CSSS508, Lecture 5

Importing, Exporting, and Cleaning Data

Michael Pearce

(based on slides from Chuck Lanfear)

October 27, 2022

UW CS&SS

Topics

Last time, we learned about,

1. Types of Data
2. Vectors
3. Matrices
4. Lists
5. Data Frames and Tibbles

Today, we will cover,

1. Importing and exporting data
2. Cleaning and reshaping data
3. Dates and times

1. Importing and Exporting Data

Data Packages

R has a *big* user base. If you are working with a popular data source, it will often have a devoted R package on *CRAN* or *Github*.

- `WDI`: World Development Indicators (World Bank)
- `WHO`: World Health Organization API
- `tidycensus`: Census and American Community Survey [1]
- `quantmod`: financial data from Yahoo, FRED, Google

You can also import data into R in a few ways...

[1] We'll use this in our lecture on geographical data!

Delimited Text Files

Besides a package, it's easiest when data is stored in a text file.

An example of a comma-separated values (**.csv**) file is below:

```
"Subject","Depression","Sex","Week","HamD","Imipramine"
101,"Non-endogenous","Second",0,26,NA
101,"Non-endogenous","Second",1,22,NA
101,"Non-endogenous","Second",2,18,4.04305
101,"Non-endogenous","Second",3,7,3.93183
101,"Non-endogenous","Second",4,4,4.33073
101,"Non-endogenous","Second",5,3,4.36945
103,"Non-endogenous","First",0,33,NA
103,"Non-endogenous","First",1,24,NA
103,"Non-endogenous","First",2,15,2.77259
```

`readr`

R has some built-in functions for importing data, such as `read.table()` and `read.csv()`.

The `readr` package provides similar functions, like `read_csv()`, that have slightly better features:

- Faster!
- Better defaults (e.g. doesn't convert characters to factors)
- A *little* smarter about dates and times
- Loading bars for large files

```r
library(readr)
```

`readr` Importing Example

Let's import some data about song ranks on the Billboard Hot 100 in 2000:

```
billboard_2000_raw <- read_csv(file = "https://clanfear.github.io/CSSS508/Lectures/Week5/dat
```

```
## Rows: 317 Columns: 81
## ── Column specification ──────────────────────────────────────────────
## Delimiter: ","
## chr   (2): artist, track
## dbl  (66): year, wk1, wk2, wk3, wk4, wk5, wk6, wk7, wk8, wk9, wk10...
## lgl  (11): wk66, wk67, wk68, wk69, wk70, wk71, wk72, wk73, wk74, w...
## date  (1): date.entered
## time  (1): time
##
## 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.
```

Did It Load?

Look at the data types for the last few columns:

```r
str(billboard_2000_raw[, 65:ncol(billboard_2000_raw)])
```

```
## tibble [317 × 17] (S3: tbl_df/tbl/data.frame)
##  $ wk60: num [1:317] NA NA NA NA NA NA NA NA NA NA ...
##  $ wk61: num [1:317] NA NA NA NA NA NA NA NA NA NA ...
##  $ wk62: num [1:317] NA NA NA NA NA NA NA NA NA NA ...
##  $ wk63: num [1:317] NA NA NA NA NA NA NA NA NA NA ...
##  $ wk64: num [1:317] NA NA NA NA NA NA NA NA NA NA ...
##  $ wk65: num [1:317] NA NA NA NA NA NA NA NA NA NA ...
##  $ wk66: logi [1:317] NA NA NA NA NA NA ...
##  $ wk67: logi [1:317] NA NA NA NA NA NA ...
##  $ wk68: logi [1:317] NA NA NA NA NA NA ...
##  $ wk69: logi [1:317] NA NA NA NA NA NA ...
##  $ wk70: logi [1:317] NA NA NA NA NA NA ...
##  $ wk71: logi [1:317] NA NA NA NA NA NA ...
##  $ wk72: logi [1:317] NA NA NA NA NA NA ...
##  $ wk73: logi [1:317] NA NA NA NA NA NA ...
##  $ wk74: logi [1:317] NA NA NA NA NA NA ...
##  $ wk75: logi [1:317] NA NA NA NA NA NA ...
##  $ wk76: logi [1:317] NA NA NA NA NA NA ...
```

What Went Wrong?

Importing data can be tricky!

`readr` uses the first 1000 rows to guess the data type of each column (integer, logical, numeric, character).

In this case, there was enough missing data that R wasn't able to successfully choose the data type!

You can address this using the `guess_max` argument (default is 1000).

```
billboard_2000_raw <- read_csv(
  file = "https://clanfear.github.io/CSSS508/Lectures/Week5/data/bill
  guess_max = 5000) # Default is 1000
```

Alternate Solution

Import the data manually!

