tidy-data-tutorial

Brandon Loudermilk

Tidy Tutorial

> "Tidy datasets are all alike but every messy dataset is messy in its own way." – Hadley Wickham

Introduction

Statistical data sets consist of tabular data organized by rows and columns, like the tables of a traditional database. The data set below stores information about students and their scores on 3 tests.

```
(df1 <- data.frame(student = c("Jim Smith", "Fred Johnson", "Steve Jones"),
                  test1 = c(85, 81, 92),
                  test2 = c(87, 84, 95),
                  test3 = c(87, 85, 91)))
```

```
##        student test1 test2 test3
## 1    Jim Smith    85    87    87
## 2 Fred Johnson    81    84    85
## 3  Steve Jones    92    95    91
```

Importantly, however, there exist multiple ways of storing the *same* underlying data. Below, we display the same data in a different form – columns and rows are transposed, such that each column represents a student and each row a test.

```
(df2 <- data.frame("Jim Smith" = c(85, 87, 87),
                   "Fred Johnson" = c(81,84,85),
                   "Steve Jones" = c(87, 85, 91),
                   row.names = c("book1", "book2", "book3")))
```

```
##       Jim.Smith Fred.Johnson Steve.Jones
## book1        85           81          87
## book2        87           84          85
## book3        87           85          91
```

In general, a data set is just a collection of values (quantitative/qualitative) where each value belongs to both a variable and an observation. To better illustrate the relationship among values, variables, and observations, we display the same data in third table.

```
(df3 <- data.frame(student = c("Jim Smith", "Jim Smith", "Jim Smith",
                               "Fred Johnson", "Fred Johnson", "Fred Johnson",
                               "Steve Jones", "Steve Jones", "Steve Jones"),
                   test_number = c(1,2,3,1,2,3,1,2,3),
                   score = c(85, 87, 87, 81, 84, 85, 92, 95, 91)))
```

```
##        student test_number score
## 1    Jim Smith           1    85
## 2    Jim Smith           2    87
## 3    Jim Smith           3    87
## 4 Fred Johnson           1    81
## 5 Fred Johnson           2    84
## 6 Fred Johnson           3    85
## 7  Steve Jones           1    92
## 8  Steve Jones           2    95
## 9  Steve Jones           3    91
```

The underlying data behind all three of these surface forms consists of 27 values representing 3 variables and 9 observations. The variables are: **student** with three possible values (Jim, Fred, Steve) **test_number** with three possible values (1, 2, 3) **score** with nine values (85, 87, 87, 81, 84, 85, 92, 95, 91)

The transformed data above is an example of **tidy data** because it adheres to the following conventions:

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

Any data set that violates these conventions is an "untidy" data set. Untidy data sets come in several different flavors (Wickham):

1. Column headers are values, not variable names.
2. Multiple variables are 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.

In the following sections we illustrate how to use functions in the {tidyr} and {dplyr} packages to transform untidy data into tidy data.

Common Transformations

Gather

Problem: Column headers are values, not variable names **Solution:** Gathering (`gather()`) makes a wide dataset long by gathering together columns and replacing them with two new columns. Specifically, when you have columns whose headers are values rather than variable names, you can use `gather()` to replace the offending columns with: a *key* column that holds the headers of the original column names, and a *value* column for the corresponding data value.

Let's illustrate `gather()` by examing some data on religion and salary.

```
library(tidyr) # always load both tidyr & dplyr
library(dplyr) # they are meany to work together

file_name <- "pew.csv"
pew <- tbl_df(read.csv(file_name, stringsAsFactors = FALSE, check.names = FALSE))

head(pew)
```

```
## Source: local data frame [6 x 11]
##
##          ï»¿religion <$10k $10-20k $20-30k $30-40k $40-50k $50-75k
##                (chr) (int)   (int)   (int)   (int)   (int)   (int)
## 1           Agnostic    27      34      60      81      76     137
## 2            Atheist    12      27      37      52      35      70
## 3           Buddhist    27      21      30      34      33      58
## 4           Catholic   418     617     732     670     638    1116
## 5 Don't know/refused    15      14      15      11      10      35
## 6   Evangelical Prot   575     869    1064     982     881    1486
## Variables not shown: $75-100k (int), $100-150k (int), >150k (int), Don't
##   know/refused (int)
```

