Tricks for cleaning your data in R

By Christine Zhang (ychristinezhang at gmail dot com)
Storytelling with Data Workshop at Boston University (June 6, 2017)

Data cleaning is a cumbersome task, and it can be hard to navigate in programming languages like R. When I was first learning R, I relied on familiar tools like Excel to clean my datasets before importing them into R to run analyses. This approach was often not ideal because it became hard to retrace my footsteps when I wanted to check my work. I always believed it would be better to have everything in one place, so I was motivated to learn how to clean my data in R.

R is a powerful tool for data cleaning and analysis. By default, it leaves a trail of code that documents all the work you've done, which makes it extremely useful for creating reproducible workflows.

In this workshop, I'll show you some examples of real-life "messy" datasets, the problems they present for analysis in R, and the "tidy" solutions to these problems.

Underlying this workshop is Hadley Wickham's principle of Tidy Data, which you can read about here.

1. Finding and replacing non-numeric characters like , and $

Since we're in Boston, let's check out the city's Open Data portal, where the local government puts up datasets that are free for the public to analyze.

The Employee Earnings Report is one of the more interesting ones, because it gives payroll data for every person on the municipal payroll. It's where the *Boston Globe* gets stories like these every year:

- "64 City of Boston workers earn more than $250,000" (February 6, 2016)
- "Police detective tops Boston's payroll with a total of over $403,000" (February 14, 2017)

Let's take at the February 14 story from this year. The story begins:

> "A veteran police detective took home more than $403,000 in earnings last year, topping the list of Boston's highest-paid employees in 2016, newly released city payroll data show."

What if we wanted to check this number using the Employee Earnings Report?

We can use the `read.csv()` to load the csv file into R. We will call this data frame `salary`.

Note we use `stringsAsFactors = F`:

```
salary <- read.csv('employee-earnings-report-2016.csv')
```

We can use the `head()` function to inspect the first six rows of `salary`:

```
head(salary)
```

```
##                      NAME          DEPARTMENT_NAME
## 1          Abadi,Kidani A     Assessing Department
## 2        Abasciano,Joseph Boston Police Department
## 3  Abban,Christopher John   Boston Fire Department
## 4           Abbasi,Sophia            Green Academy
## 5 Abbate-Vaughn,Jorgelina     BPS Ellis Elementary
## 6        Abberton,James P  Public Works Department
##                       TITLE     REGULAR RETRO     OTHER   OVERTIME
## 1         Property Officer  $46,291.98         $300.00
## 2           Police Officer   $6,933.66         $850.00    $205.92
## 3             Fire Fighter $103,442.22         $550.00 $15,884.53
```

```
## 4     Manager (C) (non-ac)  $18,249.83
## 5                  Teacher  $84,410.28       $1,250.00
## 6 Maint Mech (Carpenter)##  $41,449.16          $81.00  $8,807.47
##      INJURED    DETAIL QUINN.EDUCATION.INCENTIVE TOTAL.EARNINGS POSTAL
## 1                                                    $46,591.98  02118
## 2 $74,331.86                           $15,258.44    $97,579.88  02132
## 3            $4,746.50                              $124,623.25  02132
## 4                                                    $18,249.83  02148
## 5                                                    $85,660.28  02481
## 6                                                    $50,337.63  02127
```

There are a lot of columns. Let's simplify by selecting the ones of interest: `NAME`, `DEPARTMENT_NAME`, and `TOTAL.EARNINGS`. We can do this using the `select()` function in `dplyr`. We will save them into a new data frame, `salary.selected`.

We load the `dplyr` pacakge using `library('dplyr')`:

```
# install.packages('dplyr') # if you don't already have the package
library('dplyr') # load the dplyr package
```

```
salary.selected <- select(salary, # the data frame
                          NAME, DEPARTMENT_NAME, TOTAL.EARNINGS) # the variables to select
```

We can also change these names to lowercase names for easier typing using `tolower()`:

```
names(salary.selected) <- tolower(names(salary.selected)) # change variable names to lowercase
```

Let's use `head()` to visually inspect the first six rows of `salary.selected`:

```
head(salary.selected)
```

```
##                       name          department_name total.earnings
## 1          Abadi,Kidani A     Assessing Department     $46,591.98
## 2        Abasciano,Joseph Boston Police Department     $97,579.88
## 3  Abban,Christopher John   Boston Fire Department    $124,623.25
## 4           Abbasi,Sophia             Green Academy     $18,249.83
## 5 Abbate-Vaughn,Jorgelina     BPS Ellis Elementary     $85,660.28
## 6        Abberton,James P  Public Works Department     $50,337.63
```

