Data manipulation
==========

Introduction:

Data needs to be examined and any problems fixed before analysis can be done.In statistics, we focus on four areas.

1. Data accuracy
    - make sure data types are correct
    - check for typos
    - check for nonsensical data
    - check categories make sense
    - correct problems if possible or delete
    - reverse code items if needed
    - score instruments if needed
    - keep track of what you do so you can be transparent
2. Missing data
3. outliers
4. Statistical assumptions

This assignment shows some more steps to take to get data ready for analysis.

Directions:

Complete each step of the assignment below.

Package management in R

```r
# keep a list of the packages used in this script
packages <- c("tidyverse","rio","jmv")
```

This next code block has eval=FALSE because you don’t want to run it when knitting the file. Installing packages when knitting an R notebook can be problematic.

```r
# check each of the packages in the list and install them if they're not installed already
for (i in packages){
  if(! i %in% installed.packages()){
    install.packages(i,dependencies = TRUE)
  }
  # show each package that is checked
  print(i)
}
```

```r
# load each package into memory so it can be used in the script
```

```
for (i in packages){
  library(i,character.only=TRUE)
  # show each package that is loaded
  print(i)
}
```

```
    ## -- Attaching packages --------------------------------------- tidyverse
1.3.0 --

    ## v ggplot2 3.3.3     v purrr   0.3.4
    ## v tibble  3.0.4     v dplyr   1.0.2
    ## v tidyr   1.1.2     v stringr 1.4.0
    ## v readr   1.4.0     v forcats 0.5.0

    ## -- Conflicts ------------------------------------------
tidyverse_conflicts() --
    ## x dplyr::filter() masks stats::filter()
    ## x dplyr::lag()    masks stats::lag()

    ## [1] "tidyverse"
    ## [1] "rio"
    ## [1] "jmv"
```

A note about working with packages and functions in R: When installing packages you may see notifications that SomePackage::function() masks OtherPackage::function(). What that means is that two different packages used the same function name. The functions may or may not do the same thing, but for whatever reason the pogrammers of each package happened to choose the same name for a function in their respective package. The last package installed will be the version of that function that will be used if only the function name is used in your R code. The SomePackage::function() notation with the colons in it means to use the function named function() from the package named SomePackage. I think I have decided when I write R code that I will always try to stick to that notation. That way it's very clear which package a desired function comes from and there will be no problems with masking if different packages used the same function name.

Open data file

The rio package works for importing several different types of data files. We're going to use it in this class. There are other packages which can be used to open datasets in R. You can see several options by clicking on the Import Dataset menu under the Environment tab in RStudio. (For a csv file like we have this week we'd use either From Text(base) or From Text (readr). Try it out to see the menu dialog.)

``` r
# import the Week3.rds dataset into RStudio

# Using the file.choose() command allows you to select a file to import from
another folder.
# dataset <- rio::import(file.choose())
```

```
# This command will allow us to import the rds file included in our project
folder.
dataset <- rio::import("Week3.rds")
```

Examine the dataset

Examine the dataset and answer the following questions. Enter any code you use in the following code block.

- How many variables are in the dataset?
    - Answer:11
- List each variable and the level of measurement. Note if there are any issues you see with the data in that variable. (The first one is done for you.)
    - sex - nominal, no issues noted
    - research- nominal, no issues noted
    - height- ratio, no issues noted
    - weight- ratio, no issues noted
    - lvst1- ordinal, no issues noted
    - lvst2- ordinal, no issues noted
    - lvst3- ordinal, no issues noted
    - lvst4- ordinal, no issues noted
    - frst1- ordinal, no issues noted
    - frst2- ordinal, no issues noted
    - frst3- ordinal, no issues noted

Calculate a new column

The dataset includes variables for height and weight. Calculate the BMI for each participant by executing the following code block. The formula for BMI is weight (kg)/height (m)^2. Round the result to one decimal place.

- What level of measurement is the new bmi variable?
    - Answer: ratio

```r
dataset <- dataset %>% mutate(bmi = round(weight/(height/100)^2,1))
```

Reverse code a column

The dataset contains questions for two fictitious instruments (questionnaires): 1) the Love of Statistics Scale, and 2) the Fear of Statistics Scale.

