12 - Tidy Data

ST 597 | Spring 2016
University of Alabama

12-tidy.pdf

Contents

1 Tidy Data 2
  1.1 Get the Rate (cases/population) 2
  1.2 Why Tidy Data 4

2 Main \texttt{tidyr} functions 4
  2.1 \texttt{gather()} into long form 5
  2.2 \texttt{spread()} into wide form 6
  2.3 \texttt{separate()} 8
  2.4 \texttt{unite()} 10

3 Missing Data 11
  3.1 Missing Values 11

4 Your Turn 12
  4.1 Problem 1: Tornado 12
  4.2 Problem 2: Time of Day 13
  4.3 Problem 3: Pew Survey 13

5 Other functions in \texttt{tidyr} package 13

Required Packages and Data

\texttt{library(tidyverse)}
1 Tidy Data

The textbook has some examples of tidy and untidy data

```r
library(tidyverse)
data(package="tidyr")
# table1, table2, table3, table4a, table4b
```

1.1 Get the Rate (cases/population)

For each table, calculate the \( \text{rate} = \frac{\text{cases}}{\text{population}} \).

1.1.1 Table 1

```r
table1
#> # A tibble: 6 × 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
```

**Your Turn #1**

What **dplyr** function can be used to create the **rate** column?

1.1.2 Table 2

```r
table2
#> # A tibble: 12 × 4
#> country year type count
#> <chr> <int> <chr> <int>
#> 1 Afghanistan 1999 cases 745
#> 2 Afghanistan 1999 population 19987071
#> 3 Afghanistan 2000 cases 2666
#> 4 Afghanistan 2000 population 20595360
#> 5 Brazil 1999 cases 37737
#> 6 Brazil 1999 population 172006362
#> 7 Brazil 2000 cases 80488
#> 8 Brazil 2000 population 174504898
#> 9 China 1999 cases 212258
#> 10 China 1999 population 1272915272
#> 11 China 2000 cases 213766
#> 12 China 2000 population 1280428583
```
Your Turn #2

What needs to be done to calculate the rate?
Hint: what constitutes an observation, and what are the variables? Another way to consider is by identifying the primary key(s) of the table.

1.1.3 Table 3

table3
#> # A tibble: 6 × 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

Your Turn #3

What needs to be done to actually calculate the rate?

1.1.4 Tables 4a and 4b

table4a
#> # A tibble: 3 × 3
#>   country `1999` `2000`
#> * <chr> <int> <int>
#> 1 Afghanistan 745 2666
#> 2 Brazil 37737 80488
#> 3 China 212258 213766

table4b
#> # A tibble: 3 × 3
#>   country `1999` `2000`
#> * <chr> <int> <int>
#> 1 Afghanistan 19987071 20595360
#> 2 Brazil 172006362 174504898
#> 3 China 1272915272 1280428583

Your Turn #4

What needs to be done to calculate the rate?
Hint: The info is split between two tables. Would it help if each table was in a different form?
1.2 Why Tidy Data?

- Tidy data (in form of a data frame) is usually the best form for analysis
  - some exceptions are for modeling (e.g., matrix manipulations and algorithms)
- For presentation of data (e.g., in tables), non-tidy form can often do better
- the functions in tidyr usually allow us to covert from non-tidy to tidy for analysis and also from tidy to non-tidy for presentation

2 Main tidyr functions

<table>
<thead>
<tr>
<th>function</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>spread()</td>
<td>Spreads a pair of key:value columns into a set of tidy columns</td>
</tr>
<tr>
<td>gather()</td>
<td>Gather takes multiple columns and collapses into key-value pairs, duplicating all other columns as needed. You use gather() when you notice that you have columns that are not variables</td>
</tr>
<tr>
<td>separate()</td>
<td>turns a single character column into multiple columns</td>
</tr>
<tr>
<td>unite()</td>
<td>paste together multiple columns into one (reverse of separate())</td>
</tr>
</tbody>
</table>