In the upper right-hand console, select:

`Import Dataset > From Text (readr)`

Once you've imported the data, you can `copy/paste` the import code from the console into your file!!

This makes the process *reproducible!*

Importing Other Data Types

- For Excel files (`.xls` or `.xlsx`), use package `readxl`
- For Google Docs Spreadsheets, use package `googlesheets4`
- For Stata, SPSS, and SAS files, use package `haven` (`tidyverse`)
- For Stata, SPSS, and Minitab, use package `foreign`

You **won't** keep text formatting, color, comments, or merged cells!!

Writing Delimited Files

Getting data out of R into a delimited file is very similar to getting it into R:

```
write_csv(billboard_2000_raw, path = "billboard_data.csv")
```

This saved the data we pulled off the web in a file called `billboard_data.csv` in my working directory.

2. Cleaning and Reshaping Data

Initial Spot Checks

First things to check after loading new data:

- Did all the rows/columns from the original file make it in?
  - Check using `dim()` or `str()`
- Are the column names in good shape?
  - Use `names()` to check; fix with `rename()`
- Are there "decorative" blank rows or columns to remove?
  - `filter()` or `select()` out those rows/columns
- How are missing values represented: `NA`, `" "` (blank), `.` (period), `999`?
  - Use `mutate()`
- Are there character data being incorrectly represented as numeric, or vice versa?
  - Use `mutate()` and `as.numeric()`

Tidy Data

Tidy data (aka "long data") are such that:

1. The values for a single observation are in their own row.
2. The values for a single variable are in their own column.
3. There is only one value per cell.

Why do we want tidy data?

- **Easier to understand** many rows than many columns
- Required for **plotting** in `ggplot2`
- Required for many types of **statistical procedures** (e.g. hierarchical or mixed effects models)
- Fewer issues with **missing values and "imbalanced"** repeated measures data

Slightly "Messy" Data

Program	First Year	Second Year
Evans School	10	6
Arts & Sciences	5	6
Public Health	2	3
Other	5	1

- What is an **observation**?
  - A group of students from a program of a given year
- What are the **variables**?
  - Program, Year
- What are the **values**?
  - Program: Evans School, Arts & Sciences, Public Health, Other
  - Year: First, Second -- **in column headings!**
  - Count: **spread over two columns!**

Tidy Version

Program	Year	Count
Evans School	First	10
Evans School	Second	6
Arts & Sciences	First	5
Arts & Sciences	Second	6
Public Health	First	2
Public Health	Second	3
Other	First	5
Other	Second	1

- Each variable is a column.
- Each observation is a row.
- Each cell has a single value.

Billboard is Just Ugly-Messy

year	artist	track	time	date.entered	wk1	wk2	wk3	wk4	wk5
2000	2 Pac	Baby Don't Cry (Keep...	04:22:00	2000-02-26	87	82	72	77	87
2000	2Ge+her	The Hardest Part Of ...	03:15:00	2000-09-02	91	87	92	NA	NA
2000	3 Doors Down	Kryptonite	03:53:00	2000-04-08	81	70	68	67	66
2000	3 Doors Down	Loser	04:24:00	2000-10-21	76	76	72	69	67
2000	504 Boyz	Wobble Wobble	03:35:00	2000-04-15	57	34	25	17	17
2000	98^0	Give Me Just One Nig...	03:24:00	2000-08-19	51	39	34	26	26
2000	A*Teens	Dancing Queen	03:44:00	2000-07-08	97	97	96	95	100
2000	Aaliyah	I Don't Wanna	04:15:00	2000-01-29	84	62	51	41	38
2000	Aaliyah	Try Again	04:03:00	2000-03-18	59	53	38	28	21
2000	Adams, Yolanda	Open My Heart	05:30:00	2000-08-26	76	76	74	69	68

Week columns continue up to `wk76`!

Billboard

- What are the **observations** in the data?
    - Week since entering the Billboard Hot 100 per song
- What are the **variables** in the data?
    - Year, artist, track, song length, date entered Hot 100, week since first entered Hot 100 (**spread over many columns**), rank during week (**spread over many columns**)
- What are the **values** in the data?
    - e.g. 2000; 3 Doors Down; Kryptonite; 3 minutes 53 seconds; April 8, 2000; Week 3 (**stuck in column headings**); rank 68 (**spread over many columns**)

`tidyr`

The `tidyr` package provides functions to tidy up data.

Key functions:

- **`pivot_longer()`**: takes a set of columns and pivots them down to make two new columns (which you can name yourself):
  - A `name` column that stores the original column names
  - A `value` with the values in those original columns

- **`pivot_wider()`**: inverts `pivot_longer()` by taking two columns and pivoting them up into multiple columns

We're going to focus only on `pivot_longer` here, but know that it can be reversed!

