Cleaning Up Your Messy Data in R with the `janitor` Package

Scott Nestler

Michiana RUG

2020-11-20

The `janitor` Package

Motivation:

- "Data scientists, according to interviews and expert estimates, spend from 50 percent to 80 percent of their time mired in this more mundane labor of collecting and preparing unruly digital data, before it can be explored for useful nuggets." (NYT, 2014)

3 Main Purposes:

- Format ugly data.frame column names
- Isolate partially duplicate records
- Create and format quick tabulations with 1 to 3 variables

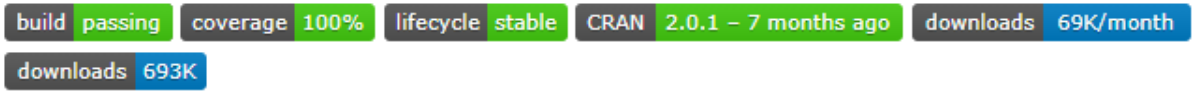
NOTES:

1. janitor is a `tidyverse`-oriented package that plays nicely with the `%>%` pipe.
2. Works best with data brought in with `readr` and `readxl` packages.

Two Ways to Install `janitor`

- Official, released version from CRAN
  - `install.packages("janitor")`
- Latest development version from GitHub
  - `install.packages("devtools")`
  - `devtools::install_github("sfirke/janitor")`

janitor



Function 1: Cleaning Dirty Data

So let's look at some dirty data. I've got some student data in an Excel file.

```
library(janitor)
```

```
## Warning: package 'janitor' was built under R version 4.0.3
```

```
##
## Attaching package: 'janitor'
## The following objects are masked from 'package:stats':
##
##     chisq.test, fisher.test
```

```
library(readxl)
dirty <- read_excel("MessyData.xlsx")
colnames(dirty)
```

```
## [1] "ID"          "First Name"  "SEX"         "age"         "Semester"
## [6] "Section"     "OVERALL %"   "Grade"       "Fav Subject"
```

So what problems do you see here?

-
-
-

Cleaning Names

As discussed, there are a number of issues with the column names in this data set:

- Inconsistent capitalization
- Spaces in two word names
- A percent sign

We can clean them with the `clean_names()` function as follows. Let's see what changed.

```
clean <- clean_names(dirty)
data.frame(colnames(dirty), colnames(clean))
```

```
##   colnames.dirty. colnames.clean.
## 1              ID              id
## 2      First Name      first_name
## 3             SEX             sex
## 4             age             age
## 5        Semester        semester
## 6         Section         section
## 7       OVERALL % overall_percent
## 8           Grade           grade
## 9     Fav Subject     fav_subject
```

Note that spaces have been replaced with underscores, capitalization is consistent (in lower case), and the % sign was replaced with the word "percent".

Cleaning Data

Let's look at the full data set to see what issues remain.

```
clean
```

```
## # A tibble: 15 x 9
##    id    first_name sex     age semester section overall_percent grade
##    <chr> <chr>      <chr> <dbl> <chr>      <dbl>           <dbl> <chr>
##  1 716 ~ Destiny    Fema~    26 FALL           1              83 B
##  2 406 ~ Carla      Fema~    24 FALL           1              59 D
##  3 120 ~ Lynelle    Fema~    23 FALL           1              77 C
##  4 602 ~ Brooke     Fema~    24 FALL           2              86 B
##  5 218 ~ Brock      Male     26 FALL           1              91 A
##  6 526 ~ Aisha      Fema~    27 FALL           2              97 A
##  7 399 ~ Colleen    Fema~    24 FALL           1              85 B
##  8 329 ~ Dylan      Male     22 FALL           2              84 B
##  9 954 ~ Caleb      Male     23 FALL           2              81 B
## 10 733 ~ Travis     Male     25 FALL           2              88 B
## 11 394 ~ Luis       Male     23 FALL           2              90 A
## 12 997 ~ Emily      Fema~    22 FALL           1              83 B
## 13 325 ~ Cole       Male     25 FALL           1              94 A
## 14 150 ~ Sarah      Fema~    24 FALL           2              95 A
## 15 603 ~ Jacob      Male     23 FALL           1              99 A
## # ... with 1 more variable: fav_subject <lgl>
```

