Tidy data

PS239T

04 February, 2019

- I adapted the following content from Wickham's R for Data Science, his earlier paper published in the Journal of Statistical Software, Efficient R Programming by Gillespie and Lovelace, and R Programming for Data Science by Roger P. Peng
- The big picture
  - Tidying data with **tidyr**
  - Processing data with **dplyr**

These two packages don't do anything new, but simplify most common tasks in data manipulation. Plus, they are fast.

1. What're tidy principles

1. The dataset: strucure (physical layout) + semantics (meaning)

1.1. Data structure

- rows and columns

1.2. Data semantics

- variables and values (numbers or strings)

2. Tidy tada: "provide a standard way to organize data values within a dataset."

2.1 Tidy principles

- 1. Each variable forms a column.
- 2. Each observation forms a row.
- 3. Each type of observational unit forms a table.

```r
# load library
library(tidyverse)
```

```
## -- Attaching packages ----------------- tidyverse 1.2.1 --

## v ggplot2 3.1.0     v purrr   0.3.0
## v tibble  2.0.1     v dplyr   0.7.8
## v tidyr   0.8.2     v stringr 1.3.1
## v readr   1.3.1     v forcats 0.3.0

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

```r
# tidy data example
table1
```

```
## # A tibble: 6 x 4
##   country      year  cases population
##   <chr>       <int>  <int>      <int>
## 1 Afghanistan  1999    745   19987071
## 2 Afghanistan  2000   2666   20595360
## 3 Brazil       1999  37737  172006362
## 4 Brazil       2000  80488  174504898
## 5 China        1999 212258 1272915272
## 6 China        2000 213766 1280428583
```

Practically, this approach is good because you're going to have consistency in the format of data across all the projects you're working on. Also, tidy data works well with key packages (e.g., dplyr, ggplot2) in R.

Computationally, this approach is useful for vectorized programming because "different variables from the same observation are always paired". To remind you, vectorized means a function applies to a vector treats each element individually.

3. Messy datasets

3.1. Signs of messy datasets

- 1. Column headers are values, not variable names.
- 2. Multiple variables are not stored in one column.
- 3. Variables are stored in both rows and columns.
- 4. Multiple types of observational units are stored in the same table.
- 5. A single observational unit is stored in multiple tables.

4. Tidy tools

Either input or output or both of them can be messy. Tidy tools can fix these problems.

4.1. Tidyr manipulation (organizing)

4.1.1. Gather (from wide to long)

```r
table4a
```

```
## # A tibble: 3 x 3
##   country     `1999` `2000`
## * <chr>        <int>  <int>
## 1 Afghanistan    745   2666
## 2 Brazil       37737  80488
## 3 China       212258 213766
```

```r
# from long to wide
table4a %>% # The first argument is the data frame
  gather('1999', '2000', key = "year", value = "population")
```

```
## # A tibble: 6 x 3
##   country     year  population
##   <chr>       <chr>      <int>
## 1 Afghanistan 1999         745
## 2 Brazil      1999       37737
## 3 China       1999      212258
## 4 Afghanistan 2000        2666
## 5 Brazil      2000       80488
## 6 China       2000      213766
```

```
# save the file
table4a_wide <- table4a %>%
  gather('1999', '2000', key = "year", value = "population")
```

4.1.2. Spread (from wide to long)

```
table4a_wide %>%
  spread(key = "year", value = "population")
```

```
## # A tibble: 3 x 3
##   country     `1999` `2000`
##   <chr>        <int>  <int>
## 1 Afghanistan    745   2666
## 2 Brazil       37737  80488
## 3 China       212258 213766
```

4.1.3. Separate (split one into many columns)

```
table3
```

```
## # A tibble: 6 x 3
##   country      year rate
## * <chr>       <int> <chr>
## 1 Afghanistan  1999 745/19987071
## 2 Afghanistan  2000 2666/20595360
## 3 Brazil       1999 37737/172006362
## 4 Brazil       2000 80488/174504898
## 5 China        1999 212258/1272915272
## 6 China        2000 213766/1280428583
```

```
table3 %>%
  separate(rate, into = c("cases" , "population"))
```

```
## # A tibble: 6 x 4
##   country      year cases  population
##   <chr>       <int> <chr>  <chr>
## 1 Afghanistan  1999 745    19987071
## 2 Afghanistan  2000 2666   20595360
## 3 Brazil       1999 37737  172006362
## 4 Brazil       2000 80488  174504898
## 5 China        1999 212258 1272915272
## 6 China        2000 213766 1280428583
```

```
table3_separated <- table3 %>%
  separate(rate, into = c("cases" , "population"))
```

4.1.4. Unite (multiple columns into one column)

