Data Tidying with Tidyr Package

Min Ju Chung

12/1/2017

> ## Content:
>
> - Introduction
> - Background
> - Four Fundamental Tidyr Operations
> - Conclusion and Take-Home Message
> - List of References

```
# install package if needed
install.packages("tidyr", repos="http://cran.rstudio.com/")
```

```
## Installing package into '/Users/MinChung/Library/R/3.3/library'
## (as 'lib' is unspecified)
```

```
##
## The downloaded binary packages are in
##  /var/folders/wp/pb5dh4c97bg2vtb6zmjg2rnw0000gn/T//RtmpjmtTKT/downloaded_packages
```

```
library(tidyr)
```

```
## Warning: package 'tidyr' was built under R version 3.3.2
```

Introduction

This post will explore new and consistent ways to organize and clean datasets for data analysis. Created by Hadley Wickham, the tidyr package was specifically designed for data tidying.

The **motivation** of this post is to introduce practical tools for data cleaning. Having tidy data will not only save time but also make data visualization, manipulation, and modeling easier! Because raw data in the real-world is often messy and unorganized, it is essential to first create tidy data for efficient data analysis. This post will discuss the principles of a tidy data set and focus on four fundamental tidyr functions, `gather()`, `spread()`, `separate()`, and `unite()`.

Background

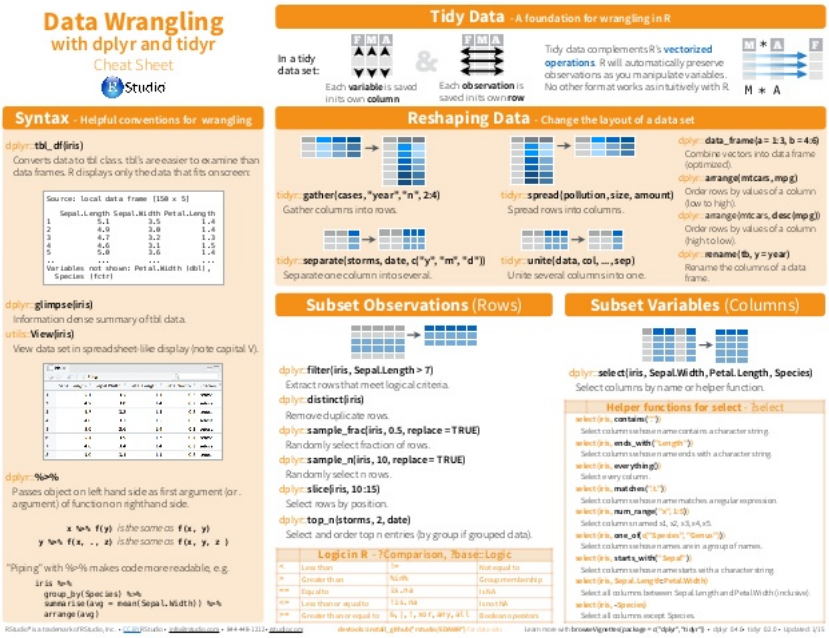
What is Tidy Data? A data set is called **tidy** if it satisfies the following components:

- Each variable forms a column.
- Each observation forms a row.
- Each type of observational unit forms a table.

One of advantages of tidy data is its suitability for vectorized programming languages like R because the format of tidy data ensures that the values of different variables from the same observation are paired. The tidyr package provides one uniform way of storing data.

What is Messy Data? This includes any other arrangement of the data. One common problem of messy datasets is the use of values as column headers, not variable names. In a messy dataset, you need to use multiple methods to extract different variables. However, most messy datasets can be tidied with tools, such as the four fundamental tidyr functions.

Here is the Data Wrangling with dplyr and tidyr cheat sheet! This summarizes the main functions used to reshape and tidy data.



Four Fundamental Tidyr Operations

gather() function

- `gather()` : reformats the data by taking multiple columns and placing them into key-value pairs.
- This function is generally used when the column headers are values, not variable names.
- Format: gather(data, key, value, …, na.rm = FALSE, convert = FALSE)
- Key arguments:
    - `key` corresponds to the column name of the new variable
    - `value` corresponds to the column name of variable values
    - `…` represents the names of columns to gather

We can use `gather()` to tidy the data frame in the following example.

```
df1 <- data.frame(
  States = c("Alaska", "California", "Texas", "Oregon"),
  "2014" = c(1296, 191, 8, 4),
  "2015" = c(1575, 130, 21, 3),
  check.names = FALSE
)
df1
```

```
##        States 2014 2015
## 1      Alaska 1296 1575
## 2 California  191  130
## 3       Texas    8   21
## 4      Oregon    4    3
```