Function 2: Finding Duplicates

The `get_dupes()` function is provided for this purpose, and can be used with one or multiple variables.

```
clean %>% get_dupes(id)
```

```
## No duplicate combinations found of: id
## # A tibble: 0 x 10
## # ... with 10 variables: id <chr>, dupe_count <int>, first_name <chr>,
## #   sex <chr>, age <dbl>, semester <chr>, section <dbl>, overall_percent <dbl>,
## #   grade <chr>, fav_subject <lgl>
```

```
clean %>% get_dupes(overall_percent, grade)
```

```
## # A tibble: 2 x 10
##   overall_percent grade dupe_count id    first_name sex     age semester section
##             <dbl> <chr>      <int> <chr> <chr>      <chr> <dbl> <chr>      <dbl>
## 1              83 B              2 716 ~ Destiny    Fema~    26 FALL           1
## 2              83 B              2 997 ~ Emily      Fema~    22 FALL           1
## # ... with 1 more variable: fav_subject <lgl>
```

It shouldn't surprise us that no duplicates were found for "id", since these should be unique. However, note that there are two records with the same "overall_percent" of 83 and and "grade" of B. This clearly isn't an issue, but shows how `get_dupes` works.

Function 3: Creating Tables Quickly

- In base R, we could produce a table of grades with:

```
table(clean$grade)
```

```
##
## A B C D
## 6 7 1 1
```

- With the `taybl()` function from `janitor`, we would do this. Note that the flow is changed from horizontal to vertical, and we get both the count and the percentage.

```
tabyl(clean, grade)
```

```
##  grade n    percent
##      A 6 0.40000000
##      B 7 0.46666667
##      C 1 0.06666667
##      D 1 0.06666667
```

- If we want to get a bit fancier, using the tidyverse, this works too and produces the same output.

```
clean %>% tabyl(grade)
```

```
##  grade n    percent
##      A 6 0.40000000
##      B 7 0.46666667
##      C 1 0.06666667
##      D 1 0.06666667
```

Function 3: Making Tables Look Better

But that still isn't very attractive. Do we really need 8 decimal places in the percent column? Probably not.

A simple modification to the previous code, using the `adorn_pct_formatting` function will take care of this.

```
clean %>% tabyl(grade) %>% adorn_pct_formatting(digits = 0, affix_sign = TRUE)
```

```
##  grade n percent
##      A 6     40%
##      B 7     47%
##      C 1      7%
##      D 1      7%
```

There are a number of other `adorn` functions, including:

- `adorn_percentages` : convert a data.frame of counts to percentages
- `adorn_rounding` : round the numeric columns in a data.frame
- `adorn_title` : add column name to the top of a two-way tabyl (1)
- `adorn_totals` : append a totals row and/or column to a data.frame

(1) Makes tabyl prettier, but renders data.fram less useful for further manipulation.

Tables With More Than One Variable

This also works in base R, but note that it isn't clear what the first variable actually is in the output.

```
table(clean$grade, clean$sex)
```

```
##
##     Female Male
##   A      2    4
##   B      4    3
##   C      1    0
##   D      1    0
```

Whereas, with `taybl`, you can see both variable names.

```
clean %>% tabyl(grade, sex)
```

```
##  grade Female Male
##      A      2    4
##      B      4    3
##      C      1    0
##      D      1    0
```

Using the `adorn_totals` Function

Here, you can see how `adorn_totals()` adds row totals by default. You can get column totals, or both, with the `where` parameter.

```
clean %>% tabyl(grade, sex) %>% adorn_totals()
```

```
##  grade Female Male
##      A      2    4
##      B      4    3
##      C      1    0
##      D      1    0
##  Total      8    7
```

```
clean %>% tabyl(grade, sex) %>% adorn_totals(where = "col")
```

```
##  grade Female Male Total
##      A      2    4     6
##      B      4    3     7
##      C      1    0     1
##      D      1    0     1
```

```
clean %>% tabyl(grade, sex) %>% adorn_totals(where = c("row","col"))
```

```
##  grade Female Male Total
##      A      2    4     6
##      B      4    3     7
##      C      1    0     1
##      D      1    0     1
##  Total      8    7    15
```

Using the `adorn_percentages` Function

We already saw `the adorn_pct_formatting` function. But this function is a little bit different.

