio

To use R, you first need to download R (https://www.r-project.org/) and then download RStudio (https://rstudio.com/products/rstudio/). The computers in the lab should already have these installed. Go ahead and open up RStudio (you will almost never open R - you will always interact with it through RStudio).

2. Interacting with RStudio

See Day 2 of Kiirsti Owen's R Advent calendar



RStudio is an interface that uses four tabbed panes to facilitate interactions with R.

- The top-left pane (1) is called the "source" window. It contains the analysis script, or multiple scripts, you are working on (aka, it is where you will type your code). You will typically want to save your script (i.e. everything in this window) fairly regularly.

- The bottom-left pane (2) contains the R console. When we type run code in our script (displayed in our source window), you should see some sort of output in the console. Some examples: If you run a code and there's something wrong with it, this is where you will see the error message. If you run a statistical test, this is where the results will be displayed.
- The top-right pane (3) is the "environment" window. It contains an overview of all datasets and other objects you have loaded, as well as a history of the commands you provided to R. Any objects you create will be added to this list.
- Finally, the bottom-right pane (4) provides easy access to an overview of all files in your project, the plots you produced, the R packages you have installed and loaded, and the online manual of R and your packages.

3. Setting a working directory

A working directory is just a fancy name for a folder on your computer. Wherever you save your R script will establish be your working directory. However, you can also set your working directory in R. You can just copy and paste this from your file explorer (PC), although you'll have to switch the back slashes to forward slashes. You can check what your current working directory is by using the function **getwd()**.

```
setwd("~/Users/Melissa/Documents/UCSB/214/Lab 8")
getwd
()
```

Learning to Code

1. Basics

Copied from Day 4 of Kiirsti Owen's R Advent Calendar

Here's some basic code you can run. Just like in Excel or on a calculator, R can do simple calculations for you so that you don't have to think too hard!

To run each line at a time, click anywhere in the line of code and run each line. To run them all at once, highlight the entire code and run it. **R shortcut: to run code press Ctrl + enter (PC) or Command + enter (Mac).**

```
2+5
2^5
67^8
log(25)
sqrt(25)
abs(-25)
```

P.S. log(), sqrt(), abs() are examples of functions

2. Vectors

Copied from Day 6 of Kiirsti Owen's R Advent Calendar

Instead of a single number or character, vectors allow us to group many together.

For example, run this line of code:

```
numbers <- c(1,2,3,4,5)
```

Now highlight just the word 'numbers' or retype it below and run it. Your numbers should come up in the output. Just like that! You just created a vector. This is called a numeric vector. Note that the numbers do not need to be in " ".

The other type of vector is called a character vector. Here's a fun one:

```
hi <-c("Happy holidays, I hope you have a wonderful day!")
```

Now type hi and run that (or highlight the word hi in your console and run). Note that the words DO need to be in " ". Note: if your vector includes words and numbers, R will consider everything as characters.

Vectors make our lives easier when we don't want to type out long lists of code (especially if the vector is an entire dataset!).

As an example of what we can do with a vector, let's do something with your first vector:

```
sum(numbers)
```

Vectors are super useful, you will use them a lot.

R fun fact: the c in c() stands for "concatenate" or 'to link together'.

3. Operators

Copied from Day 7 of Kiirsti Owen's R Advent Calendar

You're already seen <- (referred to as the 'assignment operator'). Here are some useful logical operators you might need to use.

- < less than
- <= less than or equal to
- > greater than
- >= greater than or equal to
- == exactly equal to
- != not equal to

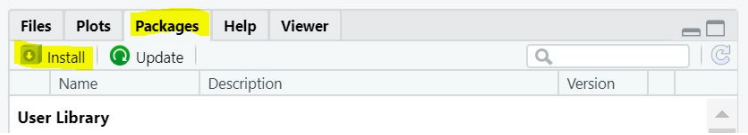
Try them out with these codes:

```
x <-2
y <-10
x<y
x>y
y>=100
```