Now let's try sorting the data by `total.earnings` using the `arrange()` function in `dplyr`:

```
salary.sort <-  arrange(salary.selected, # dataset to sort
                        total.earnings) # variable to sort by
```

We can use `head()` to visually inspect `salary.sort`:

```
head(salary.sort)
```

```
##                  name              department_name total.earnings
## 1 Fowlkes,Lorraine E.          Boston City Council      $1,000.00
## 2    Lally,Bernadette          Boston City Council      $1,000.00
## 3        Nolan,Andrew             Parks Department      $1,000.00
## 4  White-Pilet,Yoni A BPS Substitute Teachers/Nurs      $1,006.53
## 5         Dunn,Lori D         BPS East Boston High      $1,010.05
## 6     Hopkins,Susan R        BPS Mather Elementary      $1,017.94
```

What went wrong?

The problem is that there are non-numeric characters, `,` and `$`, in the `total.earnings` column. We can see with `class()` that `total.earnings` is recognized as factor rather than numeric.

```
class(salary.selected$total.earnings)
```

```
## [1] "factor"
```

We need to find the , and $ in `total.earnings` and remove them—in computer science lingo, "pattern matching and replacement." The `str_replace` function in the `stringr` package lets us do this easily.

Let's start by removing the comma and write the result to the original column. (The format for calling a column from a data frame in R is `data frame.name$column.name`)

We load the `stringr` pacakge using `library('stringr')`:

```
# install.packages('stringr') # if you don't already have the package
library('stringr') # load the stringr package
```

```
salary.selected$total.earnings <- str_replace(
    salary.selected$total.earnings, # column we want to search
    pattern = ',', # what to find
    replacement = '' # what to replace it with
)
```

Using `head()` to visually inspect `salary.selected`, we see that the commas are gone:

```
head(salary.selected) # this works - the commas are gone
```

```
##                       name          department_name total.earnings
## 1          Abadi,Kidani A     Assessing Department      $46591.98
## 2        Abasciano,Joseph Boston Police Department      $97579.88
## 3  Abban,Christopher John   Boston Fire Department     $124623.25
## 4           Abbasi,Sophia            Green Academy      $18249.83
## 5 Abbate-Vaughn,Jorgelina     BPS Ellis Elementary      $85660.28
## 6        Abberton,James P  Public Works Department      $50337.63
```

The dollar sign $ is trickier. Let's try doing the exact same thing, except let's set `pattern = '$'` instead of `pattern = ','`:

```
salary.selected$total.earnings <- str_replace(
    salary.selected$total.earnings, # column we want to search
    pattern = '$', # what to find
    replacement = '' # what to replace it with
)
```

Using `head()` to visually inspect `salary.selected`, we see that the dollar signs are still there:

```
head(salary.selected) # this didn't work - the dollar signs are still there
```

```
##                       name          department_name total.earnings
## 1          Abadi,Kidani A     Assessing Department      $46591.98
## 2        Abasciano,Joseph Boston Police Department      $97579.88
## 3  Abban,Christopher John   Boston Fire Department     $124623.25
## 4           Abbasi,Sophia            Green Academy      $18249.83
## 5 Abbate-Vaughn,Jorgelina     BPS Ellis Elementary      $85660.28
## 6        Abberton,James P  Public Works Department      $50337.63
```

$ is known as a "special character" or "metacharacter", along with `* + . ? [ ] ^ { } | ( ) \`). Dealing with these is a bit complicated (more info on them here), but basically if we want R to literally find a dollar sign, $, in `salary$total.earnings`, we can add two backslashes before the dollar sign: `\\$`, which lets R know to ignore or "escape" the special attributes of $ on its own.

```
salary.selected$total.earnings <- str_replace(
    salary.selected$total.earnings, # column we want to search
    pattern = '\\$', # what to find
    replacement = '' # what to replace it with
)
```

Using `head()` to visually inspect `salary.selected`, we see that the dollar signs are gone:

```
head(salary.selected)
```

```
##                       name          department_name total.earnings
## 1          Abadi,Kidani A     Assessing Department       46591.98
## 2        Abasciano,Joseph Boston Police Department       97579.88
## 3  Abban,Christopher John   Boston Fire Department      124623.25
## 4           Abbasi,Sophia             Green Academy       18249.83
## 5 Abbate-Vaughn,Jorgelina     BPS Ellis Elementary       85660.28
## 6        Abberton,James P  Public Works Department       50337.63
```

