The principles of tidy data and introduction to tidyr

Kexin Wan

2017/10/29

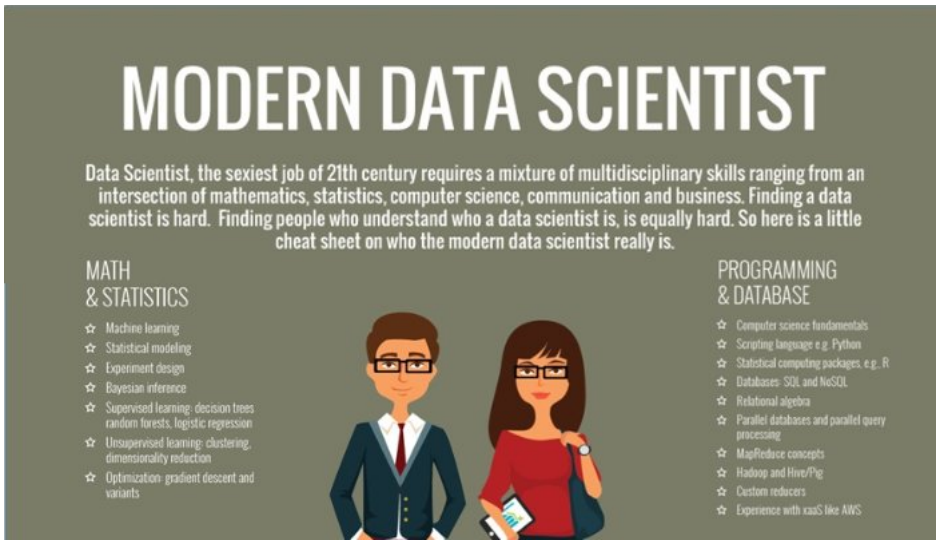
Introduction:

> For data scientists, almost 80% of the data analysis is spent on cleaning and preparing data.

I am sure most of you have heard this quote, which sheds light to the importance of tidy data for later data analysis. However, though so far in this course we have learned how to organize a clean data set in spreadsheet, we haven't touched on the principles of tidy data in R and the usage of data tidying tools in R to clean messy data. Therefore, this post will focus on teaching how to tidy a raw, messy dataset in R. I divide this post into 2 parts. In the first part I would introduce the **principles of tidy data** (and also list examples of messy datas to show common symptons of messy datas) to establish the rules for cleaning messy data. In the second part I would introduce four **basic funtions in tidyr**, which is a package designed specifcally for data tidying, so that we can learn how to use data tidying tools to "clean up" the dataset.

Motivation

Getting data organized is usually the first step towards efficient data analysis. In order to be able to handle real-world datasets, it's important for us to have a framework understanding of what defines tidy data and also how to obtain a tidy data. So far, most of the data used in our homework and lab are already processed and therefore tidy enough to analyze. However, when I try to do some individual data analysis project using real life datasets, I am soon daunted by its disorganization and messiness. After reading some papers, self-studying and practicing I am now able to prepare tidy data by myself. I think this is a really practical skill to possess if we are interested in data analyzing and hope to work in data-related fields in the future.