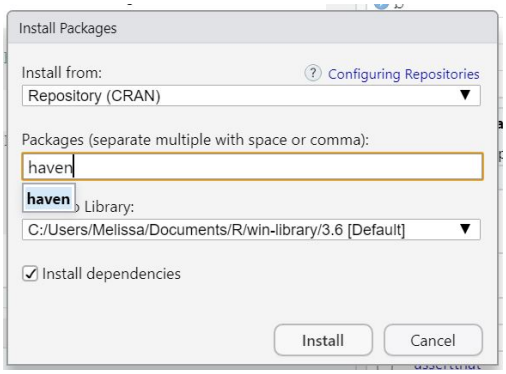
Installing libraries and reading data into R

1. SPSS Data

In this class, you're often given datasets as an SPSS file. To get a dataset into R, we'll use the point-and-click interface in R. First, we need to install a package to read the data in from SPSS. On the right, click on **Packages** > **Install**.



Type in **haven**, check **Install dependencies** and press **Install**. Everything in R is case-sensitive, including library names.

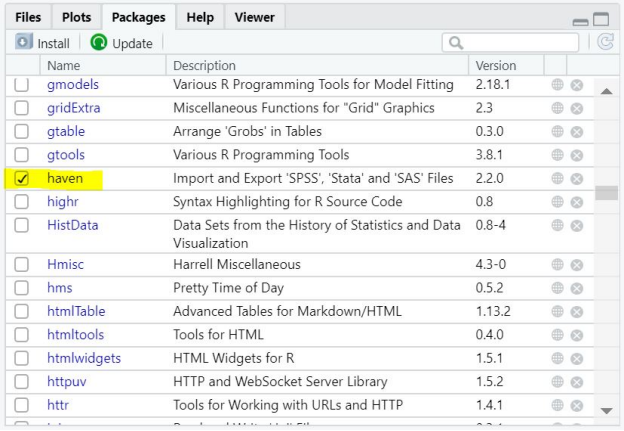


You can also do this in the R script:

```
install.packages("haven")
```

Note the quotations around the package name.

Once a library is installed, you'll never have to install it again. However, you will still have to **load** the library every time you open R. You can do this using the point-and-click interface in R. Scroll down under packages, and check the box next to **haven**.



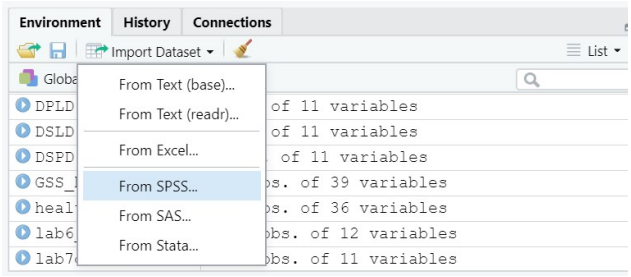
If the library you want to load does not appear in your list of packages, it is because it has not been installed.

You can also do this in the R script:

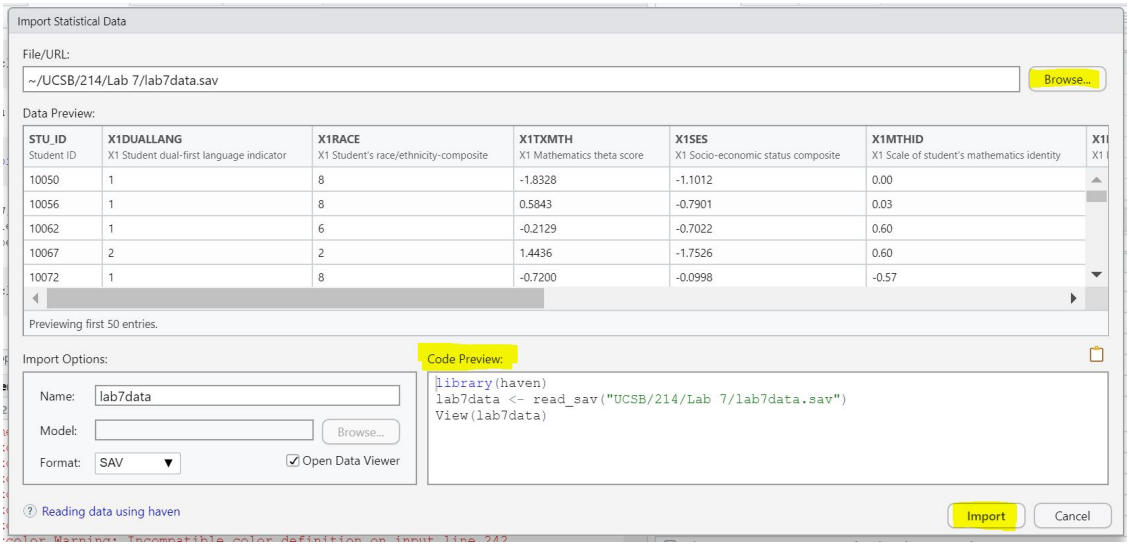
```
library(haven)
```

Note that there are no quotations around the package name.

Now, let's import the dataset. We'll use the same dataset from Lab 7 in 214A, titled "lab7data.sav" (you should have downloaded this from gauchospace). On the right, select **Import Dataset** and select **From SPSS**. As you will see, you can read several different file types into R.



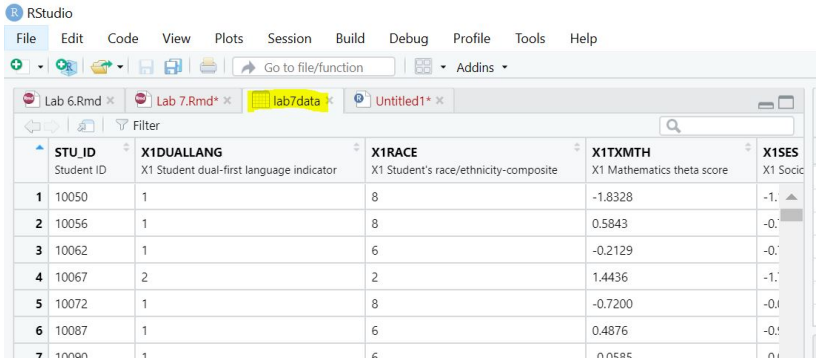
Select **Browse** and find the SPSS dataset. When you select it, you'll see it populate in the **Data Preview** window. Also, notice that R has automatically created the code you would need to read the dataset in under **Code Preview**. Select **Import**.



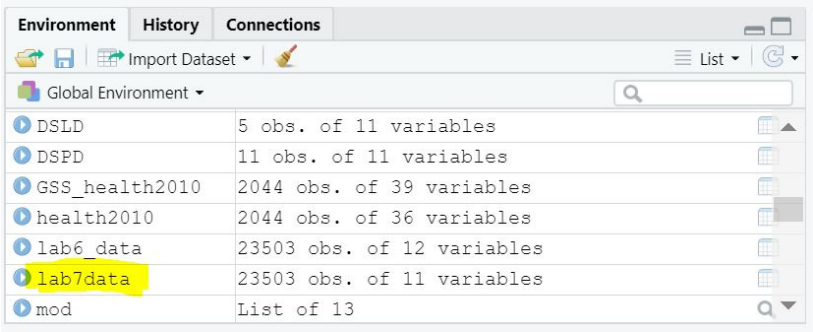
Alternatively, you can run the code that R generated (see below). However, I almost always use the point-and-click interface here.

```
library(haven)
lab7data <- read_sav("C:/Users/Melissa/Documents/UCSB/214/Lab 7/lab7data.sav")
```

Now, your dataset is in R! You'll notice that you can toggle back and forth between your R script and the dataset.



You can also locate (and open) the dataset from the Global Environment.