Now can we use `arrange()` to sort the data by `total.earnings`?

```
salary.sort <- arrange(salary.selected,
                       total.earnings)
```

Let's take a look at `salary.sort`, using `head()`:

```
head(salary.sort)
```

```
##                       name      department_name total.earnings
## 1          Charles,Yveline   BPS Transportation          10.07
## 2     Jean Baptiste,Hugues   BPS Transportation          10.12
## 3            Piper,Sarah A   BPS Transportation          10.47
## 4       Laguerre,Yolaine M   BPS Transportation          10.94
## 5             Mayo,Wanda M Food & Nutrition Svc         100.00
## 6 Rosario Severino,Yomayra Food & Nutrition Svc         100.00
```

What's the problem?

Again, we can use the `class()` function to check on how the `total.earnings` variable is encoded.

```
class(salary.selected$total.earnings) # a character, not numeric
```

```
## [1] "character"
```

It's a "character" now (still not numeric), because we didn't tell R that it should be numeric. We can do this with `as.numeric()`:

```
salary.selected$total.earnings <- as.numeric(salary.selected$total.earnings)
```

Now let's run `class()` again:

```
class(salary.selected$total.earnings)
```

```
## [1] "numeric"
```

Now let's sort using `arrange()`:

```
salary.sort <- arrange(salary.selected,
                       total.earnings)

head(salary.sort) # ascending order by default
```

```
##                     name      department_name total.earnings
## 1      Jameau,Bernadette   BPS Transportation           2.14
## 2  Bridgewaters,Sandra J   BPS Transportation           2.50
## 3     Milian,Sonia Maria   BPS Transportation           3.85
## 4 Burke II,Myrell Nadine   BPS Transportation           4.38
## 5    Gillard Jr.,Trina F Food & Nutrition Svc           5.00
## 6     Lucas,Mona-Lisa L. Food & Nutrition Svc           5.36
```

One last thing: we have to specify `desc(total.earnings)` within `arrange()` because the function by default sorts the data in ascending order.

```
salary.sort <- arrange(salary.selected,
                       desc(total.earnings)) # descending order

head(salary.sort) # Waiman Lee from the Boston PD is the highest paid city employee
```

```
##                name          department_name total.earnings
## 1       Lee,Waiman Boston Police Department       403408.6
## 2  Josey,Windell C. Boston Police Department       396348.5
## 3    Painten,Paul A Boston Police Department       373959.3
## 4     Brown,Gregory Boston Police Department       351825.5
## 5     Hosein,Haseeb Boston Police Department       346105.2
## 6 Kervin,Timothy M. Boston Police Department       343818.2
```

We see that Waiman Lee from the Boston PD is the top earner with >403,408 per year, just as the *Boston Globe* article states.

The Boston Police Department has a lot of high earners. We can figure out the average earnings by department, which we'll call `average.earnings`, by using the `group_by()` and `summarise()` functions in dplyr.

Now would be a good time to introduce %>%, known as the pipe operator.

`%>%` is an extremely valuable tool in R, because it allows functions to be chained rather than nested. `%>%` looks strange but can be read as "then"—it tells R to do whatever comes after it to the stuff comes before it.

```
salary.average <- salary.selected %>% # take the salary.selected data frame, THEN
  group_by(department_name) %>% # group by department_name, THEN
  summarise(average.earnings = mean(total.earnings)) # calculate the mean of total.earnings for each de
```

If we were to do this without piping, it would look like

```
summarise(group_by(salary.selected, department_name), average.earnings = mean(total.earnings))
```

Let's look at `salary.average` using `head()`:

```
head(salary.average) # first six rows of average salary by department (alphabetical order)
```

```
## # A tibble: 6 x 2
##                department_name average.earnings
##                         <fctr>            <dbl>
## 1               Accountability        102073.28
## 2              Achievement Gap         60105.52
## 3  Alighieri Montessori School         55160.03
## 4          ASD Human Resources         67236.15
## 5  ASD Intergvernmtl Relations         83787.58
## 6 ASD Office of Budget Mangmnt         73946.04
```

We can find the Boston Police Department using `filter()`:

```
salary.average %>% filter(department_name == 'Boston Police Department')
```

```
## # A tibble: 1 x 2
##            department_name average.earnings
##                      <fctr>            <dbl>
## 1 Boston Police Department         124787.2
```

Exercise: The `salary.average` data frame is currently ordered alphabetically by department. How would you sort this dataset by average earnings, from highest to lowest?