```
table3_separated %>%
  unite(rate, cases, population)
```

```
## # A tibble: 6 x 3
##   country      year rate
##   <chr>       <int> <chr>
## 1 Afghanistan  1999 745_19987071
## 2 Afghanistan  2000 2666_20595360
## 3 Brazil       1999 37737_172006362
## 4 Brazil       2000 80488_174504898
## 5 China        1999 212258_1272915272
## 6 China        2000 213766_1280428583
```

4.2. dplyr manipulation (process)

dplyr is better than the base R approaches to data processing:

- fast to run (due to the C++ backed) and intuitive to type
- works well with tidy data and databases

There are so many useful functions that come from dplyr: filter(), slice(), arrange(), select(), rename(), distinct(), mutate(), summarize(), sample_n().

The pipe operator %>% originally comes from the magrittr package. The idea behind the pipe operator is similar to what we learned about chaining functions in high school. f: B -> C and g: A -> B can be expressed as $f(g(x))$. Basically, the pipe operator chains operations.

Some functions are designed to work together. For instance, the group_by function defines the strata that you're going to use for summary statistics. Then, use summarise() or summarize() for producing summary statistics.

```
library(dplyr)
library(gapminder) # load gapminder package

gapminder # the data is already organized by tidy principles
```

```
## # A tibble: 1,704 x 6
##    country     continent  year lifeExp      pop gdpPercap
##    <fct>       <fct>     <int>   <dbl>    <int>     <dbl>
##  1 Afghanistan Asia       1952    28.8  8425333      779.
##  2 Afghanistan Asia       1957    30.3  9240934      821.
##  3 Afghanistan Asia       1962    32.0 10267083      853.
##  4 Afghanistan Asia       1967    34.0 11537966      836.
##  5 Afghanistan Asia       1972    36.1 13079460      740.
##  6 Afghanistan Asia       1977    38.4 14880372      786.
##  7 Afghanistan Asia       1982    39.9 12881816      978.
##  8 Afghanistan Asia       1987    40.8 13867957      852.
##  9 Afghanistan Asia       1992    41.7 16317921      649.
## 10 Afghanistan Asia       1997    41.8 22227415      635.
## # ... with 1,694 more rows
```

```
names(gapminder) # the column names
```

```
## [1] "country"   "continent" "year"      "lifeExp"   "pop"       "gdpPercap"
```

```r
gapminder %>%
  filter(continent == "Europe") %>% # filter by Europe
  group_by(country) %>% # group by country
  summarize(Mean = mean(gdpPercap)) %>% # collapse data by mean
  top_n(5, Mean) %>% # count only top 5 by mean
  arrange(desc(Mean)) # arrange by descending order
```

```
## # A tibble: 5 x 2
##   country       Mean
##   <fct>        <dbl>
## 1 Switzerland 27074.
## 2 Norway      26747.
## 3 Netherlands 21749.
## 4 Denmark     21672.
## 5 Germany     20557.
```

```r
gapminder <- gapminder %>%
  rename(population = pop) # rename pop variable (old name = new name)

gapminder$pop # It should be NULL
```

```
## Warning: Unknown or uninitialised column: 'pop'.
```

```
## NULL
```

Another powerful feature of dplyr package is select function.

```r
survey_results <- read.csv("./data/ps239t_responses.csv") # load csv file using relativre path

names(survey_results) # column names
```

```
##  [1] "Timestamp"
##  [2] "Name"
##  [3] "I.know.how.to.import.and.save.various.forms..e.g...csv..rds..of.files.in.R."
##  [4] "I.know.how.to.use.pipe.operator.......in.R."
##  [5] "I.know.how.to.use.ggplot2.to.visualize.data.in.R."
##  [6] "I.can.explain.the.differences.between.character.and.factor.variables.in.R."
##  [7] "I.can.explain.what.regular.expressions.are."
##  [8] "I.can.explain.what.tuples.are.in.Python."
##  [9] "I.know.how.to.use.pandas.library.in.Python."
## [10] "I.can.explain.what.high.dimensional.data.are."
## [11] "I.can.explain.the.differences.between.supervised.and.unsupervised.machine.learning."
```

```r
survey_results_r <- survey_results %>%
  dplyr::select(ends_with("in.R.")) # What does it show?

names(survey_results_r) <- c("import", "pipe", "ggplot2", "factor") # change col names using base r fun

survey_results_r %>%
  gather("type", "value", import:factor) %>% # reshape from wide to long
  group_by(type, value) %>% # group by type and value
  summarise(n = n()) %>%  # summarise n
  mutate(freq = n/sum(n)) %>% # calculate frequency
  arrange(desc(freq)) %>% # descending order by freq
  filter(value == "No") # filter by no
```

```
## # A tibble: 4 x 4
```

```
## # Groups:   type [4]
##   type    value     n  freq
##   <chr>   <chr> <int> <dbl>
## 1 ggplot2 No        7 0.778
## 2 factor  No        6 0.667
## 3 pipe    No        6 0.667
## 4 import  No        2 0.222
```

** Tips **

- Set column names: names() in data frame, colnames() in matrix
- Set row names: row.names() in data frame, rownames() in matrix

4.3. Modeling

Will not be extensively covered in this course.

4.4. Visualization

Will be covered in the future session.