4. IMPORT & RECODE DATA

Often the first obstacle to data analysis is just getting the data into your software application. Data can come to us in a dizzying variety of formats including .csv, .txt, .json, .xls, .Rdata, and many others. R can handle most data formats, but uses different commands for different file types. In the beginning, students mostly work with .csv data and use the read_csv command from the tidyverse package. Note that this is a different command than read.csv, a base R command.

Before importing data, let's clear our environment of all the dataframes and values we have saved in cache memory. This is good practice whenever importing or transitioning to new data sets within an R project, as values and variables may overlap between data sets.

Clear Environment & Import data

```
# Clear all objects and data from Environment.
rm(list = ls())
df <- read_csv(file = "data/Friends_Cholesterol.csv")
```

The read_csv command above takes a file path as the first argument. The command uses this file path to find and reach into the file on your computer and then copy the values from the file into a dataframe called "df". Understand that we have only temporarily "opened" the file with R Studio to take a snapshot of the data and then close the file. Any changes we make to the dataframe called df will have no effect on the original data stored in Friends_Cholesterol. This is an important feature that protects the integrity of raw data, by preventing the analysis from changing the raw data.

Absolute & Relative File Paths

At this point, you might be wondering, how exactly did R find the Friends_Cholesterol.csv file from a short and incomplete file path? Normally, applications need a file path that starts at the root directory, the very top level of everything on your hard drive, and proceeds through all nested directories (folders) to the target file. This long and complete file path is called an **absolute file path** and is unique to every file on every computer. On my computer, the absolute file path to the Friends_Cholesterol.csv file is...

/Users/stevenhobbs/Documents/3_Current_Classes/3.5800_SP22/2.R_Projects /Data_Essentials_With_R-update-Jul-31-22/data/Friends_Cholesterol.csv

How did we avoid typing this lengthy file path? R Projects! An R Project is essentially a special folder created by R Studio. When R projects are first created (File Menu > New R Project...) R also creates a special file with a .Rproj extension. The .Rproj file sets the working directory to the R Project folder. The working directory becomes the new starting point for file paths used by R and gives the entire R project the ability to use **relative file paths**, like the one used above to import the Friends_Cholesterol.csv data.

An great benefit to using R Project is that the entire project folder (and all contents) can be moved anywhere on your computer, or even transferred to a new computer, and the relative file paths will be unchanged. Without R Projects, the file path would be wrong and the import command would fail every time the .Rmd file is moved locally, or shared with a collaborator. For this reason, I recommend that you *ALWAYS* use relative file paths inside of R Projects to import data.

Important tip: Do not confuse the R Project folder with the R Markdown file. Click on the **Files** button in the bottom right R Studio panel and then click on the blue box with an R at the far right. You should now see a list of all the files inside the R Project folder. You should see the Data folder as well as several.Rproj and .Rmd files and any files you've created from knitting. The .Rproj file does the magic that allows us to use relative file paths. The .Rmd files are where we write Markdown text and R code to create a document such as this one. If you don't see the blue box with an R, you mostly likely opened this .Rmd file from *outside* the .Rproj. The solution is to close the .Rmd file and then open the .Rproj file before re-opening the .Rmd file.

Recoding Data

Raw data is often coded in ways that are problematic for analysis, difficult for human interpretation, or fraught with other problems.

For example, the Friends_Cholesterol data has a variable called gender that has 3 problems:

1. The variable should be called sex (there is a difference).
2. The variable has values that are 0 or 1, but should be "male" and "female".
3. The gender/sex variable is numeric, but should be a factor (categorical) variable.

Similar problems exist for the group variable, which should be a factor variable coded as "treat" or "statin".

Correcting the variable type is crucial, because variable type determines the way commands operate on the data. We can fix these issues (and many others) during the initial import using tidyverse commands **"piped"** into a chain as shown below. Most of the commands used for this process come from the dplyr and forcats packages that are loaded with tidyverse.

dplyr contains tidyverse commands that are designed to work on dataframes.

forcats contains tidyverse commands that are designed to work on factor variables, and which all begin with the prefix "fct".

Recode with import pipe chains Run the code chunk below and then read the explanatory text that follows.

```
df <-
    read_csv(file = "data/Friends_Cholesterol.csv",
             col_types = "cnfnfnnnnnn") %>%
    rename(sex = gender) %>%
    mutate(sex = fct_recode(sex,
                            'male' = '0',
                            'female' = '1'),
           group = fct_recode(group,
                              'control' = '0',
                              'statin' = '1'))
```

Let's dissect the commands and arguments used in this pipe chain:

coltypes =

This optional argument to the read_csv() command provides a string of characters that identify the desired type of every variable in the imported data. This argument overrides the often incorrect default variable types that R assumes based on the dataframe values. The length of this string must match the number of variables exactly. For most imports we need only three letters to identify our variable types: c = character, n = numeric, f = factor.

rename()

This command takes a dataframe as its first argument (piped in above from the output of read_csv) and then uses the format = to rename a column.

mutate()

Mutate takes a dataframe (piped in above from the output of rename) and creates new variables from existing variables, or saves over an existing variable after mutating it in some useful way. Here we use the fct_recode command inside the mutate command to re-label levels of the sex variable.

Piping chains like the one used above rely upon an important design feature. Each use of the pipe operator pulls a dataframe from the left and inserts it into the command to the right as the very first argument. This means the command that comes before the pipe operator must output a dataframe, and the command to the right of the pipe operator must take in a dataframe as the first argument. Most tidyverse commands are designed with piping in mind, meaning they expect the first argument to be a dataframe and they also output a dataframe. However, commands like fct_recode() that take in a vector (a single column) as their first argument can not be used as a direct link in the piping chain. They should instead be used within mutate as shown in this example.

Recode without import pipe chains While the above piping chain can be broken out into individual commands as shown below, this approach is more typing, worse readability, and greater potential for warnings and errors. For example, after running the entire code chunk below, run just the fct_recode(df$sex...) command again and you should see a warning message in the console window.

```
# Read in data and save over the existing df dataframe
df <- read_csv(file = "data/Friends_Cholesterol.csv")

# Change the name of column 3 to 'sex'
names(df)[3] <- 'sex'

# Change sex to factor and rename levels
df$sex <- factor(df$sex)
df$sex <- fct_recode(df$sex,
                     'male' = '0',
                     'female'='1')

# Change group to factor and rename levels using pipe operator %>%.
df$group <-
  df$group %>%
  factor() %>%
  fct_recode('control' = '0',
             'statin' = '1')
```

Important tip: Regardless of the commands and approach used, I recommend recoding data and solving other data wrangling problems in the same code chunk as the data import command, and *before* using the dataframe to run analyses, create graphs, etc. If additional problems with the data are discovered later in the analysis, return to the data import code chunk to fix the problems and re-run the code chunk before resuming the analysis.