2. Merging datasets

Now we have two main datasets, `salary.sort` (the salary for each person, sorted from high to low) and `salary.average` (the average salary for each department). What if I wanted to merge these two together, so I could see side-by-side each person's salary compared to the average for their department?

We want to join by the `department_name` variable, since that is consistent across both datasets. Let's put the merged data into a new dataframe, `salary.merged`:

```
salary.merged <- merge(x = salary.sort, y = salary.average, by = 'department_name')
```

Now we can see the department average, `salary.average`, next to the individual's salary, `total.earnings`:

```
head(salary.merged)
```

```
##   department_name                     name total.earnings
## 1  Accountability Guttenberg,Nicole Desiree       120132.7
## 2  Accountability  Hedley-Mitchell,Angela E       120373.0
## 3  Accountability            Martin,Dean M.       117132.9
## 4  Accountability         Solomon,Stacey L.       109129.7
## 5  Accountability            Lipkin,Linda S       115418.4
## 6  Accountability          Anderson,Daniel        108408.9
##   average.earnings
## 1         102073.3
## 2         102073.3
## 3         102073.3
## 4         102073.3
## 5         102073.3
## 6         102073.3
```

3. Reshaping data

Here's a dataset on unemployment rates by country from 2012 to 2016, from the International Monetary Fund's World Economic Outlook database (available here).

When you download the dataset, it comes in an Excel file. We can use the `read_excel()` from the `readxl` package to load the file into R.

We load the `readxl` pacakge using `library('readxl')`:

```
# install.packages('readxl') # if you don't already have the package
library('readxl') # load the readxl package
```

```
unemployment <- read_excel('unemployment.xlsx')
```

Right now, the data are in what's commonly referred to as "wide" format, meaning the variables (unemployment rate for each year) are spread across rows. This might be good for presentation, but it's not great for certain calculations or graphing. "Wide" format data also becomes confusing if other variables are added.

We need to change the format from "wide" to "long," meaning that the columns (`2012`, `2013`, `2014`, `2015`, `2016`) will be converted into a new variable, which we'll call `Year`, with repeated values for each country. And the unemployment rates will be put into a new variable, which we'll call `Rate.Unemployed`.

We'd like the data to look like this:

```
## # A tibble: 10 x 3
##    Country  Year   Rate.Unemployed
##      <chr> <chr>             <chr>
##  1 Albania  2012              13.4
##  2 Albania  2013                16
##  3 Albania  2014              17.5
##  4 Albania  2015              17.1
##  5 Albania  2016              16.1
##  6 Algeria  2012                11
##  7 Algeria  2013 9.829000000000001
##  8 Algeria  2014              10.6
##  9 Algeria  2015            11.214
## 10 Algeria  2016            10.498
```

To do this, we'll use the `gather()` function in `tidyr` to create a new data frame, `unemployment.long`.

We load the `tidyr` pacakge using `library('tidyr')`:

```
# install.packages('tidyr')
library('tidyr') # load the tidyr package
```

```
unemployment.long <- gather(unemployment, # data to reshape
                            Year, # column we want to create from the rows
                            Rate.Unemployed, # the values of interest
                            -Country # already a column in the data
                            )
```

Inspecting `unemployment.long` using `head()` shows that we have successfully created a long dataset.

```
head(unemployment.long)
```

```
## # A tibble: 6 x 3
##     Country  Year Rate.Unemployed
##       <chr> <chr>            <chr>
## 1   Albania  2012             13.4
## 2   Algeria  2012               11
## 3 Argentina  2012              7.2
## 4   Armenia  2012             17.3
## 5 Australia  2012            5.217
## 6   Austria  2012            4.933
```

But there's a problem. `Rate.Unemployed` is not recognized as a numeric variable.

```
class(unemployment.long$Rate.Unemployed) ## "character", not "numeric"
```

```
## [1] "character"
```

Why do you think this is? (hint, use `head()` to find out)

We can use `as.numeric()` to convert `Rate.Unemployed` to a numeric variable.

```
unemployment.long$Rate.Unemployed <- as.numeric(unemployment.long$Rate.Unemployed)
```

```
## Warning: NAs introduced by coercion
```