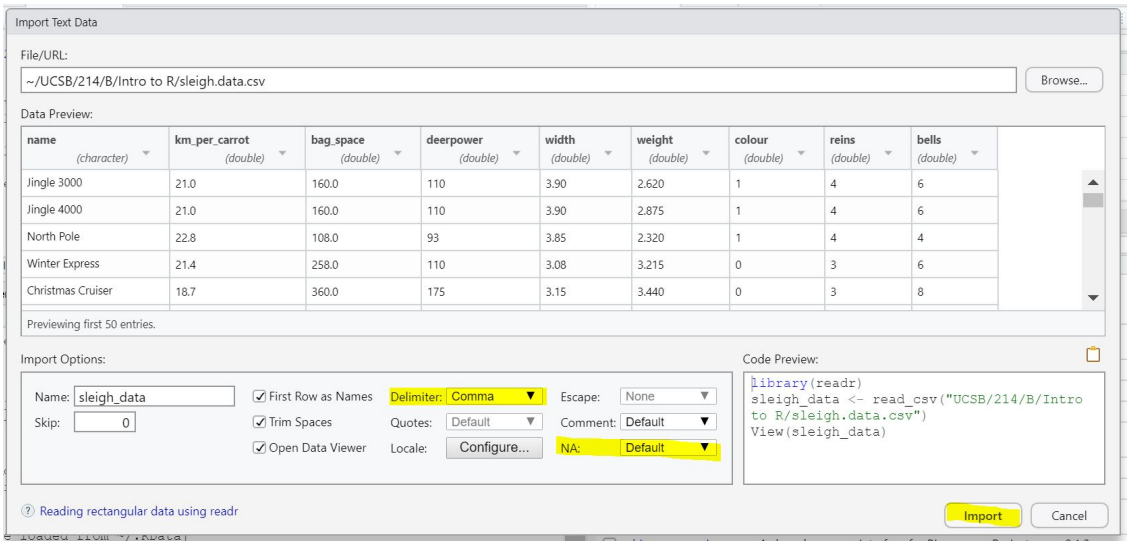


2. CSV files

See Day 8 of Kiirsti Owen's R Advent Calendar

In the real world, you will get datasets in many different formats. Let's load in a CSV file that Kiirsti Owen created for the R Advent calendar. The CSV file is "sleigh.data.csv".

Follow the same steps as before, but this time select "From Text **(readr)**" under "Import Dataset". Navigate to wherever you saved the CSV file and select it. **Make sure that the delimiter is a comma. Change the delimiter to "white space". What happens in the data preview? Switch it back to comma to read the data in correctly.** Note that R will automatically install the "readr" package necessary to read in the csv file.



This dataset includes nine columns and 34 rows. The columns include:

- name = name of the sleigh model (there are 33 rows of sleigh models).
- km_per_carrot = how many kilometres the sleigh can go based on reindeer energy (measured in carrots)
- bag_space = the amount of space in the back of the sleigh for Santa's toy bag. Units aren't important.
- deer power = similar to horsepower in cars
- width = width of the sleigh in meters
- weight = weight of the sleigh in grams (sleighs are magical and thus weigh VERY little!)

- colour = 0 is the code for green and 1 is the code for red. Santa wouldn't even consider any other colours.
- reins = the number of reins the sleigh is equipped with
- bells = number of magical bells on the sleigh.

Working with data

1. Viewing the dataset

Copied from Day 9 of Kiirsti Owen's R Advent Calendar

Let's start by renaming the dataset to something shorter.

```
sl.dat<-sleigh_data
```

Now, let's have a look at just the column names so we are reminded of the position of each column.

```
colnames(sl.dat)
```

You can do the same for rows, which isn't as meaningful here, but could be with other datasets.

```
rownames(sl.dat)
```

We want to isolate the column that looks at how much deerpower the sleighs have. There are several ways we can do this, one way is to identify the column position.

```
sl.dat[,4]
```

Note we use [ ] instead of ( ).

Now we want to look at a single row. Let's look at the Christmas Cruiser This is the 5th line down in our dataset.

```
sl.dat[5,]
```

Let's say we just want to call up how much bag space the Christmas Cruiser has and ignore the other stuff.

```
sl.dat[5,3]
```

What if we want to call a certain variable from a dataset? To do this, we will use the $ operator. For example:

```
dataset$variable
```

If you ran this line of code, you would get a list of every value in this variable. The intent is not for you to run this, just to show you how to call a variable from a dataset!

