Lab 2: Working with datasets

PSC 103B - Spring 2024

Installing and loading packages

There are many functions available in base R. By this I mean, if you open R right after installing it, there are a number of functions already available for you to use. All the functions we've used in last week's lab are base R functions. They're available as soon as you install R.

But we can add more functions (other than the ones available in base R), by installing and loading packages that contain different functions.

There's something important to understand about packages. There is a difference between installing and loading a package. First, you have to install the package. This downloads it onto your computer and puts it somewhere R can find it. You only have to install a package once (unless you uninstall and install R again).

We can use the `install.packages()` function to do this. The name of the package goes in the parenthesis and must be **inside quotation marks**!

```
install.packages("tidyverse", dependencies = TRUE)
```

`dependencies = TRUE` will tell your computer to also install any packages that package needs to work properly. This usually makes for a smoother installation process. While you are doing this, you may be prompted to answer a question with yes (y) or no (n). You should answer that by typing y in the console and pressing enter. It's generally safe to answer the questions with yes if you don't understand them.

After installing the package it is now on your computer. And you can easily use it **after** loading the package. Every time you open R you must load (But not install!) all the packages you want to use. You can use the library() function to do this. And you don't need quotation marks.

```
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.3     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.0
## v ggplot2   3.4.4     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.0
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

Summary:

- A package must be installed before it can be loaded (use `install.packages()`).
- A package must be loaded before it can be used (use `library()`).

Tidyverse

In R, there are always multiple ways to do the same thing, so you might see code that looks different than what you're about to learn, but does the same thing. That's okay. Today we'll learn how to read data into R (for example, an excel file or other types of data you might have on your computer) and how to select and change the information on that dataset. We'll do all this using a set of packages called tidyverse. Tidyverse is nice because the functions are more intuitive, and we hope that will make it easier to learn. There are also plenty of online resources that can help you when you run into problems (which you inevitably will – this is just a normal part of coding!).

So if you haven't done that yet, go ahead and load tidyverse using the `library()` function.

Working Directory & Reading Data into R

There are different kinds of datafiles that you might want to read into R. Different data files often require different code to be read in successfully. Today we'll cover how to read in one kind of data file In the future, if you need to read in another kind and this code isn't working... Google it! An example google search... "How to read a .rda file in R".

You should have all downloaded a the datafile 'Lab2Data.csv' from Canvas. We'll go over how to load this dataset into R.

(If you are using RStudio Could, this process will be a little different, but not too complicated. You'll need to first upload the dataset you downloaded from canvas into RStudio Cloud using the upload button that's in the Files tab on the lower right hand corner of your screen.)

First we need to set the working directory. We need to tell R where to look for the file we want to read into R. So we set the working directory to the folder where the data file is. If you just downloaded the datafile it'll likely be in your Downloads or Desktop folder. Check where the file is before setting the working directory!

We can use the `setwd()` function. We can put the path to the folder inside the function. **IMPORTANT**: This path *MUST* be within quotations.

```
# Example
setwd("C:/Users/marwin/OneDrive/Documentos/Rprojects/phd/24-Spring/psc103b-sq24/lab2")
       # You will need to change this ^^^ path!
```

The `getwd()` function tells you what folder is currently the working directly. This is helpful to check that the working directory is set to the folder you want.

Another way to change the working directory in RStudio is by using the point & click options. Go to Session > Set Working Directory > Choose Directory. This will open up a new window. Navigate to the folder you want to be your working directory. When you get to that folder, click the option 'Open'. This also works for RStudio Cloud in a similar way. You can use `getwd()` to check that it worked.

The kind of file we have now is a CSV file, or a comma separated values file. This is just a simplified type of spreadsheet that has rows and columns and values in the cells of those rows and columns.

To read in a csv file, we can use the `read_csv()` function that comes in one of the packages in tidyverse. When you read in a datafile, you need to save it under some label. We'll save our data file as dat.

```
dat <- read_csv(file = 'Lab2Data.csv')
```

```
## Rows: 205 Columns: 10
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): fave_animal
## dbl (9): id, e, n, c, a, o, pa, na, ls
##
```

```
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
  # Name of data file needs to be within quotations
```

Note: never name your data "data". This is because `data()` is a function is R and if you name any of your variables or objects the same name as one of its functions, it won't know which one you're talking about.

Note 2: `read_csv()` helpfully tells you the data type it assigned each of the columns you just read in. col_double means it's numeric, while col_character means... you guessed it, it's character! Can you guess what it says when it identifies one of the columns as being type logical?

Let's look at our data:

```
head(dat)
```

```
## # A tibble: 6 x 10
##      id     e     n     c     a     o    pa    na    ls fave_animal
##   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <chr>
## 1     1  2.05  4.75  3.2   5     4.65    17    16   1.4 cat
## 2     2  1.3   5     2.75  3.2   1.8     17    25   1.4 dog
## 3     3  2.95  4.35  3.8   3.9   3.15    19    12   3   other
## 4     4  1.5   4.85  3.35  2.85  4.6     10    25   1   cat
## 5     5  1.85  4.1   3.2   4.2   3.25    17    18   1   dog
## 6     6  1.4   3.9   4.45  4.45  2.85    24    32   1   other
```

