L06 Data Import

STAT 301-1 Data Science I

Shay Lebovitz

October 23rd, 2019

Contents

Overview

The goal of this lab is to learn the how to utilize the `readr` package, a member of the `tidyverse`, to load flat files (e.g., `csv` & `txt`) — `readr` package home page

Datasets

All datasets are either coded inline or contained in the `data` sub-directory within the downloaded zip file.

Exercises

Please complete the following exercises. Be sure your solutions are clearly indicated and that the document is neatly formatted.

```
library(tidyverse)
```

Load Packages

Exercise 1

Demonstrate how to read in `TopBabyNamesByState.txt` contained in the `data` sub-directory using the appropriate function from `readr` package. After reading in the data, determine the top male and female names in 1984 for South Dakota.

```
top_baby_names <- read_delim('data/TopBabyNamesbyState.txt', '|')
top_baby_names %>%
  filter(state == 'SD', year == 1984) %>%
  arrange(desc(occurences))
```

```
## # A tibble: 2 x 5
##   state gender  year top_name occurences
##   <chr> <chr>  <dbl> <chr>         <dbl>
## 1 SD    M       1984 Matthew         220
## 2 SD    F       1984 Jessica         166
```

Exercise 2

What is the main difference between `read_csv()`, `read_csv2()`, and `read_tsv()`? They all accept the parameters `na`, `col_names`, and `trim_ws`. Please describe in your own words what each of these parameters control.
`read_csv()` reads comma delimited files, `read_csv2()` reads semicolon delimited files, and `read_tsv()` reads tab delimited files. Setting `col_names` to `FALSE` tells R to not treat the first row as the column names of the table. `na` tells R which values to assign as `NA`. `trim_ws` removes leading and trailing whitespace from strings.

Exercise 3

Read in the fixed width file `fwf_example.txt` contained in the sub-directory `data`. We have provided the column names so the final dataset has appropriate names. You may want to look at the dataset in a text editor to get an idea of what you are dealing with. (Hint: Use `fwf_empty()` for `col_positions` and the `skip` parameter where appropriate.)

```
# Column names.
column_names <- c("Entry", "Per.", "Post Date", "GL Account", "Description", "Srce.", "Cflow", "Ref.",

fwf <- read_fwf(file = 'data/fwf_example.txt',
                col_positions = fwf_empty('data/fwf_example.txt',
                                          skip = 2, col_names = column_names), skip = 2)
```

Exercise 4 (Website: 11.2.2 Ex. 5)

Identify what is wrong with each of the following inline CSV files. Describe what happens when you run the code.

```
read_csv("a,b\n1,2,3\n4,5,6") #1
read_csv("a,b,c\n1,2\n1,2,3,4") #2
read_csv("a,b\n\"1") #3
read_csv("a,b\n1,2\na,b") #4
read_csv("a;b\n1;3") #5
```

1) Not a complete matrix: 2 values in first row, 3 in second and third. 2) Not a complete matrix: 3 values in first row, 2 in second row, and 3 in third. Fills in `NA` for missing value. 3) Not a complete matrix: 2 values in first row, one in second row. 4) Nothing seems wrong with this one, it is a little strange that you would have integers and characters in the same column of data though. 5) Should use `read_csv2()` for this semicolon delimited inline.

Exercise 5 (Website: 11.3.5 Ex. 6)

What are the most common encodings used in Europe? What are the most common encodings used in Asia? Do some googling to find out. **Latin1 is used predominantly for western European languages, and**

Latin2 is used predominantly for eastern European languages. Big5 and GB18030 are used for Chinese languages.

Exercise 6 (Website: 11.3.5 Ex. 7)

Generate the correct format string to parse each of the following dates and times:

```
d1 <- "January 1, 2010"
d2 <- "2015-Mar-07"
d3 <- "06-Jun-2017"
d4 <- c("August 19 (2015)", "July 1 (2015)")
d5 <- "12/30/14" # Dec 30, 2014
t1 <- "1705"
t2 <- "11:15:10.12 PM"

parse_date(d1, '%B %d, %Y')
```

```
## [1] "2010-01-01"
```

```
parse_date(d2, '%Y-%b-%d')
```

```
## [1] "2015-03-07"
```

```
parse_date(d3, '%d-%b-%Y')
```

```
## [1] "2017-06-06"
```

```
parse_date(d4, '%B %d (%Y)')
```

```
## [1] "2015-08-19" "2015-07-01"
```

```
parse_date(d5, "%m/%d/%y")
```

```
## [1] "2014-12-30"
```

```
parse_time(t1, '%H%M')
```

```
## 17:05:00
```

```
parse_time(t2, '%I:%M:%OS %p')
```

```
## 23:15:10.12
```

Exercise 7

Read in and store `exercise7.csv` from the `data` sub-directory. Use `problems()` to diagnose and fix issues when parsing the file. Once you have corrected all parsing issues and typed each variable (`x` and `y`) correctly, display the last 6 observations/records by using `tail()`. (Hint: There are 2 ways to ensure that the columns (variables `x` and `y`) are correctly typed. You should try to do both ways — `col_types()` and `guess_max`.)

```
exercise <- read_csv('data/exercise7.csv',  col_types = cols(
  x = col_double(),
  y = col_date()))
tail(exercise)
```

```
## # A tibble: 6 x 2
##       x y
##   <dbl> <date>
## 1 0.805 2019-11-21
```

```
## 2 0.164 2018-03-29
## 3 0.472 2014-08-04
## 4 0.718 2015-08-16
## 5 0.270 2020-02-04
## 6 0.608 2019-01-06
```