2. Descriptives

Before we analyze data, we always want to check out the descriptive statistics. Let's begin by looking at the descriptive statistics for the continuous variable, deerpower. We can use a **Base R** function called **summary**.

```
summary(sl.dat$deerpower)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##    52.0    96.5   123.0   146.7   180.0   335.0       1
```

Alternatively, we could use **describe** from the **psych** package. First, install and load the library:

```
install.packages("psych")
library(psych)
```

Next, run the command:

```
describe(sl.dat$deerpower)
```

```
##    vars  n   mean    sd median trimmed  mad min max range skew kurtosis    se
## X1    1 32 146.69 68.56    123  141.19 77.1  52 335   283 0.73    -0.14 12.12
```

This package gave us a lot more information than base R did.

3. Frequencies

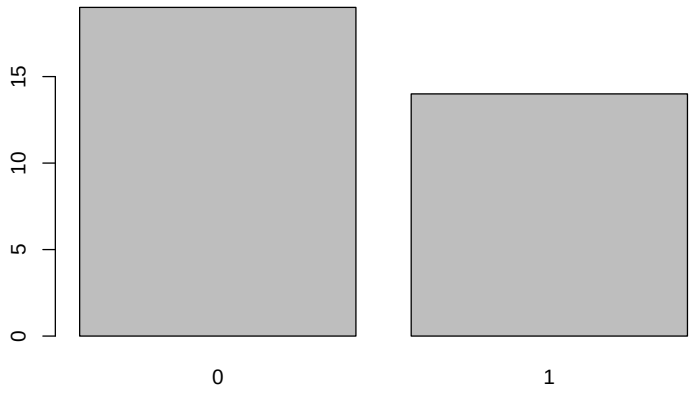
Let's use the **descr** package to create a frequency table. By default, the **freq** command in the **descr** package will also create a basic bar graph.

First, install and load the library:

```
install.packages("descr")
library(descr)
```

Next, run the command for the colour variable. Remember from before that 0 = green and 1 = red.

```
freq(sl.dat$colour)
```




```
## sl.dat$colour
##       Frequency Percent
## 0            19   57.58
## 1            14   42.42
## Total        33  100.00
```

4. Recoding variables

Let's say that we want to recode 0 into green and 1 into red to better visualize the frequency table and bar graph. We'll use the **recode** command from the **car** package.

First, install the package:

```
install.packages("car")
library(car)
```

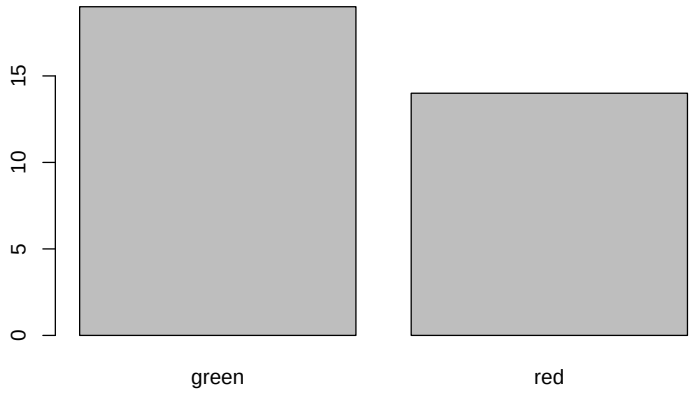
Next, let's run the recode command below. Here is what the command says in words:

1. Recode the variable colour from the sl.dat dataset: `recode(sl.dat$colour,`.
2. Change 0 to green, and 1 to red: `"0='green';1='red'")`.
3. Save that recoded variable as a new variable in the same dataset: `sl.dat$colour_r<-`.

- *There will be no output from these steps.*

4. Next, run frequencies from the psych package for the new variable to check that it worked: `freq(sl.dat$colour_r)`.

```
sl.dat$colour_r<-recode(sl.dat$colour, "0='green';1='red'")
freq(sl.dat$colour_r)
```




```
## sl.dat$colour_r
##       Frequency Percent
## green        19   57.58
## red          14   42.42
## Total        33  100.00
```

Continuing on

Congratulations! You've almost made it half way through the R Advent calendar. Want to learn more? Go to: **https://kiirstio.wixsite.com/kowen/post/the-25-days-of-christmas-an-r-advent-calendar** and start on **Day 10**. You may find some material that is redundant to what we've already covered - if so, skip ahead!