The data above shows the distribution of individuals by religion and income bracket. Note that columns 2:11 contain headers that are values (e.g., $10k-20k), not variable names (e.g., income). We need to transform these multiple columns into two columns that map keys to values. Function `gather()` accepts four arguments: *data* for the underlying `data.frame`; *key* - name of new column whose values are the headers in the originnal data; *value* - name of the new column that contains the values; and ... - the triple dots arguments selects the target columns to transform.

```
## dplyr::gather()
## data - data.frame or tbl_df
## key - name of new key column
## value - name of the new value column
## ... - columns to include in transformation (these cols are removed)

gather(data = pew, key = salary, value = count, ... = 2:11)
```

```
## Source: local data frame [180 x 3]
##
##                ï»¿religion salary count
##                      (chr)  (chr) (int)
## 1                 Agnostic  <$10k    27
## 2                  Atheist  <$10k    12
## 3                 Buddhist  <$10k    27
## 4                 Catholic  <$10k   418
## 5       Don't know/refused  <$10k    15
## 6         Evangelical Prot  <$10k   575
## 7                    Hindu  <$10k     1
## 8  Historically Black Prot  <$10k   228
## 9        Jehovah's Witness  <$10k    20
## 10                  Jewish  <$10k    19
## ..                     ...    ...   ...
```

Above, we replaced 10 columns with two new columns: `$salary`, which stores income bracket, and `$count`, a tally of frequency. This new data form is tidy because each column represents a variable and each row represents an observation, in this case a demographic unit corresponding to a combination of religion and income.

Separate

Problem: Multiple variables stored in one column **Solution:** Separate (`separate()`) the single variable column into multiple columns.

```
(df1 <- data.frame(year = c(2001, 2001, 2001, 2001),
                   sex_groupnum = c("m1", "m2", "f1", "f2"),
                   score = c(75, 85, 77, 94)))
```

```
##   year sex_groupnum score
## 1 2001           m1    75
## 2 2001           m2    85
## 3 2001           f1    77
## 4 2001           f2    94
```

The above data contains a column `$sex_groupnum` that represents a combination of gender and group number. We would like to separate this column into two variables, `$sex` and `$groupnum`. Function `separate()` takes the following arguments: *data* - the data.frame, *col* - the column to split, *into* - vector of column names that should be created, and *sep* - how to separate the values in the target column.

```
## data - data.frame
## col - column to be separated
## into - vector of new column names
## sep - separator (regex for character match, or int for position match)
separate(data = df1,
         col = sex_groupnum,
         into = c("sex", "groupnum"),
         sep = 1)
```

```
##   year sex groupnum score
## 1 2001   m        1    75
## 2 2001   m        2    85
## 3 2001   f        1    77
## 4 2001   f        2    94
```

Above we separated column `$sex_groupnum` using a positional separator (sep = 1), which splits values based on character position. The *sep* variable can also accept a regex value for more sophisticated splitting options.

```
(df2 <- data.frame(year = c(2001, 2001, 2001, 2001),
                   sex_groupnum = c("male_1", "male_2", "female_1", "female_2"),
                   score = c(75, 85, 77, 94)))
```

```
##   year sex_groupnum score
## 1 2001       male_1    75
## 2 2001       male_2    85
## 3 2001     female_1    77
## 4 2001     female_2    94
```

```
separate(data = df2,
         col = sex_groupnum,
         into = c("sex", "groupnum"), sep = "_")
```

```
##   year    sex groupnum score
## 1 2001   male        1    75
## 2 2001   male        2    85
## 3 2001 female        1    77
## 4 2001 female        2    94
```

Unite

Problem: Multiple columns contain values that should be a single column. **Solution:** Call `unit()` (inverse of separate) to combine columns into a single column.

```
(df1 <- data.frame(author = c("Chomsky", "Pinker", "Humboldt"),
                   title = c("Syntactic Stuctures",
                             "The Sense of Style",
                             "The Heterogeneity of Language"),
                   century = c(19,20, 18) ,
                   year = c(57, 15, 36)))
```

```
##     author                         title century year
## 1  Chomsky           Syntactic Stuctures      19   57
## 2   Pinker            The Sense of Style      20   15
## 3 Humboldt The Heterogeneity of Language      18   36
```

In the data above, we see that the book's publication year has been split into two columns, one for century and another for year. We need to combine them into a single column called *$publication_year* by calling `unite()`.

```
unite(data = df1, col = pubyr, ... = c(century, year), sep = "")
```

```
##     author                         title pubyr
## 1  Chomsky           Syntactic Stuctures  1957
## 2   Pinker            The Sense of Style  2015
## 3 Humboldt The Heterogeneity of Language  1836
```