```
# prints out 5 first rows, to see whole dataset use View(dat)
# or click the object in the Environment pane on the right.
```

- Columns labels key
  - e = extraversion
  - n = neuroticism
  - c = conscientiousness
  - a = agreeableness
  - o = openness
  - pa = positive affect
  - na = negative affect
  - ls = life satisfaction

We can look up some features of the dataset we just read in:

```
class(dat) # What kind of data structure is dat?
```

```
## [1] "spec_tbl_df" "tbl_df"      "tbl"         "data.frame"
```

```
dim(dat) # What are the dimensions of dat?
```

```
## [1] 205  10
```

```
str(dat)
```

```
## spc_tbl_ [205 x 10] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ id         : num [1:205] 1 2 3 4 5 6 7 8 9 10 ...
##  $ e          : num [1:205] 2.05 1.3 2.95 1.5 1.85 1.4 2.4 2.1 2.2 2.3 ...
##  $ n          : num [1:205] 4.75 5 4.35 4.85 4.1 3.9 3.8 3.2 3.05 3.55 ...
##  $ c          : num [1:205] 3.2 2.75 3.8 3.35 3.2 4.45 2.3 3.35 4.1 1.85 ...
##  $ a          : num [1:205] 5 3.2 3.9 2.85 4.2 4.45 2.65 4.4 3.45 3.9 ...
##  $ o          : num [1:205] 4.65 1.8 3.15 4.6 3.25 2.85 2.15 3.7 2.9 4.4 ...
##  $ pa         : num [1:205] 17 17 19 10 17 24 14 12 26 25 ...
##  $ na         : num [1:205] 16 25 12 25 18 32 30 16 12 12 ...
```

```
##  $ ls         : num [1:205] 1.4 1.4 3 1 1 1 2 1 6 2.4 ...
##  $ fave_animal: chr [1:205] "cat" "dog" "other" "cat" ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   id = col_double(),
##   ..   e = col_double(),
##   ..   n = col_double(),
##   ..   c = col_double(),
##   ..   a = col_double(),
##   ..   o = col_double(),
##   ..   pa = col_double(),
##   ..   na = col_double(),
##   ..   ls = col_double(),
##   ..   fave_animal = col_character()
##   .. )
##  - attr(*, "problems")=<externalptr>
```

```
# This is a new function.
# By looking at the output what do you think the str() function tells you?
```

```
summary(dat)
```

```
##        id            e               n               c               a
##  Min.   :  1   Min.   :1.150   Min.   :1.000   Min.   :1.650   Min.   :1.700
##  1st Qu.: 52   1st Qu.:2.900   1st Qu.:2.100   1st Qu.:3.100   1st Qu.:3.350
##  Median :103   Median :3.300   Median :2.650   Median :3.500   Median :3.800
##  Mean   :103   Mean   :3.266   Mean   :2.662   Mean   :3.513   Mean   :3.741
##  3rd Qu.:154   3rd Qu.:3.750   3rd Qu.:3.150   3rd Qu.:3.900   3rd Qu.:4.150
##  Max.   :205   Max.   :4.800   Max.   :5.000   Max.   :4.900   Max.   :5.000
##                NA's   :5                                       NA's   :8
##        o               pa              na              ls
##  Min.   :1.800   Min.   :10.0   Min.   :10.00   Min.   :1.000
##  1st Qu.:3.300   1st Qu.:24.0   1st Qu.:10.00   1st Qu.:3.200
##  Median :3.700   Median :30.0   Median :12.00   Median :4.800
##  Mean   :3.731   Mean   :30.6   Mean   :15.05   Mean   :4.439
##  3rd Qu.:4.150   3rd Qu.:37.0   3rd Qu.:18.00   3rd Qu.:5.800
##  Max.   :5.000   Max.   :50.0   Max.   :46.00   Max.   :7.000
##                                                 NA's   :2
##  fave_animal
##  Length:205
##  Class :character
##  Mode  :character
##
##
##
##
```

```
# The summary() function will give you the breakdown of a bunch of stats
# for each column.
```

How many missing values are there in the extraversion variable?

```
sum(is.na(dat$e))
```

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

If you look at the last answer we got, it's a little confusing to read, right? You have to start with the innermost

thing, and think about what that is (a vector of numbers), and then imagine what the next function that's around that will do to it (test to see if it's NA or not and return a vector of TRUE/FALSES) and then what the other thing around it does (sums all the TRUE values). With just three steps, it's not too terrible, but you can imagine it can get a lot worse if you need to use a lot of functions, one on top of the other. It's just not natural for most people to read from the inside out. Thankfully, there's a better way!

Pipe