How to tidy data with Tidyr

R-Wizardry Tutorial

Susan Anderson April 13, 2016

Learning objectives:

—learn what tidy data is and why its useful
—learn key functions in the tidyr package
—manipulate the form and organization of untidy data

Use this tutorial if:

This is a beginner tutorial, and it covers how to structure and organize a dataset for further analyses with R. Specifically, it's how to make a data set "tidy." Some functions are similar to those in dplyr, and yes, there are other ways to code the same functions.

Cleaning data often takes up a majority of the time spent on data analysis and tidying data is one part of cleaning. One estimate is that cleaning takes up 80% of the time spend on most analyses. Learning how to tidy data will hopefully save time and make running analyses less frustrating.

Theory and background:

> "Happy families are all alike; every unhappy family
> is unhappy in its own way." — Leo Tolstoy

Hadley Wickham (2014) used this quotation first, but it's a brilliant analogy for understanding dirty and untidy data (also, I'm currently reading *Anna Karenina* so I feel I'm justified); clean datasets are all alike, but messy ones are uniquely messy.

Almost all raw data is "dirty" meaning it's not in a form where you could run a statistical test on it, visualize it, or make conclusions with it. The data collected in the field, analyzed in the lab, or found in a database needs to be cleaned before you can perform other analyses on it.

Data cleaning involves parsing data and numbers (making sure the grammar is correct), dealing with missing values, correcting character encoding (for international data), identifying typos (data points that are the same to humans but not computers), verifying experimental design (an on-going process) and filling in structural missing values. It also involves looking for inconsistencies in your data, such as outliers and impossible values. Data cleaning should be done in a consistent and repeatable way (meaning coded). There isn't one way to clean data; instead, it's an ongoing process.

Tidying data is a small part of cleaning data. To go over a few definitions, a dataset contains values, which are either numbers or strings. Every value is part of both a variable and an observation. The variable measures the same underlying attribute (such as height, temperature, concentration of nitrogen), and an observation is all the variables that are measured on one unit (such as a person or a lake).

Data is “tidy” when each variable is a column, each observation is a row.

While you might intuitively know to do this with data in R, these functions will help you if you ever receive a dataset that isn’t tidy. Also, if you want to ask a slightly different question, you might need to rearrange your data.

Tidying your data will make running statistical tests and making plots easier because most analysis need a tidy input. Tidy data is consistent, repeatable and transferable. Most modelling works best with tidy data.

Also, the outputs of different statistical tests are often not tidy, so if you work with the outputs of tests, it’s important re-tidy this new data.

Packages used: tidyr

Tidyr is a package by Hadley Wickham and is an evolution of the **reshape** and **reshape2** packages. It is designed to work with the **dplyr** and **ggplot2** packages. It does fewer functions better than reshape and reshape2. It currently only works on data frames and it doesn’t aggregate. Simple functions can be strung together with %>%.

Files needed:

Some example datasets will be coded/created before the example.
One example data set will be “dirty_iris.csv”.

Functions:

Gather()
Spread()
Unite()
Separate()
Extract_numeric()
Fill()
Full_seq()
Replace_na()

NOTE: Column names refers the to “bare” name, meaning the number (1, 2, 3 etc.) of the column. You can add an underscore to each function e.g. unite_() to call the character names of the columns.

GATHER FUNCTION

What it does:

It rearranges how the columns and rows are organized. It will replicate several column names into one column of data point IDs.

Gather is arguably the most useful function in the tidyr package. You can do unite and separate with other functions, but this function is what tidyr does well.

Make an example dataset:

```
messy.lakes <- data.frame(
  id = 1:4,
  salinity = sample(rep(c('high','low'), each =2)),
  nitrite.1 = runif(4), nitrite.2 =runif(4),
  phosphorus.1 = runif(4), phosphorus.2 = runif(4))
```

Print out:

	id	week	nitrite.1	nitrite.2	phosphorus.1	phosphorus.2
1	1	2	0.08513597	0.6158293	0.1135090	0.05190332
2	2	1	0.22543662	0.4296715	0.5959253	0.26417767
3	3	2	0.27453052	0.6516557	0.3580500	0.39879073
4	4	1	0.27230507	0.5677378	0.4288094	0.83613414

From the help file:

```
gather(data, key, value, ..., na.rm = FALSE, convert = FALSE,
  factor_key = FALSE)
```

What this means:

gather(your data, key=the name of the first new column (this is the column where the names of the existing columns will go), value= the name of second new column, (this is where the individual data points will go))

Example of the function:

```
tidier.lakes <- gather(messy.lakes, key, nutrient, -id, -
salinity)
```

Print out: (first 6 points)

	id	week	key	nutrient
1	1	2	nitrite.1	0.08513597
2	2	1	nitrite.1	0.22543662
3	3	2	nitrite.1	0.27453052
4	4	1	nitrite.1	0.27230507
5	1	2	nitrite.2	0.61582931
6	2	1	nitrite.2	0.42967153

Spread Function

What it does:

Spread() does the opposite as Gather(), meaning it turns “long” data into “wide”. However, it’s not as common to use spread(), (gather() is more common).
Spread() can turn the gathered data back into the “messy” form. It would be useful if you needed to rearrange tidy data to ask a different question.