Chain Operator

Often your starting data will require several transformations to get it into tidy data form. The following data has two untidy issues: column headers contain values and multiple values are stored in single columns.

```
file_name <- "tb.csv"
tb1 <- read.csv(file_name)
head(tb1)
```

```
##   iso2 year m04 m514 m014 m1524 m2534 m3544 m4554 m5564 m65 mu f04 f514
## 1   AD 1989  NA   NA   NA    NA    NA    NA    NA    NA  NA NA  NA   NA
## 2   AD 1990  NA   NA   NA    NA    NA    NA    NA    NA  NA NA  NA   NA
## 3   AD 1991  NA   NA   NA    NA    NA    NA    NA    NA  NA NA  NA   NA
## 4   AD 1992  NA   NA   NA    NA    NA    NA    NA    NA  NA NA  NA   NA
## 5   AD 1993  NA   NA   NA    NA    NA    NA    NA    NA  NA NA  NA   NA
## 6   AD 1994  NA   NA   NA    NA    NA    NA    NA    NA  NA NA  NA   NA
##   f014 f1524 f2534 f3544 f4554 f5564 f65 fu
## 1   NA    NA    NA    NA    NA    NA  NA NA
## 2   NA    NA    NA    NA    NA    NA  NA NA
## 3   NA    NA    NA    NA    NA    NA  NA NA
## 4   NA    NA    NA    NA    NA    NA  NA NA
## 5   NA    NA    NA    NA    NA    NA  NA NA
## 6   NA    NA    NA    NA    NA    NA  NA NA
```

Let's start by gathering the columns with value headers and replacing them with columns `$gender_age` and `$value`.

```
tb2 <- gather(tb1, gender_age, value, 3:22)
head(tb2)
```

```
##   iso2 year gender_age value
## 1   AD 1989        m04    NA
## 2   AD 1990        m04    NA
## 3   AD 1991        m04    NA
## 4   AD 1992        m04    NA
## 5   AD 1993        m04    NA
## 6   AD 1994        m04    NA
```

Now, we see that the column `$gender_age` needs to be separated into two independent columns `$gender` and `$age`.

```
tb3 <- separate(tb2, col = gender_age, into = c("gender", "age"), sep = 1)
car::some(tb3)
```

```
##        iso2 year gender  age value
## 15563    PA 1982      m  014    NA
## 37146    JO 2001      m 4554    10
## 38166    MV 1985      m 4554    NA
## 45546    TO 1985      m 5564    NA
## 67846    QA 2005      f  514    NA
## 72466    MG 1993      f  014    NA
## 73058    NL 1981      f  014    NA
## 76863    GE 1995      f 1524     8
## 91992    VC 1995      f 3544    NA
## 107963   PG 1991      f   65    NA
```

In the above example, we performed a number of individual transformations, `gather()` and `separate()`, storing the results in temporary variables before passing them to the next function.

Packages {dply} and {tidyr} provide a convenience operator `%>%` that allows use to chain together numerous transformations into a single line of code. This has the added benefit of reducing computation (no need to store data in a temporary variable).

```
tb1 %>%
  gather(key = gender_age, value = value, ... = 3:22, na.rm = TRUE) %>%
  separate(col = gender_age, into = c("gender", "age"), sep = 1)
```

If you inspect the code above you will notice that the syntax differs from our previous examples. Here, rather than explicitly assigning the *data* variable a value (via `gather(data = tb1)`), we use the `%>%` operator to redirect the data variable to the gather function (via `tb1 %>% gather()`). The `%>%` operator is used extensively in {dplyr} and {tidyr} packages and provides a more precise syntax that allows you to chain together functions with the `%>%` operator. Basically `%>%` informs the right hand function to assign the left hand object to the first formal variable of the function. Because all functions in {tidyr} & {dplyr} take *data* as the first formal variable and all of these functions return *data* as output, this enables developers to chain together their logic by using `%>%`.