The column names, “2014” and “2015”, are values, not variable names. It is hard to interpret this table as the column names are neither clear nor specific. We can use the gather function to tidy this table!

```
tidy_df1 <- gather(df1, "Year", "Earthquake Counts", 2:3)
tidy_df1
```

```
##        States Year Earthquake Counts
## 1      Alaska 2014               1296
## 2 California 2014                191
## 3       Texas 2014                  8
## 4      Oregon 2014                  4
## 5      Alaska 2015               1575
## 6 California 2015                130
## 7       Texas 2015                 21
## 8      Oregon 2015                  3
```

spread() function

- `spread()` : cleans the dataset by spreading a key-value pair across multiple columns.
- This function is mainly used when the variables are stored in both rows and columns.
- Format: spread(data, key, value, fill = FALSE, convert = FALSE)
- Key arguments:
    - `key` corresponds to the column name to convert to multiple columns
    - `value` corresponds to the column name to convert to the values of the multiple columns

By applying `spread()` to the following table, the data is presented in a more concise manner.

```
df2 <- data.frame(
  States = c(rep("California", 4), rep("Texas", 4)),
  Year = c(2014, 2014, 2015, 2015, 2014, 2014, 2015, 2015),
  Key = c(rep(c("Earthquake Counts", "Population (in millions)"), 2)),
  Value = c(191, 38.68, 130, 39.99, 8, 26.94, 21, 27.43)
)
df2
```

```
##       States Year                       Key  Value
## 1 California 2014        Earthquake Counts 191.00
## 2 California 2014 Population (in millions)  38.68
## 3 California 2015        Earthquake Counts 130.00
## 4 California 2015 Population (in millions)  39.99
## 5      Texas 2014        Earthquake Counts   8.00
## 6      Texas 2014 Population (in millions)  26.94
## 7      Texas 2015        Earthquake Counts  21.00
## 8      Texas 2015 Population (in millions)  27.43
```

```
tidy_df2 <- spread(df2, Key, Value)
tidy_df2
```

```
##       States Year Earthquake Counts Population (in millions)
## 1 California 2014               191                    38.68
## 2 California 2015               130                    39.99
## 3      Texas 2014                 8                    26.94
## 4      Texas 2015                21                    27.43
```

separate() function

- `separate()` : splits a single column variable into two columns
- This function is generally used when multiple variables are stored in one column.
- Format: separate(data, col, into, sep = " " , remove = FALSE, convert = FALSE)
- Key arguments:
    - `col` corresponds to the column name of the selected variable to split
    - `into` corresponds to the vector of character strings to use as new column names
    - `sep` corresponds to the non-alphanumeric character, such as `.` , `-` , `:` .

Let's take a look at the following example!

```
df3 <- data.frame(
  Names = c("John", "Steve", "Christine", "Stan"),
  sex_age = c("M-23", "M-25", "F-21", "M-29")
)
df3
```

```
##       Names sex_age
## 1      John    M-23
## 2     Steve    M-25
## 3 Christine    F-21
## 4      Stan    M-29
```

```
tidy_df3 <- separate(df3, col = sex_age,
                     into = c("Sex", "Age"),
                     sep = "-")
tidy_df3
```

```
##       Names Sex Age
## 1      John   M  23
## 2     Steve   M  25
## 3 Christine   F  21
## 4      Stan   M  29
```

By applying the `separate()` function, the variable, `sex_age` , is split, at the dash character, into two columns, `Sex` and `Age` .

unite() function

- `unite()` : merges two variables into one variable
- This function is generally used when one variable is spread across multiple columns
- Format: spread(data, col, …, sep = " " , remove = FALSE)
- Key arguments:
    - `col` corresponds to the column name of combined column
    - `…` corresponds to the names of the columns to merge
    - `sep` corresponds to the non-alphanumeric character, such as `.` , `-` , `:` , to use between merged values

In the following example, I combined the columns, "Sex" and "Age", into one column. Essentially, I "undid" the `separate()` function on df3.

```
tidy_df4 <- unite(tidy_df3, col = "sex_age", 2:3, sep = "-")
tidy_df4
```

```
##       Names sex_age
## 1      John    M-23
## 2     Steve    M-25
## 3 Christine    F-21
## 4      Stan    M-29
```

Conclusion and Take-Home Message

The Tidyr package is useful for cleaning and tidying messy datasets. The process of data tidying is a crucial step that should not be overlooked! It will save time in the longer run and make data analysis easier. The fundamental functions, `gather()`, `spread()`, `separate()`, and `unite()`, are four of the many tools in the tidyr package that easily allows users to organize data set in a consistent and tidy manner.

References

- Data Processing with dplyr & tidyr
- Data Tidying
- Tidy data
- Data manipulation with tidyr and dplyr
- Tidyr: Crucial Step Reshaping Data with R for Easier Analyses
- Tidy Data
- Video on Tidyr R Tutorial