Introduction to Tidy Data in R

Data sets are stored in tabular format and there are many possible ways to organize tabular data. Some organizational schemes are designed to be easily read on the page (or screen), while others are designed to be easily used in analysis. In this tutorial, we focus on how a data set should be formatted for analysis in R.

Make sure that the tidyverse is loaded

```
library(tidyverse)
```

and load in the example data sets

```
# You can paste the urls of the data sets in place of the file names
UBSprices <- read.csv("data/UBSprices.csv", as.is = TRUE)
polls <- read.csv("data/rcp-polls.csv", na.strings = "--", as.is = TRUE)
airlines <- read.csv("data/airline-safety.csv", as.is = TRUE)
```

1. Definition of a tidy data set

In R, it is easiest to work with data that follow five basic rules:

1. Every **variable** is stored in its own **column**.
2. Every observation is stored in its own **row**—that is, every row corresponds to a single **case**.
3. Each **value** of a variable is stored in a **cell** of the table.
4. Values should not contain units. Rather, units should be specified in the supporting documentation for the data set, often called a *codebook*.
5. There should be no extraneous information (footnotes, table titles, etc.).

A data set satisfying these rules is said to be **tidy**, a term popularized by Hadley Wickham.

Remark: Most of the time data that violate rules 4 and 5 are obviously not tidy, and there are easy ways to exclude footnotes and titles in spreadsheets by simply omitting the offending rows. This tutorial focuses on the "sneakier" form of untidiness that violates at least one of the first three rules.

This tutorial will describe the following `tidyr` commands, which can be thought of as verbs for tidying data:

Command	Meaning
`gather`	collapses multiple columns into two columns
`spread`	creates multiple columns from two columns
`separate`	splits compound variables into individual columns

2. Tidying longitudinal data (`gather`)

UBS is an international bank that reports prices of various staples in major cities every three years. The data set in `UBSprices` data set contains prices of a 1 kg bag of rice in 2009 and 2003 in major world cities. The data set was extracted from the `alr4` R package.

```
head(UBSprices)
```

```
##        city rice2003 rice2009
## 1 Amsterdam        9       11
## 2    Athens       19       27
## 3  Auckland        9       13
## 4   Bangkok       25       27
```

```
## 5 Barcelona       10        8
## 6    Berlin       16       17
```

This data set is not tidy because each row contains two cases: the city in 2003 and the city in 2009. Additionally, the column names 2003 and 2009 contain the year, which should be the value of a variable. In order to tidy these data, we need to

1. Reorganize the data so that each row corresponds to a city in a specific year.
2. Create a single variable for the price of rice.
3. Add a variable for year.

To do this, we will use the `gather` function in the `tidyr` package. `gather` collapses multiple columns into two columns: a **key** column and a **value** column. The **key** will be the new variable containing the old column names and the **value** will contain the information recorded in the cells of the collapsed columns.

In our example, we want to collapse `rice2003` and `rice2009` into the key-value pair `year` and `price`. To do this, we use the following command:

```
tidy_ubs <- gather(data = UBSprices, key = year, value = price, rice2003, rice2009)
head(tidy_ubs)
```

```
##        city     year price
## 1 Amsterdam rice2003     9
## 2    Athens rice2003    19
## 3  Auckland rice2003     9
## 4   Bangkok rice2003    25
## 5 Barcelona rice2003    10
## 6    Berlin rice2003    16
```

Remarks

- The first argument passed to `gather` should be the data frame being tidied. This is true for all of the `tidyr` functions we discuss in this tutorial.
- After specifying the data frame, the next two arguments specify the column names you wish to give to two new columns. One column is called the **key** and the other is called the **values**.
- After the first three arguments, specify the columns that you wish to collapse, separated by commas. Notice that the original column names are now listed in the key column and the original cell values are now all in one column.

Questions:

1) How are the number of rows adjusted by using the `gather` command? Use the `dim(UBSprices)` command to determine how many rows are in the **UBSprices** data set and `dim(tidy_ubs)` to determine how many are in the **tidy_ubs** data set).

2) How many rows would there be if used the gather command and the original **UBSprices** data set had five columns of years: `rice2003`, `rice2006`, `rice2009`, `rice2012`, and `rice2015`?

