post1-serena-wu

Serena Wu

10/23/2017

(Because I write this post in a folder consisting of `code`, `data`, `images`, `output`, and `report`. I include my github link here for my post1, in case if I refer to any of the folders in the following content.)

Data Manipulation

Introduction

My topic here is data manipulation. Data manipulation has drawn my attention as it becomes more and more noteworthy in data science realm. When performing data analytics and presenting data, clean and tidy data can give people an easier grasp of main information and a more accurate way of interpreting data, which is the reason why data manipulation becomes so important, and why I want to write a post about it. In this post, I will focus on data reshaping and wrangling with tidyr and data formatting, which are specific topics in data manipulation.

I will have extensive examples and guidelines to illustrate the application of each process and function, and therefore my readers can benefit from learning about how to use these methods. The packages that I will use are dplyr and tidyr, which are two most commonly used packages for data manipulation. I search for relative information of my topic and the examples online from R-bloggers, R-tutorials, and so on. I have listed all the references at the bottom of this file, which are really great resources to find more examples from.

I will have 2 main sections - data reshaping & wrangling, and data formatting. Inside each section, I will have sub-sections like examples, and discussion & summary. Afterwards, I will have a big summary section.

Key Concepts

Data Manipulation: due to uncontrollable factors in data collecting process that might lead to inaccuracy in data interpretation, data manipulation is done to reduce bias and enhance accuracy by manipulating data using available variables.

Data Reshaping & Wrangling: to organize data into rows and columns and transform data from raw data to more appropriate format. Data is often described to be in “wide” format when there is one observation row per subject, and in “long” format when there is one observation row per measurement.

Data Formatting: to organize data in columns and rows according to certain organization guidelines, to tidy up datasets according to observations, variables, and types, making it easy to manage and analyze.

Let’s begin our journey to data manipulation!!

I will use `nba2017-players.csv` that we have used in class as data source to illustrate application for data manipulation. Further information and references can be found in `nba2017-players-dictionary.md` under my `post1` folder in the `data` folder.

Section 1: Data Reshaping & Wrangling

To begin with, I need to load `tidyr` and `dplyr` library and read data from `nba2017-players.csv`.

```
# install.packages("tidyverse")
library(tidyr)
```

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

```
library(dplyr)
```

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

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
data <- read.csv(file = "../data/nba2017-players.csv", stringsAsFactors = FALSE)
```

First, we can learn about some helpful conventions in dplyr and utils for data wrangling. They are `tbl_df`, `glimpse`, and `View`.

What `tbl_df` does is to convert data to tbl. I will use `data` from `nba2017-players.csv` that we loaded above.

```
dplyr::tbl_df(data)
```

```
## # A tibble: 441 x 15
##               player  team position height weight   age experience
##                <chr> <chr>    <chr>  <int>  <int> <int>      <int>
##  1        Al Horford   BOS        C     82    245    30          9
##  2      Amir Johnson   BOS       PF     81    240    29         11
##  3     Avery Bradley   BOS       SG     74    180    26          6
##  4 Demetrius Jackson   BOS       PG     73    201    22          0
##  5      Gerald Green   BOS       SF     79    205    31          9
##  6     Isaiah Thomas   BOS       PG     69    185    27          5
##  7       Jae Crowder   BOS       SF     78    235    26          4
##  8       James Young   BOS       SG     78    215    21          2
##  9      Jaylen Brown   BOS       SF     79    225    20          0
## 10     Jonas Jerebko   BOS       PF     82    231    29          6
## # ... with 431 more rows, and 8 more variables: college <chr>,
## #   salary <dbl>, games <int>, minutes <int>, points <int>, points3 <int>,
## #   points2 <int>, points1 <int>
```

`glimpse` denses the information summary of tbl data.