`str()` is another way to check how variables are encoded. It returns the structure of the entire dataset:

```
str(unemployment.long) # Rate.Unemployed is now "num", which stands for "numeric"
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    560 obs. of  3 variables:
##  $ Country        : chr  "Albania" "Algeria" "Argentina" "Armenia" ...
##  $ Year           : chr  "2012" "2012" "2012" "2012" ...
##  $ Rate.Unemployed: num  13.4 11 7.2 17.3 5.22 ...
```

4. Calculating year-over-year change in panel data

Sort the data by `Country` using the `arrange()` function in `dplyr`:

```
unemployment.long <- arrange(unemployment.long, # data frame to sort
                             Country, Year) # variables to sort by
```

The above code is equivalent to the following, which uses the pipe operator, `%>%`:

```
unemployment.long <- unemployment.long %>% # Take the unemployment.long data frame, THEN
  arrange(Country, Year) # sort it by Country and then Year.
```

Now let's use `head()` to inspect the `unemployment.long`, but instead of the first six rows (the default), let's look at the first five:

```
head(unemployment.long, 5) # First five rows of the data
```

```
## # A tibble: 5 x 3
##   Country  Year Rate.Unemployed
##     <chr> <chr>           <dbl>
## 1 Albania  2012            13.4
## 2 Albania  2013            16.0
## 3 Albania  2014            17.5
## 4 Albania  2015            17.1
## 5 Albania  2016            16.1
```

This type of data is known in time-series analysis as a panel; each country is observed every year from 2012 to 2016.

For Albania, the percentage point change in unemployment rate from 2012 to 2013 would be 16 - 13.4 = 2.5 percentage points. What if I wanted the year-over-year change in unemployment rate for every country?

This is an example where having a tidy dataset really helps. We can use the `mutate()` function in `dplyr` to create a new variable, `Change`, which is the difference between `Rate.Unemployed` and `lag(Rate.Unemployed)` (the default for `lag()` is 1 position, which is good for us since we want the change from the previous year).

```
unemployment.long <- unemployment.long %>% # take the unemployment.long dataset, THEN
  mutate(Change = Rate.Unemployed - lag(Rate.Unemployed)) # create a variable called Change
```

Let's inspect the first five rows again, using `head()`:

```
head(unemployment.long, 5)
```

```
## # A tibble: 5 x 4
##   Country  Year Rate.Unemployed Change
##     <chr> <chr>           <dbl>  <dbl>
```

```
## 1 Albania  2012            13.4     NA
## 2 Albania  2013            16.0    2.6
## 3 Albania  2014            17.5    1.5
## 4 Albania  2015            17.1   -0.4
## 5 Albania  2016            16.1   -1.0
```

So far so good. It also makes sense that Albania's `Change` is `NA` in 2012, since the dataset doesn't contain any unemployment figures before the year 2012.

But a closer inspection of the data reveals a problem. What if we used `tail()` to look at the *last* 5 rows of the data?

```
tail(unemployment.long, 5)
```

```
## # A tibble: 5 x 4
##   Country  Year Rate.Unemployed  Change
##     <chr> <chr>           <dbl>   <dbl>
## 1 Vietnam  2012            2.74 -18.493
## 2 Vietnam  2013            2.75   0.010
## 3 Vietnam  2014            2.05  -0.700
## 4 Vietnam  2015            2.40   0.350
## 5 Vietnam  2016            2.40   0.000
```

Why does Vietnam have a -18.493 percentage point change in 2012?

```
unemployment.long <- unemployment.long %>%
  group_by(Country) %>%
  mutate(Change = Rate.Unemployed - lag(Rate.Unemployed))

tail(unemployment.long, 5)
```

```
## Source: local data frame [5 x 4]
## Groups: Country [1]
##
## # A tibble: 5 x 4
##   Country  Year Rate.Unemployed Change
##     <chr> <chr>           <dbl>  <dbl>
## 1 Vietnam  2012            2.74     NA
## 2 Vietnam  2013            2.75   0.01
## 3 Vietnam  2014            2.05  -0.70
## 4 Vietnam  2015            2.40   0.35
## 5 Vietnam  2016            2.40   0.00
```