Finally, we need to modify the year column by removing the word rice from each cell. To do this, we can use the `extract_numeric` function in the `tidyr` package. We now have a data set that we can call tidy.

```
tidy_ubs$year <- extract_numeric(tidy_ubs$year)
```

```
## extract_numeric() is deprecated: please use readr::parse_number() instead
```

```
head(tidy_ubs)
```

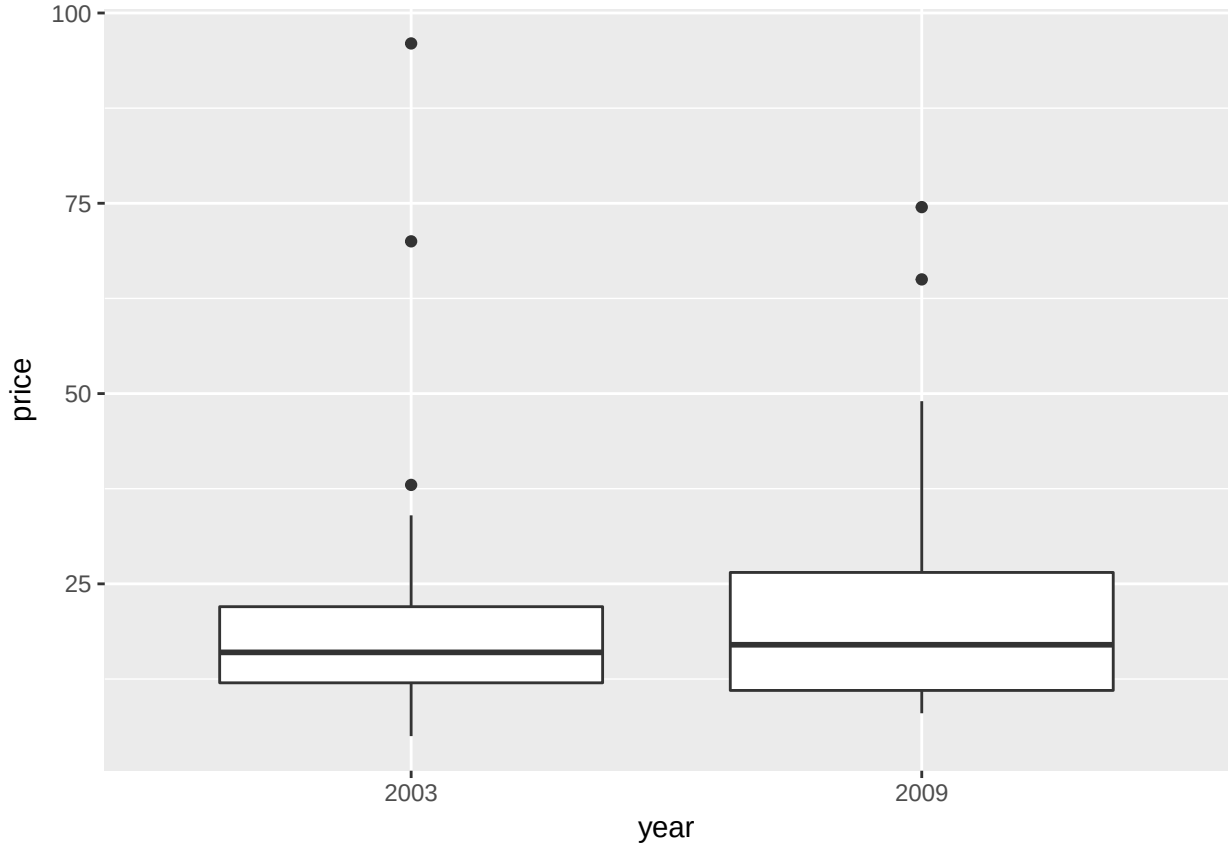
```
##        city year price
## 1 Amsterdam 2003     9
## 2    Athens 2003    19
## 3  Auckland 2003     9
```

```
## 4   Bangkok 2003    25
## 5 Barcelona 2003    10
## 6    Berlin 2003    16
```

Remark

This data set started in a relatively tidy form, so it may be difficult to see the benefit of tidying it. Tidy data is typically required for summarizing and plotting data in R. For example, consider making a side-by-side boxplot using `ggplot2` (see the tutorial IntroToGgplot).

```
tidy_ubs %>%
  ggplot(aes(x = factor(year), y = price)) +
  geom_boxplot() +
  labs(x = "year")
```



This was straightforward since **tidy_ubs** was already tidy, but would have required extra manipulation in the original format.