From the help file:

```
spread(data, key, value, fill = NA,convert = FALSE, drop = TRUE)
```

What this means:

spread(my data, key=“column containing values to turn into column names”, value=“the name of the column where the values will be separated into different columns”)

Example of the function:

```
spread(tidier.lakes, key, nutrient, fill= NA, convert=F, drop=T)
  id week  nitrite.1 nitrite.2 phosphorus.1 phosphorus.2
1  1    2 0.08513597 0.6158293    0.1135090   0.05190332
2  2    1 0.22543662 0.4296715    0.5959253   0.26417767
3  3    2 0.27453052 0.6516557    0.3580500   0.39879073
4  4    1 0.27230507 0.5677378    0.4288094   0.83613414
```

UNITE FUNCTION

What it does:

It merges two or more columns into one column. It’s similar to the paste function.

From the help file:

```
unite(data, col, ..., sep = "_", remove = TRUE)
```

What this means:

Unite(“your data”, “the name of the new column”, “the columns to merge”, “what you want the data points to be separated by”, “if true it will remove the two old columns from the new dataframe")
Unite() accepts the number of the column e.g. 1, 2, 3. You can use colons and minus symbols to specify columns e.g. 1:3 -2

Example of the function: (with and without underscore added)

```
united.data <- unite(mydata,sep.wl, c(1:2), sep="_")
un.data<-unite_(mydata,"sep.w.l", c("sep.l", "sep.w"), sep="_")
```

SEPARATE FUNCTION

What it does:

It separates one column into several columns.

From the help file:

```
separate(data, col, into, sep = "[^[:alnum:]]+", remove = TRUE,
  convert = FALSE, extra = "warn", fill = "warn", ...)
```

What this means:

separate(data: “your data”, col: "the column you want to separate", into "the names of the new columns, concatenate (c) if more than one column", sep = "": "what the data is currently separated by e.g. a comma, a space", remove = TRUE, convert = FALSE, extra = "warn", fill = "warn", ...)

Remember to store the new data as a new object, or overwrite the original object of the data

Example of the function:

```
mydata<-separate(mydata, xx, into=c("sep.l","sep.w", "pet.l",
"pet.w", "species"), sep=",")
head(mydata)
```

EXTRACT_NUMERIC FUNCTION

What it does:

It removes all the characters that aren't numbers, and forces it to be a number. It’s useful for strings that humans would think of a numbers, but have extra symbols that confuse the computer. It will ruthlessly extract numbers.

From the help file:

extract_numeric(x)

Example of the function:

example1 <- c("$4,000.00", "50%", "$7", "9b", "#84m")

> extract_numeric(example1)
[1] 4000 50 7 9 84

FILL FUNCTION

What it does:

It fills in missing data by repeating what was entered above. It's useful for when a data point is only recorded when it changes.

From the help file:

```
fill(data, ..., .direction = c("down", "up"))
```

Example of the function:

```
df <- data.frame(Month = 1:12, Year = c(2000, rep(NA, 11)))
df[6,2] <-2001
```

```
   Month Year
1      1 2000
2      2   NA
3      3   NA
4      4   NA
5      5   NA
6      6 2001
7      7   NA
8      8   NA
9      9   NA
10    10   NA
11    11   NA
12    12   NA
```

```
df %>% fill(Year)
```

```
   Month Year
1      1 2000
2      2 2000
3      3 2000
4      4 2000
5      5 2000
6      6 2001
7      7 2001
8      8 2001
9      9 2001
10    10 2001
11    11 2001
12    12 2001
```

FULL_SEQ FUNCTION

What it does:

It completes and fills in the sequence of a group of numbers.

From the help file:

```
full_seq(x, period, tol = 1e-06)
```

The period is what the data is going up by. The data has to be numeric.

Example of the function:

```
part<- c(1,3,6, 7, 10)
[1]  1  3  6  7 10

full_seq(part,1)
 [1]  1  2  3  4  5  6  7  8  9 10

full_seq(part, 0.5)
 [1]  1.0  1.5  2.0  2.5  3.0  3.5  4.0  4.5  5.0  5.5  6.0  6.5
[13]  7.0  7.5  8.0  8.5  9.0  9.5 10.0
```

REPLACE_NA FUNCTION

What it does:

It replaces NAs. (There are many different ways to deal with NAs.)

From the help file:

```
replace_na(data, replace = list(), ...)
```

What this means:

replace_na(list("the name of the column" = "what you want all NAs in that column replaced by"))

Example of the function:

```
example3 <- data_frame(x = c(12, 23, NA, 35, NA), y =
c("nitrogen", NA, "phosphorus", NA, "carbon"))
print(example3)

replace_na(example3, list(x = 0, y = "unknown"))
```

Sources

De Jonge, E and van der Loo M. An introduction to data cleaning with R. 2013. Statistics Netherlands. Discussion Paper.

Wickham, H. Tidy Data. 2014. Journal of Statistical Software. 59(10):1-23.

Wickham, H. Package 'tidyr'. Feb 5, 2016. Version 0.4.1. CRAN repository. https://cran.r-project.org/web/packages/tidyr/tidyr.pdf