Tidy data is often the form we want for further analysis. For example, here are some basic plots that would be difficult to make in the untidy versions.

```r
tidy_table = table1 %>% mutate(rate = cases / population)

#- line plot
ggplot(tidy_table, aes(x = as.factor(year), y = rate, color = country, group = country)) + geom_line() + geom_point(aes(size = population)) + xlab("year")

#- bar plot
ggplot(tidy_table, aes(x = as.factor(year), y = rate, fill = country)) + geom_bar(stat = "identity", position = "dodge") + xlab("year")
```
One exception is if we want to facet (or group) by type column(s). Then table2 is better.

```r
ggplot(table2, aes(x=country, y=count, fill=as.factor(year))) + geom_bar(stat="identity", position="fill") + facet_wrap(~type)
```

The `tidyr` package provides functionality to convert to and from tidy data, which can greatly speed up analysis and help structure your thinking.

### 2.1 `gather()` into long form

The `gather()` function collects a set of column names and places them into a single “key” column. It also collects the field of cells associated with those columns and places them into a single value column.

In the example from 12.3.1 R4DS, `table4a (cases)` and `table4b (population)` are gathered into two columns: year and value.

```r
table4a
#> # A tibble: 3 × 3
#>    country `1999` `2000`
#> * <chr> <int> <int>
#> 1 Afghanistan 745 2666
#> 2 Brazil 37737 80488
#> 3 China 212258 213766
(tidy4a = gather(table4a, key="year", value="cases", 2:3))
#> # A tibble: 6 × 3
#>    country year cases
#>    <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
```
The function is:

```r
gather(
  data = <data frame>,
  key = <name of new key column>,
  value = <name of new value column>,
  ... = <specification of columns to gather>,
  <optional.args>)
```

where the specification of columns could be by name, index, or any method allowed by the `?dplyr::select()` function.

### Your Turn #5

1. For tidying `table4`, how were the columns to gather specified?
2. What would be an alternative way to specify them?
3. Tidy up `table4b`.
4. Calculate the disease rate.

## 2.2 spread() into wide form

The `spread()` function is the opposite of `gather()` and converts two columns (one key, one value) into a set of columns (one new column for every unique key value).

The `table2` can be `spread` into a tidy format

```
> # A tibble: 12 × 4
> #  country year type count
> #   <chr> <int> <chr> <int>
> # 1 Afghanistan 1999 cases 745
count
> 2 Afghanistan 1999 population 19987071
> 3 Afghanistan 2000 cases 2666
> 4 Afghanistan 2000 population 20595360
> 5 Brazil 1999 cases 37737
> 6 Brazil 1999 population 212258
> 7 Brazil 2000 cases 80488
> 8 Brazil 2000 population 213766
> 9 China 1999 cases 212258
>10 China 1999 population 172006362
>11 China 2000 cases 213766
>12 China 2000 population 172006362
```
7 Brazil 2000 cases 80488
8 Brazil 2000 population 174504898
9 China 1999 cases 212258
10 China 1999 population 1272915272
11 China 2000 cases 213766
12 China 2000 population 1280428583

unique(table2$type)
> [1] "cases"    "population"

spread(table2, key=type, value=count)
> # A tibble: 6 × 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

Notice that 2 extra columns were added (cases and population) according the unique values in type.

Figure 2: Spreading table2 makes it tidy.
The function is:

```r
spread(
  data = <data frame>,
  key = <unquoted name of key column>,
  value = <unquoted name of value column>,
  fill = <the value to replace NA's>,
  convert = <logical. Convert (parse) the new columns.>
  <optional.args>)
```

### 2.3 separate()

The `separate()` function pulls apart one column into multiple columns, by splitting wherever the separator (`sep=`) character appears.