3. Tidying pollster data (`separate + gather`)

The **polls** data set contains the results of various presidential polls conducted during July 2016, and was scraped from RealClear Politics.

```
polls
```

```
##                     Poll        Date  Sample MoE Clinton..D. Trump..R.
## 1              Monmouth 7/14 - 7/16  688 LV 3.7          45        43
## 2               CNN/ORC 7/13 - 7/16  872 RV 3.5          42        37
## 3     ABC News/Wash Post 7/11 - 7/14  816 RV 4.0          42        38
```

```
## 4 NBC News/Wall St. Jrnl  7/9 - 7/13 1000 RV 3.1          41        35
## 5       Economist/YouGov  7/9 - 7/11  932 RV 4.5          40        37
## 6   Associated Press-GfK  7/7 - 7/11  837 RV  NA          40        36
## 7       McClatchy/Marist   7/5 - 7/9 1053 RV 3.0          40        35
##   Johnson..L. Stein..G.
## 1           5         1
## 2          13         5
## 3           8         5
## 4          11         6
## 5           5         2
## 6           6         2
## 7          10         5
```

Here, the data set is not tidy because

- The `Date` column contains both the beginning and end dates. These should be stored in separate columns.
- The `Sample` column contains two variables: the number of people in the sample and the population that was sampled (likely voters or registered voters). These should be stored in separate columns.
- The last four column names are values of `candidate` and `party` variables, which should be stored in their own columns.

To break a single character column into multiple new columns we use the `separate` function in the `tidyr` package.

To begin, let's break the `Date` column into `Begin` and `End` columns:

```
tidy_polls <- separate(data = polls, col = Date, into = c("Begin", "End"), sep = " - ")
tidy_polls
```

```
##                    Poll Begin  End  Sample MoE Clinton..D. Trump..R.
## 1              Monmouth  7/14 7/16  688 LV 3.7          45        43
## 2               CNN/ORC  7/13 7/16  872 RV 3.5          42        37
## 3     ABC News/Wash Post  7/11 7/14  816 RV 4.0          42        38
## 4 NBC News/Wall St. Jrnl   7/9 7/13 1000 RV 3.1          41        35
## 5       Economist/YouGov   7/9 7/11  932 RV 4.5          40        37
## 6   Associated Press-GfK   7/7 7/11  837 RV  NA          40        36
## 7       McClatchy/Marist   7/5  7/9 1053 RV 3.0          40        35
##   Johnson..L. Stein..G.
## 1           5         1
## 2          13         5
## 3           8         5
## 4          11         6
## 5           5         2
## 6           6         2
## 7          10         5
```

Remarks

- The second argument, `col`, specifies the name of the column to be split.
- The third argument, `into`, specifies the names of the new columns. Note that since these are specific column names we are creating, they should be given in quotes.
- R will try to guess how the values should be separated by searching for non-alphanumeric values; however, if there are multiple non-alphanumeric values this may fail. In this example, if we did not specify that `sep = " - "`, then R would erroneously use \ as the separator. To manually specify the separator between columns we can place the character(s) in quotes.
- In `sep = " - "`, the spaces around `-` avoid excess whitespace in the resulting cell values.

We also need to separate the `Sample` column into `size` and `population` columns.

```
tidy_polls <- separate(data = tidy_polls, col = Sample, into = c("size", "population"), sep = " ")
tidy_polls
```

```
##                     Poll Begin  End size population MoE Clinton..D.
## 1               Monmouth  7/14 7/16  688         LV 3.7          45
## 2                CNN/ORC  7/13 7/16  872         RV 3.5          42
## 3     ABC News/Wash Post  7/11 7/14  816         RV 4.0          42
## 4 NBC News/Wall St. Jrnl   7/9 7/13 1000         RV 3.1          41
## 5       Economist/YouGov   7/9 7/11  932         RV 4.5          40
## 6   Associated Press-GfK   7/7 7/11  837         RV  NA          40
## 7       McClatchy/Marist   7/5  7/9 1053         RV 3.0          40
##   Trump..R. Johnson..L. Stein..G.
## 1        43           5         1
## 2        37          13         5
## 3        38           8         5
## 4        35          11         6
## 5        37           5         2
## 6        36           6         2
## 7        35          10         5
```