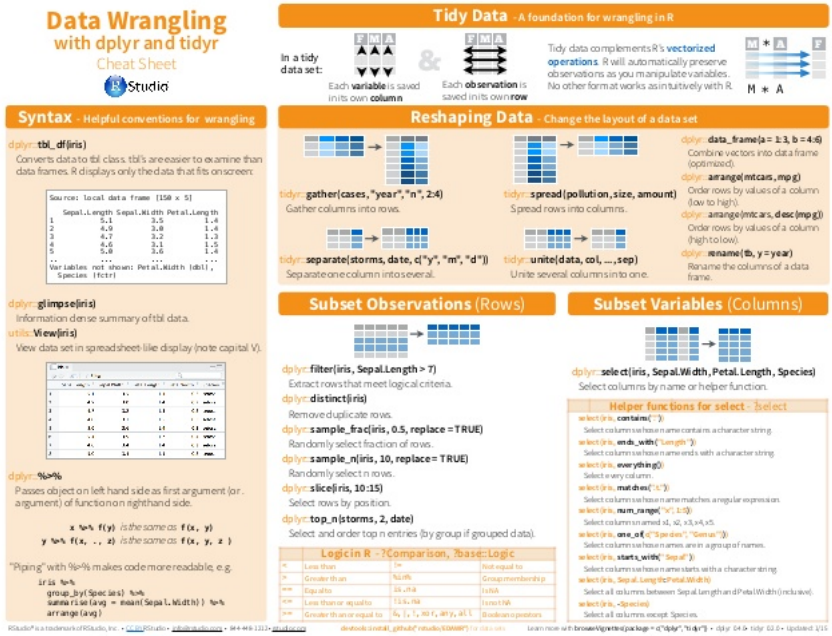


Background

Tidy data is the data obtained as a result of a process called data tidying. It is one of the important cleaning processes during big data processing and is a recognized step in the practice of data science. The tidy data sets have structure and working with them are easy, they are

easy to manipulate, model and visualize. Tidy data sets main concept is to arrange data in a way that each variable is a column and each observation is a row. source

Tidyr is new package designed by Hadley Wickham that makes it easy to tidy your data. Tidy data is data that is easy to work with: it is easy to munge (with dplyr), visualise (with ggplot2 or ggvis) and model (with hundreds of modelling packages in R). source



Tidyr Cheatsheet

Content

- The principles of tidy data (and exmaples of messy dataset)
- Four important functions in tidyr helpful for obtaining a tidy data

Part One – Principles of Tidy Data and Messy Dataset Example

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

In this section I am first going to explain the principles that define tidy dataset, then I will present some examples of messy dataset and explains what principles do they violate.

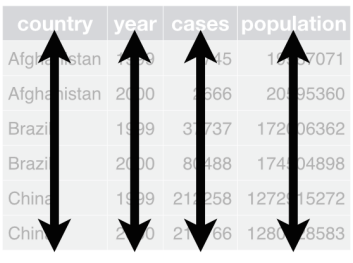
Principles of Tidy Data

Before I introduce the three key principles of tidy data, there are three definitions for dataset we should understand.

- Variable: A quantity, quality, or property that you can measure.
- Observation: A set of values that display the relationship between variables. To be an observation, values need to be measured under similar conditions, usually measured on the same observational unit at the same time.
- Value: The state of a variable you observe when you measure it.

country	year	cases	population
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	1280428583

table1



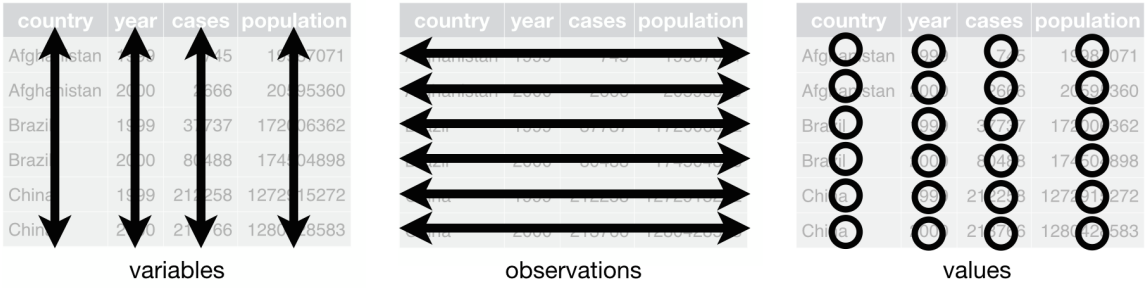
variables

observations

A dataset is said to be tidy if it satisfies the following three principles.

- Each observation in its own row
- Each variable in its own column
- Each value is in its own set

To be more clear, here is an example that illustrates the three rules of tidy dataset.



It might seem like a tidy data is too obvious. However, in practice, raw data is rarely tidy and is much harder to work with. To illustrate this, let's consider some bad examples of messy datasets.