In `table3`, the *equation* for the rate is given, but not the calculated value. One approach is to use the `separate()` function from `tidyr` to separate this one column into two which gives us `table1`.

```
separate(table3, rate, into=c("cases", "population"), sep="/", convert=TRUE) %>%
  mutate(rate=cases/population)
```

Notice that we used the optional arguments `sep="/"` and `convert=TRUE`.

```
separate(
  data = <data frame>,
  col = <unquoted name column to separate>,
  into = <names of new columns (character vector)>,
  sep = <the separator>,
  remove = <logical. remove original column?>
  convert = <logical. Convert (parse) the new columns.>
  <optional.args>)
```

The `separate()` function is also useful for extracting date and time elements.
Consider the following data that has date and event information.

```r
url = "https://raw.githubusercontent.com/mdporter/ST597/master/data/date-event.csv"
(df = read_csv(url))
#> Parsed with column specification:
#> cols(
#>   date = col_date(format = ""),
#>   event = col_character()
#> )
#> # A tibble: 100 × 2
#>   date   event
#>   <date> <chr>
#> 1 2016-01-16 D
#> 2 2016-03-29 D
#> 3 2016-01-17 B
#> 4 2016-05-16 A
#> 5 2016-04-13 C
#> 6 2016-03-29 B
#> 7 2016-01-14 A
#> 8 2016-01-25 C
#> 9 2016-04-18 D
#> 10 2016-01-25 A
#> # ... with 90 more rows
```

We want to know the distribution of event type by day of the month. One way to get this information is with the `separate()` function. The `separate()` function will split up a character column, according to some pattern, into multiple new columns. It essentially does a `str_split` and then adds the new columns into the data frame.

Here is the result with default settings
Notice a few things:

- The original `date` column was removed. We can keep it in with the argument `remove=FALSE`
- The new columns are still `character` vectors. If we want them to be numeric, we can set `convert=TRUE`, which attempt to convert the columns to the appropriate type.

This produces the following:

```r
separate(df, col=date, into=c("year", "month", "day"), sep="-", remove=FALSE, convert=TRUE)
```

If we want counts per day:

```r
df %>%
  separate(col=date, into=c("year", "month", "day"), sep="-") %>%
  count(day, event)
```

Now use `spread()` to get into table form for easier display:

```r
df %>%
  separate(col=date, into=c("year", "month", "day"), sep="-", remove=FALSE, convert=TRUE) %>%
  count(day, event) %>%
  spread(key=event, value=n, fill=0)
```

### 2.4 unite()

The `unite()` function is the opposite of `separate()` and will recombine multiple columns.

```r
df %>%
  separate(col=date, into=c("year", "month", "day"), sep="-", remove=FALSE, convert=TRUE) %>%
  unite(col="USdate", month, day, year, sep="/")
```

```r
# A tibble: 100 × 3
#> date  USdate  event
#> * <date> <chr> <chr>
#> 1 2016-01-16 1/16/2016 D
#> 2 2016-03-29 3/29/2016 D
```
3 Missing Data

3.1 Missing Values

Changing the representation of a dataset brings up an important subtlety of missing values. Surprisingly, a value can be missing in one of two possible ways:

- **Explicitly**, i.e. flagged with NA.
- **Implicitly**, i.e. simply not present in the data.

In the previous example, there is some implicit missing data. What is missing, and what should be the value of the missing data?

```r
df %>%
  separate(col=date, into=c("year", "month", "day"), sep="-",
          remove=FALSE, convert=TRUE) %>%
  count(day, event) %>%
  arrange(day, event)
```

```
# Source: local data frame [66 x 3]
# Groups: day [29]
#> # ... with 56 more rows
```

Now to fill in missing days with `complete()`

```r
df %>%
  separate(col=date, into=c("year", "month", "day"), sep="-",
          remove=FALSE, convert=TRUE) %>%
  count(day, event) %>%
  complete(day=1:31, event=c('A', 'B', 'C', 'D'), fill=list(n=0L))
```

```
# Source: local data frame [3,596 x 3]
```
3.1.1 Functions to know

- `complete()`
- `fill()`

4 Your Turn

4.1 Problem 1: Tornado

The US Storm Prediction Center make severe weather data available from the website [http://www.spc.noaa.gov/wcm/#data](http://www.spc.noaa.gov/wcm/#data). This data is used by insurance companies to help with their claims evaluation and forecasting. A description of the data can be found [http://www.spc.noaa.gov/wcm/data/SPC_severe_database_description.pdf](http://www.spc.noaa.gov/wcm/data/SPC_severe_database_description.pdf).