Spread

Problem: Variables are stored in both rows and columns **Solution:** Spread (`spread()`)column row values into columns. Function `spread()` is the inverse of `gather()`.

```
(df1 <- data.frame(user_id = c(1,1,2,2),
                   type = c("max", "min", "max", "min"),
                   value = c(99,67,88,57)))
```

```
##   user_id type value
## 1       1  max    99
## 2       1  min    67
## 3       2  max    88
## 4       2  min    57
```

The data above stores a unique `user_id` associated with two scores, `min` and `max`. We want to spread these row values into their own column variables.

```
spread(data = df1, key = type, value = value)
```

```
##   user_id max min
## 1       1  99  67
## 2       2  88  57
```

Move data types from single table into multiple tables

Problem: Multiple types in one table **Solution:** Using an assortment of {dplyr} functions, store each type of observational unit in its own table. In the following example, we examine a single table that stores data on the top 100 Billboard songs by year. Each row consists of song information (year, artist, track, and play time) coupled to Billboard ranking data (week on charts, rank on charts, date of first listing on charts). One of the trademarks of data that contains multiple types in one table, is that you find values repeated across rows. In the case of the billboard data, we see that song information gets duplicated for every week the song is on the charts.

```
file_name <- "bb.csv"
df1 <- read.csv(file_name)
head(df1)
```

```
##   X year       artist                   track time week rank       date
## 1 1 2000        2 Pac Baby Don't Cry (Keep... 4:22    1   87 2000-02-26
## 2 2 2000      2Ge+her The Hardest Part Of ... 3:15    1   91 2000-09-02
## 3 3 2000 3 Doors Down              Kryptonite 3:53    1   81 2000-04-08
## 4 4 2000 3 Doors Down                   Loser 4:24    1   76 2000-10-21
## 5 5 2000     504 Boyz           Wobble Wobble 3:35    1   57 2000-04-15
## 6 6 2000         98^0 Give Me Just One Nig... 3:24    1   51 2000-08-19
```

```
df1[df1$track == "Baby Don't Cry (Keep...",]
```

```
##         X year artist                   track time week rank       date
## 1       1 2000  2 Pac Baby Don't Cry (Keep... 4:22    1   87 2000-02-26
## 318   318 2000  2 Pac Baby Don't Cry (Keep... 4:22    2   82 2000-03-04
## 630   630 2000  2 Pac Baby Don't Cry (Keep... 4:22    3   72 2000-03-11
## 937   937 2000  2 Pac Baby Don't Cry (Keep... 4:22    4   77 2000-03-18
## 1237 1237 2000  2 Pac Baby Don't Cry (Keep... 4:22    5   87 2000-03-25
## 1529 1529 2000  2 Pac Baby Don't Cry (Keep... 4:22    6   94 2000-04-01
## 1809 1809 2000  2 Pac Baby Don't Cry (Keep... 4:22    7   99 2000-04-08
```

What we would like to do with this data is to split it into two separate data sets, one that holds song information (songs) and one that holds ranking information (rank).

```
# select() and mutate() are {dplyr} functions
# select columns relating to songs

songs <- df1 %>% select(artist, track, year, time) %>% unique() %>% dplyr::mutate( song_id = row_number

head(songs)
```

```
##         artist                   track year time song_id
## 1        2 Pac Baby Don't Cry (Keep... 2000 4:22       1
## 2      2Ge+her The Hardest Part Of ... 2000 3:15       2
## 3 3 Doors Down              Kryptonite 2000 3:53       3
## 4 3 Doors Down                   Loser 2000 4:24       4
## 5     504 Boyz           Wobble Wobble 2000 3:35       5
## 6         98^0 Give Me Just One Nig... 2000 3:24       6
```

Above, we selected columns relating to song information, added a unique index `$song_id`, and created a `songs` data.frame from the results. Now, we want to extract out the ranking information into its own table and link it back to `songs` via the shared column `$song_id`.

```
rank <- df1 %>% left_join(y = songs, c("artist", "track", "year", "time"))%>%
  select(song_id, date, week, rank) %>%
  arrange(song_id, date)
head(rank)
```

```
##   song_id       date week rank
## 1       1 2000-02-26    1   87
## 2       1 2000-03-04    2   82
## 3       1 2000-03-11    3   72
## 4       1 2000-03-18    4   77
## 5       1 2000-03-25    5   87
## 6       1 2000-04-01    6   94
```

Using a variety of {dplyr} function calls we transformed a single table into two tables each representing their own observational unit. Below, we execute a SQL query on thes new tables.

```
library(sqldf)
join_string <- "select songs.artist, songs.track, rank.week, rank.rank
                from songs
                left join rank
                on songs.song_id = rank.song_id"
res <- sqldf(join_string)
head(res)
```

```
##   artist                   track week rank
## 1  2 Pac Baby Don't Cry (Keep...    1   87
## 2  2 Pac Baby Don't Cry (Keep...    2   82
## 3  2 Pac Baby Don't Cry (Keep...    3   72
## 4  2 Pac Baby Don't Cry (Keep...    4   77
## 5  2 Pac Baby Don't Cry (Keep...    5   87
## 6  2 Pac Baby Don't Cry (Keep...    6   94
```