`pivot_longer()`

This function usually takes three arguments:

1. **cols**: The columns we want to modify
2. **names_to**: New variable name to store original columns
3. **values_to**: New variable name to store original values

Example of `pivot_longer()`

```r
library(tidyr)
library(dplyr)
billboard_2000 <- billboard_2000_raw %>%
  pivot_longer(cols=wk1:wk76,
               names_to ="week",
               values_to = "rank")
billboard_2000 %>% head(5)
```

```
## # A tibble: 5 × 7
##    year artist track                   time   date.entered week   rank
##   <dbl> <chr>  <chr>                   <time> <date>       <chr> <dbl>
## 1  2000 2 Pac  Baby Don't Cry (Keep... 04:22  2000-02-26   wk1      87
## 2  2000 2 Pac  Baby Don't Cry (Keep... 04:22  2000-02-26   wk2      82
## 3  2000 2 Pac  Baby Don't Cry (Keep... 04:22  2000-02-26   wk3      72
## 4  2000 2 Pac  Baby Don't Cry (Keep... 04:22  2000-02-26   wk4      77
## 5  2000 2 Pac  Baby Don't Cry (Keep... 04:22  2000-02-26   wk5      87
```

Now we have a single week column!

Lots of missing values?!

```
summary(billboard_2000$rank)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##    1.00   26.00   51.00   51.05   76.00  100.00   18785
```

We don't want to keep the 18785 rows with missing ranks.

Pivoting Better: `values_drop_na`

Adding the argument `values_drop_na = TRUE` to `pivot_longer()` will remove rows with missing ranks.

```r
billboard_2000 <- billboard_2000_raw %>%
  pivot_longer(cols=wk1:wk76,
               names_to ="week",
               values_to = "rank",
               values_drop_na = TRUE)
summary(billboard_2000$rank)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1.00   26.00   51.00   51.05   76.00  100.00
```

No more `NA` values!

```r
dim(billboard_2000)
```

```
## [1] 5307    7
```

And way fewer rows!

parse_number()

The week column is character, but should be numeric.

```
head(billboard_2000$week)
```

```
## [1] "wk1" "wk2" "wk3" "wk4" "wk5" "wk6"
```

`parse_number()` grabs just the numeric information from a character string:

```
billboard_2000 <- billboard_2000 %>%
    mutate(week = parse_number(week))
summary(billboard_2000$week)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
##    1.00    5.00   10.00   11.47   16.00   65.00 
```

More sophisticated tools for character strings will be covered later in this course!

3. Dates and Times

`lubridate`

The package `lubridate` (part of the tidyverse!) has a *very large* number of functions you can use!

- Converting dates/times between formats (DD-MM-YY to YY-MM-DD)
- Extracting dates/times (day of week, month, leap years, etc.)
- Math with dates/times (time zone conversions, etc.)

There's too much to cover all of it, but I'll run through a few examples.

Dates in `billboard_2000`

```r
billboard_2000 %>% select(date.entered) %>% head(10)
```

```
## # A tibble: 10 × 1
##    date.entered
##    <date>
##  1 2000-02-26
##  2 2000-02-26
##  3 2000-02-26
##  4 2000-02-26
##  5 2000-02-26
##  6 2000-02-26
##  7 2000-02-26
##  8 2000-09-02
##  9 2000-09-02
## 10 2000-09-02
```

Extracting Year, Month, or Day

```r
library(lubridate)
head(billboard_2000$date.entered,5)
```

```
## [1] "2000-02-26" "2000-02-26" "2000-02-26" "2000-02-26" "2000-02-26"
```

```r
year(billboard_2000$date.entered) %>% head(5)
```

```
## [1] 2000 2000 2000 2000 2000
```

```r
month(billboard_2000$date.entered) %>% head(5)
```

```
## [1] 2 2 2 2 2
```

```r
day(billboard_2000$date.entered) %>% head(5)
```

```
## [1] 26 26 26 26 26
```

Extracting Weekday, Quarter, and Leap Year Boolean

```r
wday(billboard_2000$date.entered) %>% head(5)
```

```
## [1] 7 7 7 7 7
```

```r
quarter(billboard_2000$date.entered) %>% head(5)
```

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

```r
leap_year(billboard_2000$date.entered) %>% head(5)
```

```
## [1] TRUE TRUE TRUE TRUE TRUE
```

Activity!

In groups of 2-3, you will use the Billboard data to investigate a question:

1. Write down a question of interest that could be studied with this data
   - *Which/how many artists had #1 hits?*
   - *How does rank for each song change over time?*
   - *Is there a relationship between highest rank and length of song?*
2. Make the Billboard data *tidy*, perhaps using the code from this lecture.
3. Perform additional steps (if necessary) to help answer your question:
   - Perhaps using `filter`, `select`, `group_by`, `mutate`, `summarize`, etc.
4. Make a plot or table that answers your question and write down your answer in a sentence.
5. Email me your question, plot/table, and written answer (`mpp790@uw.edu`)

My Example: Question

Question: Do songs that hit #1 have a different trajectory than those that don't?