```
dplyr::glimpse(data)
```

```
## Observations: 441
## Variables: 15
## $ player     <chr> "Al Horford", "Amir Johnson", "Avery Bradley", "Dem...
## $ team       <chr> "BOS", "BOS", "BOS", "BOS", "BOS", "BOS", "BOS", "B...
## $ position   <chr> "C", "PF", "SG", "PG", "SF", "PG", "SF", "SG", "SF"...
## $ height     <int> 82, 81, 74, 73, 79, 69, 78, 78, 79, 82, 80, 84, 76,...
## $ weight     <int> 245, 240, 180, 201, 205, 185, 235, 215, 225, 231, 2...
## $ age        <int> 30, 29, 26, 22, 31, 27, 26, 21, 20, 29, 22, 25, 22,...
## $ experience <int> 9, 11, 6, 0, 9, 5, 4, 2, 0, 6, 1, 3, 2, 1, 4, 10, 1...
## $ college    <chr> "University of Florida", "", "University of Texas a...
## $ salary     <dbl> 26540100, 12000000, 8269663, 1450000, 1410598, 6587...
## $ games      <int> 68, 80, 55, 5, 47, 76, 72, 29, 78, 78, 25, 75, 79, ...
## $ minutes    <int> 2193, 1608, 1835, 17, 538, 2569, 2335, 220, 1341, 1...
## $ points     <int> 952, 520, 894, 10, 262, 2199, 999, 68, 515, 299, 38...
## $ points3    <int> 86, 27, 108, 1, 39, 245, 157, 12, 46, 45, 0, 68, 94...
## $ points2    <int> 293, 186, 251, 2, 56, 437, 176, 13, 146, 69, 15, 19...
## $ points1    <int> 108, 67, 68, 3, 33, 590, 176, 6, 85, 26, 8, 90, 203...
```

`View` from utils enables us to view data set in spreadsheet.

```
# utils::View(data)
```

(Since it will open a new file here, I put # before them to stop them from making things weird.)

After these basic steps, I will now focus on main methods in data reshaping with `tidyr`. `tidyr` is a package built for creating tidy data. There are four foundamental methods:

1. `gather()`, which is used to reshape “wide” format to “long” format
2. `spread()`, which is used to reshape long format to wide format
3. `separate()`, which is used to split a variable into two
4. `unite()`, which is used to merge two variables into one

Method 1. gather()

gather() is used to reshape “wide” format to “long” format.

Key arguments that gather() takes in: data (data frame), key (column name representing new variable), value (column name representing variable values).

I will first use dplyr, which we have learnt in stat 133, to select first 5 rows of some of the columns in the data to simplify my example.

```
wide <- head(select(data, player, age, points1, points2, points3), 5)
write.csv(wide, file = "../output/wide.csv")
wide
```

```
##              player age points1 points2 points3
## 1        Al Horford  30     108     293      86
## 2      Amir Johnson  29      67     186      27
## 3     Avery Bradley  26      68     251     108
## 4 Demetrius Jackson  22       3       2       1
## 5      Gerald Green  31      33      56      39
```

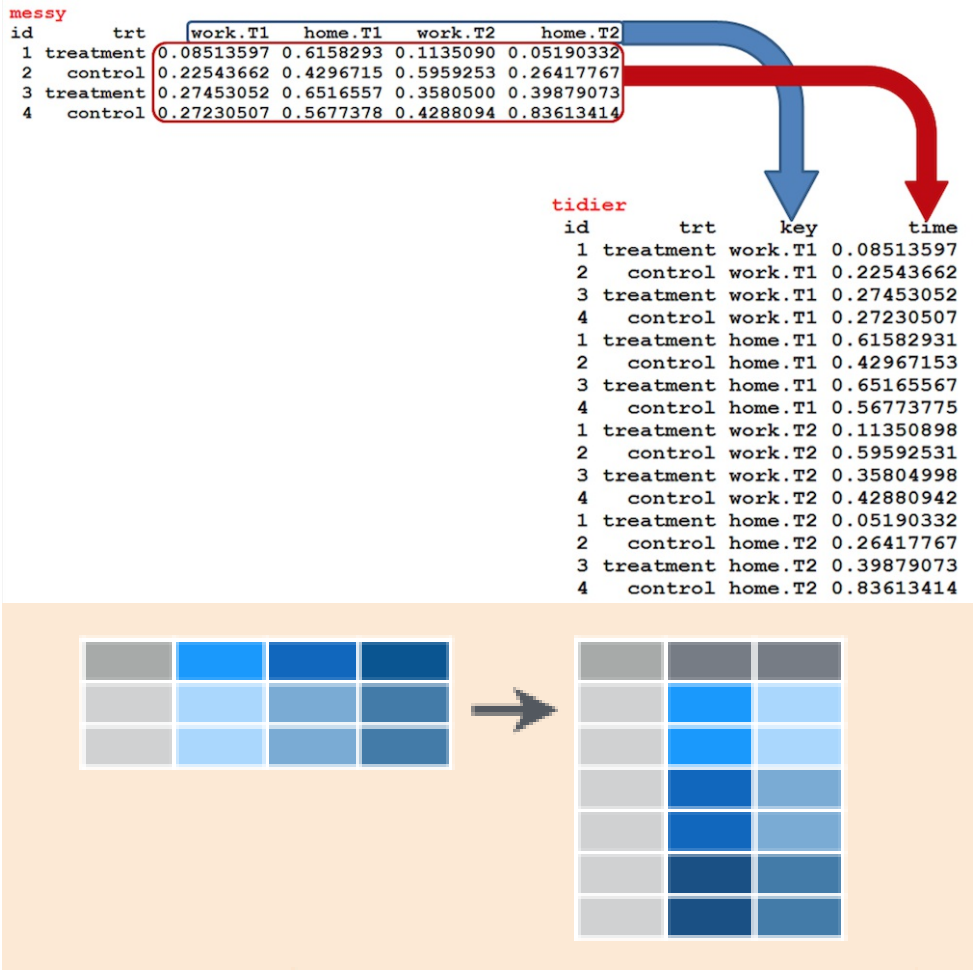
Then I can perform reshaping by using gather().