Examples of Messy Datasets

Example One

Considering the following Tuberculosis incidence dataset, why it could not be considered as a tidy data?

```
##   iso2 year new_sp new_sp_m04 new_sp_m514 new_sp_m014 new_sp_m1524
## 1   AD 1989     NA          NA           NA           NA            NA
## 2   AD 1990     NA          NA           NA           NA            NA
## 3   AD 1991     NA          NA           NA           NA            NA
## 4   AD 1992     NA          NA           NA           NA            NA
## 5   AD 1993     15          NA           NA           NA            NA
## 6   AD 1994     24          NA           NA           NA            NA
##   new_sp_m2534 new_sp_m3544 new_sp_m4554 new_sp_m5564 new_sp_m65 new_sp_mu
## 1           NA           NA           NA           NA          NA        NA
## 2           NA           NA           NA           NA          NA        NA
## 3           NA           NA           NA           NA          NA        NA
## 4           NA           NA           NA           NA          NA        NA
## 5           NA           NA           NA           NA          NA        NA
## 6           NA           NA           NA           NA          NA        NA
##   new_sp_f04 new_sp_f514 new_sp_f014 new_sp_f1524 new_sp_f2534
## 1         NA          NA          NA           NA           NA
## 2         NA          NA          NA           NA           NA
## 3         NA          NA          NA           NA           NA
## 4         NA          NA          NA           NA           NA
## 5         NA          NA          NA           NA           NA
## 6         NA          NA          NA           NA           NA
##   new_sp_f3544 new_sp_f4554 new_sp_f5564 new_sp_f65 new_sp_fu
## 1           NA           NA           NA         NA        NA
## 2           NA           NA           NA         NA        NA
## 3           NA           NA           NA         NA        NA
## 4           NA           NA           NA         NA        NA
## 5           NA           NA           NA         NA        NA
## 6           NA           NA           NA         NA        NA
```

Answer: Except for iso2 and year, the rest of the columns headers are actually values of a lurking variable, in fact combination of two lurking variavles, gender and age.

Example Two

Considering the following Whether dataset, why it could not be considered as a tidy data?

```
##            id year month element d1  d2  d3 d4  d5 d6 d7 d8 d9 d10 d11 d12
## 1 MX000017004 2010     1    TMAX NA  NA  NA NA  NA NA NA NA NA  NA  NA  NA
## 2 MX000017004 2010     1    TMIN NA  NA  NA NA  NA NA NA NA NA  NA  NA  NA
## 3 MX000017004 2010     2    TMAX NA 273 241 NA  NA NA NA NA NA  NA 297  NA
## 4 MX000017004 2010     2    TMIN NA 144 144 NA  NA NA NA NA NA  NA 134  NA
## 5 MX000017004 2010     3    TMAX NA  NA  NA NA 321 NA NA NA NA 345  NA  NA
## 6 MX000017004 2010     3    TMIN NA  NA  NA NA 142 NA NA NA NA 168  NA  NA
##   d13 d14 d15 d16 d17 d18 d19 d20 d21 d22 d23 d24 d25 d26 d27 d28 d29 d30
## 1  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA 278
## 2  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA 145
## 3  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA 299  NA  NA  NA  NA  NA  NA  NA
## 4  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA 107  NA  NA  NA  NA  NA  NA  NA
## 5  NA  NA  NA 311  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA
## 6  NA  NA  NA 176  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA
##   d31
## 1  NA
## 2  NA
## 3  NA
## 4  NA
## 5  NA
## 6  NA
```

Answer: This dataset seems to have two problems. First, it has variables in the rows in the column element. Second, it has a variable d in the column header spread across multiple columns.