5. Recoding numerical variables into categorical ones

Here's a list of some attendees for today's workshop, with names and contact info removed.

```
attendees <- read.csv('attendees.csv', stringsAsFactors = F)
head(attendees)
```

```
##           Occupation                  Job.title Age.group  Gender
## 1       Data Analyst       Data Quality Analyst     30-39    Male
## 2        PhD Student Student/Research Assistant     18-29    Male
## 3          Education               Data Analyst     18-29  Female
## 4            Manager                BAS Manager     30-39    Male
## 5 Government Finance        Performance Analyst   30 - 39    Male
## 6           Engineer           Display Engineer     30-39  Female
```

```
##   State.Province         Education
## 1             MA Bachelor's Degree
## 2             MA Bachelor's Degree
## 3       Kentucky   Master's Degree
## 4             MA Bachelor's Degree
## 5             MA   Master's Degree
## 6             MA Bachelor's Degree
##   Which.data.subject.area.are.you.most.interested.in.working.with...Select.up.to.three.
## 1                                                                                Retail
## 2                                                                                Sports
## 3                                                                                Retail
## 4                                                                             Education
## 5                                          Environment, Finance, Food and agriculture
## 6                                          Environment, Finance, Food and Agriculture
##
## 1
## 2
## 3
## 4
## 5
## 6 Explore the field of data storytelling, including career options, Improve my ability to write with
##   Which.type.of.laptop.will.you.bring. College.or.University.Name
## 1                                   PC
## 2                                   PC          Boston University
## 3                                   PC
## 4                                   PC          Boston University
## 5                                  MAC
## 6           Advanced Data Storytelling
##   Major.or.Concentration College.Year
## 1
## 2          Biostatistics          PhD
## 3
## 4                  PEMBA     Graduate
## 5
## 6
##   Which.Digital.Badge.track.best.suits.you.
## 1                Advanced Data Storytelling
## 2                Advanced Data Storytelling
## 3                Advanced Data Storytelling
## 4                Advanced Data Storytelling
## 5                Advanced Data Storytelling
## 6                Advanced Data Storytelling
##   Which.session.would.you.like.to.attend.
## 1                               June 5-9
## 2                               June 5-9
## 3                               June 5-9
## 4                               June 5-9
## 5                               June 5-9
## 6                               June 5-9
##                         Choose.your.status.
## 1            Nonprofit, Academic, Government
## 2                                    Student
## 3            Nonprofit, Academic, Government
## 4                                    Student
```

```
## 5 Nonprofit, Academic, Government Early Bird
## 6                                Professional
```

What if we wanted to quickly see the age distribution of attendees?

```
table(attendees$Age.group)
```

```
##
##   18-29 30 - 39   30-39
##       4       1       7
```

There's an inconsistency in the labeling of the `Age.group` variable here. We can fix this using `ifelse()` by replacing the "30 - 39" with "30-39":

```
attendees$Age.group <- ifelse(attendees$Age.group == '30 - 39', # if attendees$Age.group == '30 - 39'
                              '30-39', # replace attendees$Age.group with '30-39'
                              attendees$Age.group) # otherwise, keep attendees$Age.group values the sam
```

This might seem trivial for just one value, but it's useful for larger datasets.

```
table(attendees$Age.group)
```

```
##
## 18-29 30-39
##     4     8
```

Now let's take a look at the professional status of attendees, labeled in `Choose.your.status.`:

```
table(attendees$Choose.your.status.)
```

```
##
##            Nonprofit, Academic, Government
##                                          3
## Nonprofit, Academic, Government Early Bird
##                                          1
##                                Professional
##                                          3
##                                     Student
##                                          5
```

"Nonprofit, Academic, Government" and "Nonprofit, Academic, Government Early Bird" seem to be the same. We can use `ifelse()` (and the R designation `|` for "or") to combine these two categories into one big category, "Nonprofit/Gov". Let's create a new variable, `status`, for our simplified categorization.

```
attendees$status <- ifelse(attendees$Choose.your.status. == 'Nonprofit, Academic, Government' |
                             attendees$Choose.your.status. == 'Nonprofit, Academic, Government Early Bi
                           'Nonprofit/Gov',
                           attendees$Choose.your.status.)
table(attendees$status)
```

```
##
## Nonprofit/Gov  Professional       Student
##             4             3             5
```

What else?

- How would you use `ifelse()` and `|` to create a new variable in the `attendees` data (let's call it `status2`) that has just two categories, "Student" and "Other"?
- How would you rename the variables in the `attendees` data to make them easier to work with?

- What are some other issues with this dataset? How would you solve them using what we've learned?
- What are some other "messy" data issues you've encountered?