```
long <- gather(wide, "Points Type", "Number", points1:points3)
write.csv(long, file = "../output/long.csv")
long
```

```
##               player age Points Type Number
## 1          Al Horford  30     points1    108
## 2       Amir Johnson  29     points1     67
## 3      Avery Bradley  26     points1     68
## 4  Demetrius Jackson  22     points1      3
## 5       Gerald Green  31     points1     33
## 6          Al Horford  30     points2    293
## 7       Amir Johnson  29     points2    186
## 8      Avery Bradley  26     points2    251
## 9  Demetrius Jackson  22     points2      2
## 10      Gerald Green  31     points2     56
## 11         Al Horford  30     points3     86
## 12      Amir Johnson  29     points3     27
## 13     Avery Bradley  26     points3    108
## 14 Demetrius Jackson  22     points3      1
## 15      Gerald Green  31     points3     39
```

Basically, the main idea can be illustrated by the following picture that I found online:

Visualization of gather()



Legends/note: In the screenshots, cells with the same shades of blue share the same content.

(These two pictures are screenshots from online resources to illustrate the visualization idea of the method. They are not necessarily charts or graphs generated from ggplot or other graphic packages. Therefore I can't really provide labeled axes, but I have provided a label above these two pictures, and legends below.)

Like I have said, the `gather()` method is transfering wide formats to long formats, these two graphs express the same idea of what I generated above in the data frame.

Method 2. spread()

spread() is used to reshape long format to wide format.

If we feel like transfering the result on the top to wide format again, we can use the spread(), which is basically doing the opposite thing of what gather() does.

spread() usually take in main arguments similar to gather(), including data, key, value, and fill (which is the only new argument, meaning if not every combination of variables and keys have a value, value will be substituted). `fill` is pretty hard to understand but don't worry, we usually won't need to use `fill` since usually we will have value for every combination.

We can use the long from above and perform spread().

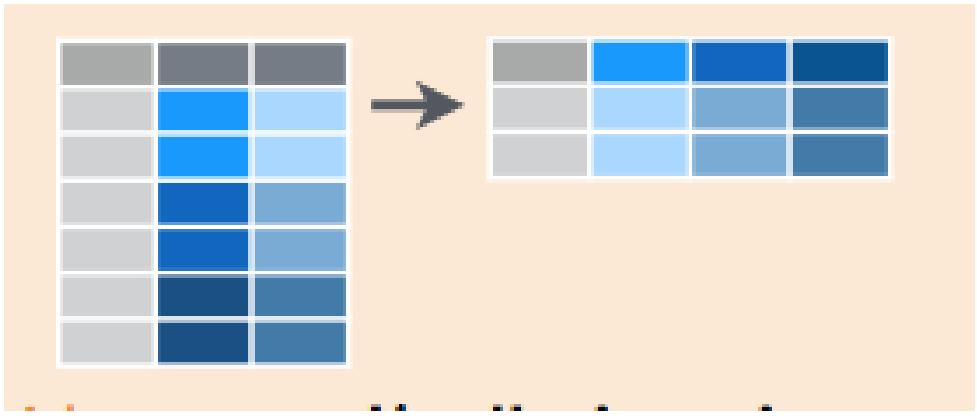
```
wide_again <- spread(long, "Points Type", "Number")
wide_again
```