Next, we need to `gather` the last four columns into a `candidate` variable.

```
tidy_polls <- gather(data = tidy_polls, key = candidate, value = percentage, 7:10)
head(tidy_polls)
```

```
##                     Poll Begin  End size population MoE   candidate
## 1               Monmouth  7/14 7/16  688         LV 3.7 Clinton..D.
## 2                CNN/ORC  7/13 7/16  872         RV 3.5 Clinton..D.
## 3     ABC News/Wash Post  7/11 7/14  816         RV 4.0 Clinton..D.
## 4 NBC News/Wall St. Jrnl   7/9 7/13 1000         RV 3.1 Clinton..D.
## 5       Economist/YouGov   7/9 7/11  932         RV 4.5 Clinton..D.
## 6   Associated Press-GfK   7/7 7/11  837         RV  NA Clinton..D.
##   percentage
## 1         45
## 2         42
## 3         42
## 4         41
## 5         40
## 6         40
```

Notice that instead of writing out the column names (`Clinton..D.`, `Trump..R.`, etc.) we can simply specify the column numbers—here `7:10` specifies that we are gathering columns 7 through 10.

Finally, we need to `separate` the candidate names from the political party.

```
tidy_polls <- separate(tidy_polls, candidate, into= c("candidate", "party"))
```

```
## Warning: Too many values at 28 locations: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
## 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, ...
```

```
head(tidy_polls)
```

```
##                     Poll Begin  End size population MoE candidate party
## 1               Monmouth  7/14 7/16  688         LV 3.7   Clinton     D
## 2                CNN/ORC  7/13 7/16  872         RV 3.5   Clinton     D
## 3     ABC News/Wash Post  7/11 7/14  816         RV 4.0   Clinton     D
```

```
## 4 NBC News/Wall St. Jrnl   7/9 7/13 1000         RV 3.1   Clinton     D
## 5       Economist/YouGov   7/9 7/11  932         RV 4.5   Clinton     D
## 6   Associated Press-GfK   7/7 7/11  837         RV  NA   Clinton     D
##   percentage
## 1         45
## 2         42
## 3         42
## 4         41
## 5         40
## 6         40
```

Remark

In the last command we let R guess which separator to use. This worked, but resulted in a warning message—we're lucky that it worked! There are many situations where the separator is too complex for R to guess correctly and it cannot be specified using a simple character in quotes. In such cases we need to use regular expressions (see the tutorial IntroStrings) to aid our data tidying, but that's a topic for another tutorial. The important thing to note here is that you should always check that `separate` worked as you expected, don't blindly trust it!

4. Tidying crash data (`gather + separate + spread`)

The **airlines** data set contains the raw data behind the article Should Travelers Avoid Flying Airlines That Have Had Crashes in the Past? that appeared on fivethirtyeight.com.

```
head(airlines)
```

```
##                 airline avail_seat_km_per_week incidents.1985_1999
## 1            Aer Lingus              320906734                   2
## 2             Aeroflot*             1197672318                  76
## 3 Aerolineas Argentinas              385803648                   6
## 4           Aeromexico*              596871813                   3
## 5            Air Canada             1865253802                   2
## 6            Air France             3004002661                  14
##   fatal_accidents.1985_1999 fatalities.1985_1999 incidents.2000_2014
## 1                         0                    0                   0
## 2                        14                  128                   6
## 3                         0                    0                   1
## 4                         1                   64                   5
## 5                         0                    0                   2
## 6                         4                   79                   6
##   fatal_accidents.2000_2014 fatalities.2000_2014
## 1                         0                    0
## 2                         1                   88
## 3                         0                    0
## 4                         0                    0
## 5                         0                    0
## 6                         2                  337
```

In this example, a case is best described as an airline in a specific time frame, so these data are not tidy because each case is not its own row. Additionally, the last six column names contain the time frame, which is a value. In order to tidy this data set we must

- have rows corresponding to airlines in a specific time frame,
- create a `years` column to specify the time frame,
- and create columns for each type of accident: `incidents`, `fatal_accidents`, and `fatalities`.