Use the tornado event data ([https://raw.githubusercontent.com/mdporter/ST597/master/data/tornado.csv](https://raw.githubusercontent.com/mdporter/ST597/master/data/tornado.csv)), to calculate the number of tornadoes by `year` and Fujita score (F) and then use `spread()` to convert the results to a table. The final result should look like this:

<table>
<thead>
<tr>
<th>yr</th>
<th>F0</th>
<th>F1</th>
<th>F2</th>
<th>F3</th>
<th>F4</th>
<th>F5</th>
</tr>
</thead>
<tbody>
<tr>
<td>2007</td>
<td>681</td>
<td>306</td>
<td>97</td>
<td>27</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>2008</td>
<td>997</td>
<td>515</td>
<td>158</td>
<td>56</td>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>2009</td>
<td>709</td>
<td>355</td>
<td>94</td>
<td>21</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>2010</td>
<td>776</td>
<td>351</td>
<td>129</td>
<td>42</td>
<td>17</td>
<td>0</td>
</tr>
<tr>
<td>2011</td>
<td>821</td>
<td>638</td>
<td>212</td>
<td>72</td>
<td>25</td>
<td>9</td>
</tr>
<tr>
<td>2012</td>
<td>577</td>
<td>242</td>
<td>100</td>
<td>32</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>2013</td>
<td>508</td>
<td>314</td>
<td>86</td>
<td>22</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>2014</td>
<td>478</td>
<td>325</td>
<td>76</td>
<td>20</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>2015</td>
<td>704</td>
<td>415</td>
<td>69</td>
<td>19</td>
<td>5</td>
<td>0</td>
</tr>
</tbody>
</table>
4.2 Problem 2: Time of Day

Your Turn #7: Time-of-Day

The goal of this task is to plot the estimated density of the time when tornadoes occur. The time column in the tornado data gives the time-of-day (24 hour clock, central time zone) when the tornado occurred. Ignoring the time zone issue, create a density plot of the fractional hour when tornadoes occur.

a. Use the separate() function to create three new columns (hour, min, sec) from the time column.
b. Add another column, named time2, that gives the fractional number of hours that a tornado occurred.
c. Generate a density plot of time2.

4.3 Problem 3: Pew Survey

Your Turn #8: Pew Survey

Results from a pew survey were presented in a non-tidy (table) format where the column headers are values instead of variable names. That is, the data are in wide formate, and we desire the long format. The data can be found https://github.com/hadley/tidyr/blob/master/vignettes/pew.csv.

a. Load the data into R. The url to the raw data is https://raw.githubusercontent.com/hadley/tidyr/master/vignettes/pew.csv
b. What are the three variables in the data?
c. Use gather() to make the data tidy (i.e., long format, with one column for each variable).
d. Make a graphic from the long data comparing the distribution of income between Catholic and Evangelical Prot.

5 Other functions in tidy package

<table>
<thead>
<tr>
<th>function</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>replace_na()</td>
<td>Replace NA’s with specific values</td>
</tr>
<tr>
<td>fill()</td>
<td>Fills missing values in using the previous entry. This is useful in the common output format where values are not repeated, they’re recorded each time they change.</td>
</tr>
<tr>
<td>function</td>
<td>description</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>extract()</td>
<td>check out separate(), but allows different patterns</td>
</tr>
<tr>
<td>expand()</td>
<td>convert <em>implicit</em> missing values (i.e., missing rows) to <em>explicit</em> missing values (include rows with NA)</td>
</tr>
<tr>
<td>complete()</td>
<td>good for tables (filling in missing with 0 counts)</td>
</tr>
</tbody>
</table>