The Love of Statistics Scale consists of 4 questions rated on a scale of 1-5. Before the instrument can be scored, the last question needs to be

reverse coded. Execute the following code block to reverse code the last question in the Love of Statistics Scale.

- What level of measurement is the new lvst4reverse variable?
    - Answer: ordinal

```r
# The line below is how I first reverse coded the variable. It seemed to work, but I later found some problems with the z-scores.
# While the text values in the variable did get changed, the order of the factor levels also got reversed numerically.
# The result was that the number values really didn't change. I left this here to make you aware to check results are what you expect when you modify the variables.
# It's easy to miss a problem like this in your code.
#dataset <- dataset %>% mutate(lvst4reverse = recode(lvst4, "completely"="Not at all", "mostly"="a little", "moderately"="moderately", "a little"="mostly", "Not at all"="completely"))

# The line below correctly reverse coded the lvst4 variable.
dataset <- dataset %>% mutate(lvst4reverse = 6 - as.numeric(lvst4))
```

Score the Love of Statistics Scale instrument

Run the following code block to calculate a final score for the Love of Statistics Scale for each participant.

- What level of measurement is the new LOSS_total
    - Answer: ordinal

```r
dataset <- dataset %>% mutate(LOSS_total = as.numeric(lvst1) + as.numeric(lvst2) + as.numeric(lvst3) + as.numeric(lvst4reverse))
```

Score the Fear of Statistics Scale instrument

Enter code in the following code block to calcuate a final score for the Fear of Statistics Scale for each participant. Call the new variable FOSS_total.

```r
# Enter code here below the comment.
dataset <- dataset %>% mutate(FOSS_total = as.numeric(frst1) + as.numeric(frst2) + as.numeric(frst3))
```

Examine the new LOSS_total and FOSS_total variables

Note any issues with the data in the new variables. Place code you use

to examine the variables in the code block below. (Or just describe what you did.) + no issues noted

Final scores using averages

Not all instruments are scored using sums. If all items are measured using the same scale a mean can be calculated. One advantage of using means is the final score is in the same scale as the original items so interpreting can be a bit more straightforward, It's also possible to calculate the mean when there are missing data without changing the scale of the final score. Run the following code block to score the instruments using means instead of sums.

``` r
dataset <- dataset %>% mutate(LOSS_mean = (as.numeric(lvst1) + as.numeric(lvst2) + as.numeric(lvst3) + as.numeric(lvst4reverse))/4)
dataset <- dataset %>% mutate(FOSS_mean = (as.numeric(frst1) + as.numeric(frst2) + as.numeric(frst3))/3)
```

z-scores

Z-scores are useful for a number of reasons. Since z-scores are unit free (the units are standard deviations) they can be used to compare scores between instruments or tests which have different units. (Like our LOSS scale which is in love units and our FOSS scale which is in fear units.) Also, z-scores let us see how many standard deviations a score is away from the mean. It is an easy way to see which scores are potenential outliers. Z-scores above 2 are more than 2 standard deviations away from the mean and are potential univariate outliers.Run the following code block to calculate z-scores for the LOSS and FOSS variables. Examine the new variables.

- How do the z-scores compare between the sum scores and the mean scores?
    - Answer: They are the same
- Are there any z-scores greater than 2 in the LOSS and FOSS variables? How many?
    - LOSS z-score \> 2:yes for positive 2, line 63
    - FOSS z-score \> 2:yes for positive 2, lines 74 & 18
- What does it mean if a z-score is positive or negative?
    - positive: above the mean
    - negative: below the mean

``` r
dataset <- dataset %>% mutate(LOSS_total_z = scale(LOSS_total))
dataset <- dataset %>% mutate(LOSS_mean_z = scale(LOSS_mean))

dataset <- dataset %>% mutate(FOSS_total_z = scale(FOSS_total))
```

```
dataset <- dataset %>% mutate(FOSS_mean_z = scale(FOSS_mean))
```

Save your data for next week

``` r
rio::export(dataset,"Week4.rds")
```

Save your output.

Save this assignment as a markdown file.

Submit your assignment.

Submit the output you just saved for your assignment.