First, the default is for it to calculate by row, whether it is specified or not.

```
clean %>% tabyl(grade, sex) %>% adorn_percentages() %>% adorn_pct_formatting()
```

```
##  grade Female  Male
##      A  33.3% 66.7%
##      B  57.1% 42.9%
##      C 100.0%  0.0%
##      D 100.0%  0.0%
```

But if we add the "col" argument, note how the percentages are calculated now.

```
clean %>% tabyl(grade, sex) %>% adorn_percentages("col") %>% adorn_pct_formattin
```

```
##  grade Female  Male
##      A  25.0% 57.1%
##      B  50.0% 42.9%
##      C  12.5%  0.0%
##      D  12.5%  0.0%
```

Using the `adorn_ns` Function

```
clean %>% tabyl(grade, sex) %>%
  adorn_totals("row") %>%
  adorn_percentages("row") %>%
  adorn_pct_formatting %>%
  adorn_ns()
```

```
##  grade     Female      Male
##      A  33.3% (2) 66.7% (4)
##      B  57.1% (4) 42.9% (3)
##      C 100.0% (1)  0.0% (0)
##      D 100.0% (1)  0.0% (0)
##  Total  53.3% (8) 46.7% (7)
```

Note that above, the count goes to the back (in parenthesis) position. If you want percentage to be there, add the "front" argument.

```
clean %>% tabyl(grade, sex) %>%
  adorn_totals("row") %>%
  adorn_percentages("row") %>%
  adorn_pct_formatting %>%
  adorn_ns("front")
```

```
##  grade     Female      Male
##      A 2  (33.3%) 4 (66.7%)
##      B 4  (57.1%) 3 (42.9%)
##      C 1 (100.0%) 0  (0.0%)
##      D 1 (100.0%) 0  (0.0%)
##  Total 8  (53.3%) 7 (46.7%)
```

What about a 3-Variable Taybl?

```
clean %>% tabyl(grade, sex, section)
```

```
## $`1`
##  grade Female Male
##      A      0    3
##      B      3    0
##      C      1    0
##      D      1    0
##
## $`2`
##  grade Female Male
##      A      2    1
##      B      1    3
##      C      0    0
##      D      0    0
```

You can see how it adds the 3rd dimension (section) by listing it as 2 tables. This is similar to how `table` would do it too, but again just looks cleaner.

Removing Empty Columns

You may have noticed that the "Fav Subject" column (or "fav_subject" after cleaning), doesn't actually have any data in it. So if we look at a taybl of it, we see:

```
clean %>% tabyl(fav_subject)
```

```
##  fav_subject  n percent valid_percent
##           NA 15       1            NA
```

That isn't particularly helpful. We can use the `remove_empty()` function from `janitor` to prevent this, as follows:

```
clean_x <- clean %>% remove_empty(which = "cols")
names(clean)
```

```
## [1] "id"              "first_name"      "sex"             "age"
## [5] "semester"        "section"         "overall_percent" "grade"
## [9] "fav_subject"
```

```
names(clean_x)
```

```
## [1] "id"              "first_name"      "sex"             "age"
## [5] "semester"        "section"         "overall_percent" "grade"
```

Note that "fav_section" is not listed in the second set of column names.

Can be used with a "rows" argument too.

Using `remove_constant` to get rid of constant columns.

Does the "Semester" column really add any value here?

```
clean_y <- clean %>% remove_constant()
names(clean)
```

```
## [1] "id"              "first_name"      "sex"             "age"
## [5] "semester"        "section"         "overall_percent" "grade"
## [9] "fav_subject"
```

```
names(clean_y)
```

```
## [1] "id"              "first_name"      "sex"             "age"
## [5] "section"         "overall_percent" "grade"
```

Note that "semester" is not listed in the second set of column names. Also, It removes "fav_section" becasue empty is a specific type of constant.

BONUS Function:
`excel_numeric_to_date`

How many times have you had issues with dates in files that came from Excel?

When is "444155" anyway?

This is essentially a wrapper to functions from the `lubridate` package.

```
excel_numeric_to_date(44155)
```

```
## [1] "2020-11-20"
```

Oh, that's today!!!

Some Helpful Resources

- janitor Package on CRAN
- janitor Reference Manual
- GitHub page for janitor
- A helpful video I used in preparing these slides.
- R-Ladies Sydney Blog
- R-bloggers Examples