Move data type in multiple tables into single table

Problem: One type in multiple tables **Solution:** Combine the multiple tables into a single table. Here is the typical sequence to resolv this issue:

1. Read the files into a list of tables.
2. For each table, add a new column that records the original file name (the file name is often the value of an important variable).
3. Combine all tables into a single table.

```
dir("data/")
```

```
## [1] "station_1.txt" "station_2.txt" "station_3.txt"
```

```
temp_df <- read.csv("data/station_1.txt")
head(temp_df)
```

```
##   obs_no temp
## 1      1 94.2
## 2      2 93.4
## 3      3 95.6
## 4      4 91.2
## 5      5 90.9
```

```
library(plyr)
```

```
## -------------------------------------------------------------------------
```

```
## You have loaded plyr after dplyr - this is likely to cause problems.
## If you need functions from both plyr and dplyr, please load plyr first, then dplyr:
## library(plyr); library(dplyr)
```

```
## -------------------------------------------------------------------------
```

```
## 
## Attaching package: 'plyr'
```

```
## The following objects are masked from 'package:dplyr':
## 
##     arrange, count, desc, failwith, id, mutate, rename, summarise,
##     summarize
```

```
paths <- dir("data/", pattern = "\\.txt$", full.names = TRUE)
names(paths) <- basename(paths)
df <- ldply(paths, read.csv, stringsAsFactors = FALSE)
head(df)
```

```
##             .id obs_no temp
## 1 station_1.txt      1 94.2
## 2 station_1.txt      2 93.4
## 3 station_1.txt      3 95.6
## 4 station_1.txt      4 91.2
## 5 station_1.txt      5 90.9
## 6 station_2.txt      1 91.7
```

Putting it all together

In the following use case, you will apply all you have learned about {tidyr} and {dplr} to create a tidy data set. The World Health Organization provided data on tuberculosis rates across the world.

```
df <- read.csv("who.csv")
names(df)
```

```
##  [1] "country"       "iso2"          "iso3"          "year"
##  [5] "new_sp_m014"   "new_sp_m1524"  "new_sp_m2534"  "new_sp_m3544"
##  [9] "new_sp_m4554"  "new_sp_m5564"  "new_sp_m65"    "new_sp_f014"
## [13] "new_sp_f1524"  "new_sp_f2534"  "new_sp_f3544"  "new_sp_f4554"
## [17] "new_sp_f5564"  "new_sp_f65"    "new_sn_m014"   "new_sn_m1524"
## [21] "new_sn_m2534"  "new_sn_m3544"  "new_sn_m4554"  "new_sn_m5564"
## [25] "new_sn_m65"    "new_sn_f014"   "new_sn_f1524"  "new_sn_f2534"
## [29] "new_sn_f3544"  "new_sn_f4554"  "new_sn_f5564"  "new_sn_f65"
## [33] "new_ep_m014"   "new_ep_m1524"  "new_ep_m2534"  "new_ep_m3544"
## [37] "new_ep_m4554"  "new_ep_m5564"  "new_ep_m65"    "new_ep_f014"
## [41] "new_ep_f1524"  "new_ep_f2534"  "new_ep_f3544"  "new_ep_f4554"
## [45] "new_ep_f5564"  "new_ep_f65"    "new_rel_m014"  "new_rel_m1524"
## [49] "new_rel_m2534" "new_rel_m3544" "new_rel_m4554" "new_rel_m5564"
## [53] "new_rel_m65"   "new_rel_f014"  "new_rel_f1524" "new_rel_f2534"
## [57] "new_rel_f3544" "new_rel_f4554" "new_rel_f5564" "new_rel_f65"
```

Here is the structure and conventions for the data set:

For columns 5-60:

1. The prefix “new_” refers to new cases of tuberculosis.
2. The next two letters define the type of tuberculosis:

2-ltr code	Description
rel	relapse
ep	extrapulmonary
sn	smear negative
sp	smear positive

3. The sixth letter describes patient sex: m = male, f = female
4. Remaining numbers describe the age group:

code	group
014	0 to 14
1524	15 to 24
2534	25 to 34
3544	35 to 44
4554	45 to 54
5564	55 to 64
65	65+ years