First, we `gather` the last six columns into a common `accidents` column. This will allow us to easily create the `years` column.

```
tidy_airlines <- gather(airlines, key = accidents, value = count, 3:8)
head(tidy_airlines)
```

```
##                 airline avail_seat_km_per_week           accidents count
## 1            Aer Lingus              320906734 incidents.1985_1999     2
## 2             Aeroflot*             1197672318 incidents.1985_1999    76
## 3 Aerolineas Argentinas              385803648 incidents.1985_1999     6
## 4           Aeromexico*              596871813 incidents.1985_1999     3
## 5            Air Canada             1865253802 incidents.1985_1999     2
## 6            Air France             3004002661 incidents.1985_1999    14
```

Next, we `separate` the values of the new `accidents` column into `var` (short for variable) and `years`. The default guessing scheme fails here, so we must specify `sep = "[.]"` to denote that the period is the separator. (If you want to learn more about why we need brackets around the period you need to delve into regular expressions.)

```
tidy_airlines <- separate(tidy_airlines, accidents, into = c("var", "years"), sep = "[.]")
head(tidy_airlines)
```

```
##                 airline avail_seat_km_per_week       var     years count
## 1            Aer Lingus              320906734 incidents 1985_1999     2
## 2             Aeroflot*             1197672318 incidents 1985_1999    76
## 3 Aerolineas Argentinas              385803648 incidents 1985_1999     6
## 4           Aeromexico*              596871813 incidents 1985_1999     3
## 5            Air Canada             1865253802 incidents 1985_1999     2
## 6            Air France             3004002661 incidents 1985_1999    14
```

Finally, we need to ensure that each row corresponds to a case. (Don't worry, this will also make each column a variable!) Currently, there are six rows for each airline: one for each `var` in each time frame. To solve this problem, we need to `spread` out the `var` column so that each variable has its own column.

```
tidy_airlines <- spread(data = tidy_airlines, key = var, value = count)
head(tidy_airlines)
```

```
##                 airline avail_seat_km_per_week     years fatal_accidents
## 1            Aer Lingus              320906734 1985_1999               0
## 2            Aer Lingus              320906734 2000_2014               0
## 3             Aeroflot*             1197672318 1985_1999              14
## 4             Aeroflot*             1197672318 2000_2014               1
## 5 Aerolineas Argentinas              385803648 1985_1999               0
## 6 Aerolineas Argentinas              385803648 2000_2014               0
##   fatalities incidents
## 1          0         2
## 2          0         0
## 3        128        76
## 4         88         6
## 5          0         6
## 6          0         1
```

Remark

Notice that the first argument given to `spread` is the data frame, followed by the key-value pair. The key is the name of the column whose values will be used as column headings and the value is the name of the column whose values will populate the cells of the new columns. In this example, we use `var` as the key and populate the cells with the `count`.

5. On Your Own

1. The file `daily_show_guests.csv` contains information on every guest Jon Stewart ever had on *The Daily Show*. (Source: https://github.com/fivethirtyeight/data/tree/master/daily-show-guests)

   Briefly explain why this is a tidy data set.

2. The file `under5mortality.csv` contains the child mortality rate per 1,000 children born for each country from 1800 to 2015. (Source: https://www.gapminder.org/data/)

   a. Briefly describe why it is not considered to be tidy data and what changes need to be made to tidy it.
   b. Use `gather` to create a tidy data set with columns `country`, `year` and `mortality`. Use `extract_numeric` to ensure that the `year` column is numeric.

3. The file `mlb2016.csv` contains the salary information presented by *USA Today* for all 862 players in Major League Baseball. (Source: http://www.usatoday.com/sports/mlb/salaries/2016/player/all/)

   a. Briefly describe why it is not considered to be tidy data and what changes need to be made to tidy it.
   b. Use `separate` and `extract_numeric` to tidy this data set.

4. The data set in `UBSprices2.csv` contains prices of a 1 kg bag of rice, a 1 kg loaf of bread, and a Big Mac in major world cities in 2009 and 2003.

   a. Briefly describe why it is not considered to be tidy data and what changes need to be made to tidy it.
   b. Use `gather` and `separate` to tidy this data set. (Hint: In addition to accepting characters, the `sep` argument can also be set to the position at which to create a split. For example, if we specify `sep = 2`, then the character strings will be split into the first two characters and the remaining characters. In this example the type of commodity is of variable length, so it is easiest to count from the left. This is specified by using a negative value.)

6. Additional Resources

- RStudio's data wrangling cheat sheet provides a nice summary of how to reshape data sets and a quick reminder of the definition of tidy data.
- The `tidyr` vignette provides additional examples and elaborates on the capabilities of the `tidyr` package.