Example Three

Considering the following Whether dataset, why it could not be considered as a tidy data?

```
##                       religion <$10k $10-20k $20-30k $30-40k $40-50k
## 1                     Agnostic    27      34      60      81      76
## 2                      Atheist    12      27      37      52      35
## 3                     Buddhist    27      21      30      34      33
## 4                     Catholic   418     617     732     670     638
## 5 Don\342\200\231t know/refused    15      14      15      11      10
## 6              Evangelical Prot   575     869    1064     982     881
##   $50-75k $75-100k $100-150k >150k Don't know/refused
## 1     137      122       109    84                96
## 2      70       73        59    74                76
## 3      58       62        39    53                54
## 4    1116      949       792   633              1489
## 5      35       21        17    18               116
## 6    1486      949       723   414              1529
```

Answer: This dataset has column headers as values but not variable names.

Features of Messy Dataset

There are various features of messy data that one can observe in practice. Here I generalize some of the more commonly observed patterns to idenfify.

- Column headers are values, not variable names
- Multiple variables are stored in one column
- Variables are stored in both rows and columns

Now we have finished talking about the principles of tidy dataset and some general patterns to identify messy datasets using examples. In the second part I am going to introduce specific funtions in tidyr that can help us deal with each corresponding problems in messy dataset.

Part Two – Introduction to tidyr

In this section I going to introduce 4 functions in tidyr, a package designed by Hadley Wickham that makes it easier to tidy the data following the three principles. These functions can help us deal with specific problems listed above to make our data set tidy.

Function One – Gather

- When to use: When column headers are values, not variable names.
- Description: Gather should be used when you have columns that are not variables and you want to collapse them into key-value pairs. The easiest way to visualize the effect of gather() is that it makes wide datasets long.
- Format: gather(data, key = “key”, value = “value”, …, na.rm = FALSE, convert = FALSE, factor_key = FALSE)
- Example: I will use the third example in part one to show how to take multiple column and collapse them into key-value pairs. Notice that in this example the various column headers should be stored as values in the dataset.

```
## Warning: package 'tidyr' was built under R version 3.4.2
```

```
# Turning the pew dataset into a tidy dataset by using the gather function in tidyr
tidy_pew <- gather(pew, salary, number, -religion)
head(tidy_pew)
```

```
##                       religion salary number
## 1                     Agnostic  <$10k     27
## 2                      Atheist  <$10k     12
## 3                     Buddhist  <$10k     27
## 4                     Catholic  <$10k    418
## 5 Don\342\200\231t know/refused  <$10k     15
## 6              Evangelical Prot  <$10k    575
```

- Comment: Notice that salary becomes the name of the second column, which takes in all of the previous column name (except religion) as the new column values, acting as keys. While number becomes the name of the third column, which takes in all of the previous values corresponding to each column name, acting as values. After the key-value are paired up together, a new tidy data set is formed. Also notice that I add “-religion” in the end so that the religion column is not turned into value under the salary columns.

Function Two – Spread

- When to use: When Variables are stored in both rows and columns
- Description: Gather should be used when variables are stored in both rows and columns. The opposite of gather() is spread(), which takes key-values pairs and spreads them across multiple columns. This is useful when values in a column should actually be column names (i.e. variables). It can also make data more compact and easier to read. The easiest way to visualize the effect of spread() is that it makes long datasets wide.
- Format: spread(data, key, value, fill = NA, convert = FALSE, drop = TRUE, sep = NULL)
- Example: To show how spread works, Let’s use a data frame called animals. Notice that in this case variables (the type of pet) are also stored in the row as values. However, the values in the pet column should be variables rather than values. Therefore, we use the spread function to spread them across multiple columns.

```
##    name     pet value
## 1  Jake  n_dogs     1
## 2  Jake  n_cats     0
## 3  Jake n_birds     1
## 4 Alice  n_dogs     1
## 5 Alice  n_cats     2
## 6 Alice n_birds     0
```

```
## Turning the animals dataset into a tidy dataset by using the spread function in tidyr
tidy_animals <- spread(animals,pet,value)
tidy_animals
```

```
##    name n_birds n_cats n_dogs
## 1 Alice       0      2      1
## 2  Jake       1      0      1
```

- Comment: By using the spread function in tidyr, we succeed in turning all of pet variable to columns. Now there is no variable in row and is a tidy dataset.