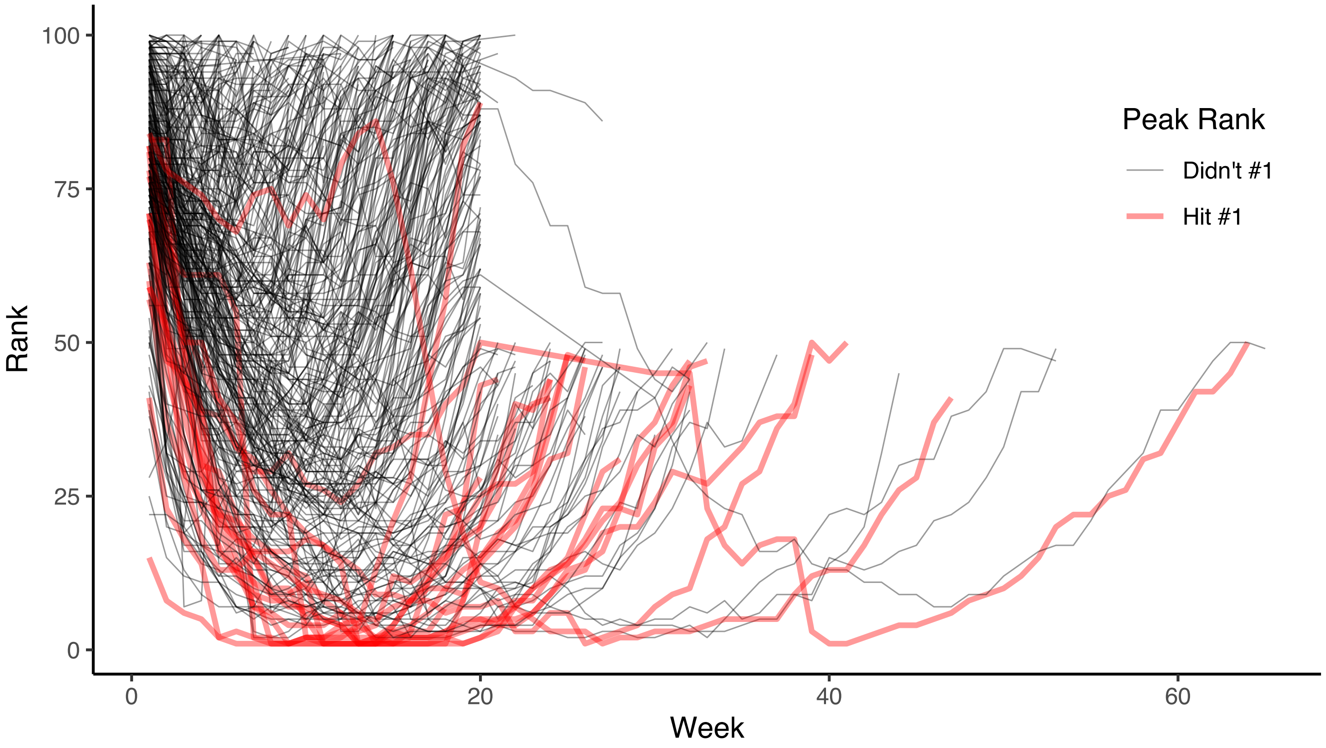
```
billboard_2000_question <- billboard_2000 %>%
    group_by(artist, track) %>%
    mutate(`Weeks at #1` = sum(rank == 1),
           `Peak Rank`   = ifelse(any(rank == 1),
                                  "Hit #1",
                                  "Didn't #1"))
```

Note: `any(min_rank==1)` checks to see if *any* value of `rank` is equal to one for the given `artist` and `track`

My Example: Figure

```r
library(ggplot2)
billboard_trajectories <-
  ggplot(data = billboard_2000_question,
         aes(x = week, y = rank, group = track,
             color = `Peak Rank`)) +
  geom_line(aes(size = `Peak Rank`), alpha = 0.4) +
  theme_classic() +
  xlab("Week") + ylab("Rank") +
  scale_color_manual(values = c("black", "red")) +
  scale_size_manual(values = c(0.25, 1)) +
  theme(legend.position = c(0.90, 0.75),
        legend.background = element_rect(fill="transparent"))
```

Charts of 2000: Beauty!



Songs that reach #1 on the Billboard charts appear to last >20 weeks on the charts, while other songs very rarely make it past that point.

Homework

Coming soon!