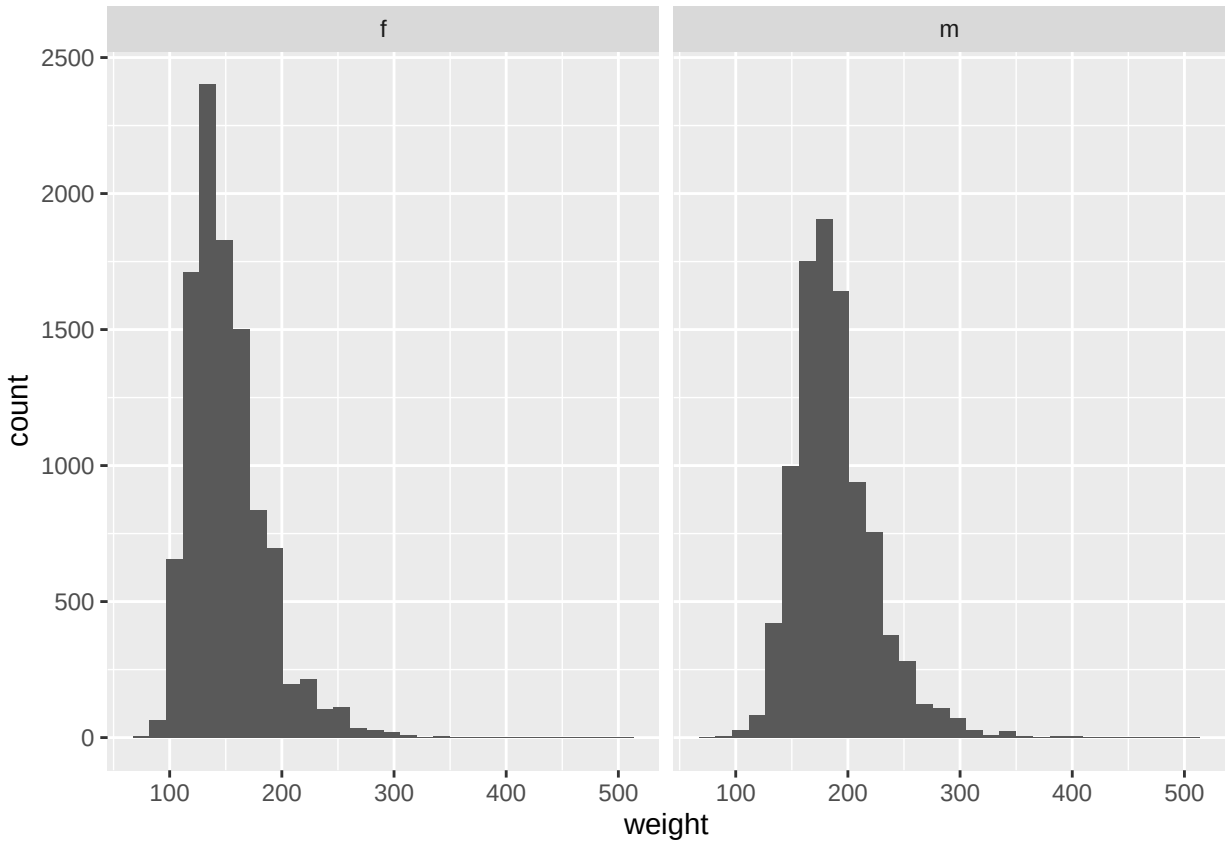
Exercise 8

Read in and store `cdc.txt` from the `data` sub-directory.

- Create a graphic that compares the distribution of weights for each gender.
- Write out a copy of the dataset to the `data` sub-directory as a tab deliminated text file named `cdc_tab.txt`.
- Write out a copy of the dataset to the `data` sub-directory as an RDS file named `cdc.rds`. *List some benefits of writing a dataset to an RDS file type.

```
cdc <- read_delim('data/cdc.txt', '|')

cdc %>%
  ggplot(mapping = aes(x = weight)) +
  geom_histogram(bins = 30) +
  facet_wrap(~gender)
```




```
write_tsv(cdc, 'data/cdc_tab.txt')
write_rds(cdc, 'data/cds.rds')
```

Saving as a .rds file doesn't lose the type information, as you do when writing to a .csv. RDS also supports list-columns.

Exercise 9

What types of files do the packages `haven` and `readxl` deal with? **`haven` reads STATA, SAS and SPSS files. `readxl` reads Excel files.**

Exercise 10

Guess the encoding of `cc-est2016-alldata.csv` which is contained in the `data` sub-directory. What happens when reading `cc-est2016-alldata.csv` into your R session? How could you increase the read in speed of this dataset for R sessions in the future?

```
guess_encoding('data/cc-est2016-alldata.csv')
alldata <- read_csv('data/cc-est2016-alldata.csv')
```

It's a huge file, so it takes a while to read the entire thing. To increase the speed, you could use `data.table::fread()`

Challenge

Required for graduate students, but not for undergraduate students.

Extract 2016 total population estimates for each state and DC from `cc-est2016-alldata.csv` contained in the `data` sub-directory. Force RStudio to display all rows and have them arranged from largest to smallest in population. (Hint: May want to read the `cc-est2016-alldata.pdf` for details).