Function Three – Separate

- When to use: When multiple variables are stored in one column
- Description: Separate should be used when you have multiple variables stored in one column. The separate() function allows you to separate one column into multiple columns. Unless you tell it otherwise, it will attempt to separate on any character that is not a letter or number. You can also specify a specific separator using the sep argument.
- Format: separate(data, col, into, sep = "[^[:alnum:]]+", remove = TRUE, convert = FALSE, extra = "warn", fill = "warn", …)
- Example: To show how to use it, let's use a data frame called treatments, which record the number and type of treatment given to each patient during a specific month.Notice that in the year_mo column, there are actually two variables stored in one column: the year and the month of the treatment, which makes it a messy dataset.

```
##   patient treatment year_mo times
## 1       X         A 2010.10     1
## 2       Y         A 2010.10     4
## 3       X         B 2010.70     2
## 4       Y         B 2010.70     5
## 5       X         C 2014.12     3
## 6       Y         C 2014.12     6
```

```
# Turning the treatment dataset into a tidy dataset by using the separate function in tidyr
tidy_treatments <- separate(treatments,col=year_mo,into=c("Year", "Month"))
tidy_treatments
```

```
##   patient treatment Year Month times
## 1       X         A 2010     1     1
## 2       Y         A 2010     1     4
## 3       X         B 2010     7     2
## 4       Y         B 2010     7     5
## 5       X         C 2014    12     3
## 6       Y         C 2014    12     6
```

- Comment: By using the separate function in tidyr, we succeed in separating the column into two columns Year and Month, therefore only one variable is stored in each column. It's now a tidy dataset.

Function Four – Unite

- When to use: When multiple columns actually store the same variable.
- Description: The opposite of separate() is unite(), which takes multiple columns and pastes them together. By default, the contents of the columns will be separated by underscores in the new column, but this behavior can be altered via the sep argument.
- Format: unite(data, col, …, sep = "_", remove = TRUE)
- Example: To show how to use it, let's use a dataframe called lipsticks, which record the color,col_type and brand of the lipstick for Jane and Audrey. Notice that in this example both color and col_type are describing the same variable – color. Therefore, we can use unite to merge this columns together.

```
##     name color col_type    brand
## 1   Coco   red     deep     Dior
## 2   Coco  plum    light    Stila
## 3   Coco  pink   medium Smashbox
## 4 Vivien   red   medium   Armani
## 5 Vivien  plum    light      YSL
## 6 Vivien  pink     deep       CT
```

```
# # Turning the lipstick dataset into a tidy dataset by using the unite function in tidyr
tidy_lipsticks <- unite(lipsticks,color,color,col_type,sep="_")
tidy_lipsticks
```

```
##      name       color    brand
## 1    Coco    red_deep     Dior
## 2    Coco  plum_light    Stila
## 3    Coco pink_medium Smashbox
## 4 Vivien  red_medium   Armani
## 5 Vivien  plum_light      YSL
## 6 Vivien   pink_deep       CT
```

- Comment: By using the unite function in tidyr, we succeed in uniting two columns color and col_type and turn them into one column, representing only one variable. It's now a tidy dataset.

Conclusion

Now we have finished talking about the principles of tidy data and the four most important functions in tidyr that can help us turn messy dataset into a tidy dataset. There are actually more functions in tidyr that can be employed for cleaning our data. If you are interested in learning more about tidyr and tidy data, here are some useful resources for you.

Comprehensive introduction of tidyr package

Principles of tidy data and tidyr PPT

Johns Hopkins Tidy Data Course on Coursea

Take Home Message

After reading this post we should now learn:

A tidy data, which is a data following the three principles, is really important for the convenience of later data analysis. Therefore, before we start to analyze the data, we should always check if a data is tidy first, if not, we can then use the four most common functions (or more) in tidyr to clean it up.

References:

- Data Tidying by Garrett Grolemund
- RPubs Tidyr
- Cran R Tidy Data
- Tidy Data by Hadley Wickham
- Tidy Data Wikipedia
- Principles of tidy data and tidyr PPT
- Comprehensive introduction of tidyr package