```
##              player age points1 points2 points3
## 1         Al Horford  30     108     293      86
## 2       Amir Johnson  29      67     186      27
## 3      Avery Bradley  26      68     251     108
## 4 Demetrius Jackson  22       3       2       1
## 5       Gerald Green  31      33      56      39
```

This is exactly the same as the `wide` data frame that we originally created from above!!

What it performs can be visualized by the following picture.

Visualization of spread()



Legends/note: In the screenshots, cells with the same shades of blue share the same content.

(This picture is screenshot from online resources to illustrate the visualization idea of the method. They are not necessarily charts or graphs generated from ggplot or other graphic packages. Therefore I can't really provide labeled axes, but I have provided label above the picture, and legends below)

Like I have said, the `spread()` method is transfering long formats to wide formats, this graph expresses the same idea of what I generated above in the data frame.

Method 3. separate()

separate() is used to split a variable into two.

separate() takes in main arguments like data (data frame), col (column name representing current variable), into (names of variables representing new variables), sep (how to separate current variable. e.g. char, num, or symbol).

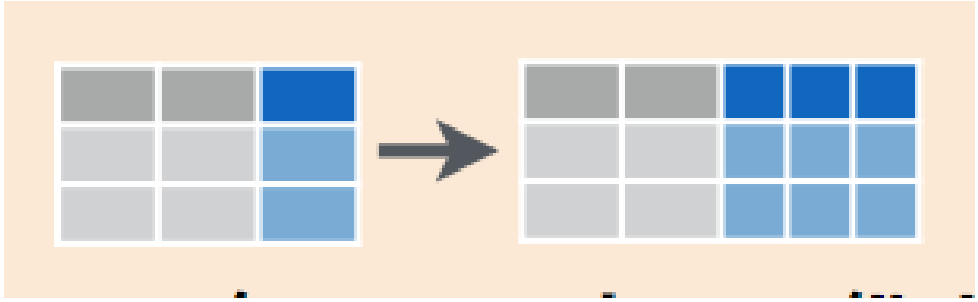
We can use the `long` data frame from above to illustrate the application of separate().

```
separated <- separate(long, col = "Points Type", into = c("Points", "Type Number"), sep = 6)
write.csv(separated, file = "../output/separated.csv")
separated
```

```
##               player age Points Type Number Number
## 1         Al Horford  30 points           1    108
## 2       Amir Johnson  29 points           1     67
## 3      Avery Bradley  26 points           1     68
## 4  Demetrius Jackson  22 points           1      3
## 5       Gerald Green  31 points           1     33
## 6         Al Horford  30 points           2    293
## 7       Amir Johnson  29 points           2    186
## 8      Avery Bradley  26 points           2    251
## 9  Demetrius Jackson  22 points           2      2
## 10      Gerald Green  31 points           2     56
## 11        Al Horford  30 points           3     86
## 12      Amir Johnson  29 points           3     27
## 13     Avery Bradley  26 points           3    108
## 14 Demetrius Jackson  22 points           3      1
## 15      Gerald Green  31 points           3     39
```

Here, by `sep = 6` , I am specifying that the column should be spearated at the 6th of the `Points Type` character.

Visualization of separate()



Legends/note: In the screenshots, cells with the same shades of blue share the same content.

(This picture is screenshot from online resources to illustrate the visualization idea of the method. They are not necessarily charts or graphs

generated from ggplot or other graphic packages. Therefore I can't really provide labeled axes, but I have provided label above the picture, and legends below)

Like I have said, the `separate()` method is splitting one column variable into two, this graph expresses the same idea of what I generated above in the data frame.

Method 4. unite()

unite() is used to merge two variables into one.

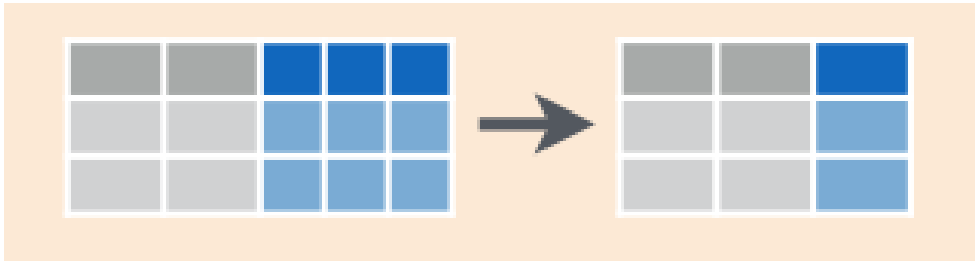
unite() is basically a function to reverse what separate() does. It takes in arguments like data, col (column name of the newly merged column), sep, similar to separate().

We can use `separated` from above and perform `unite()` on it to make it back to `long` again.

```
united <- unite(separated, "Points Type", "Points", "Type Number", sep = "")
write.csv(united, file = "../output/united.csv")
united
```

```
##               player age Points Type Number
## 1         Al Horford  30     points1    108
## 2       Amir Johnson  29     points1     67
## 3      Avery Bradley  26     points1     68
## 4  Demetrius Jackson  22     points1      3
## 5       Gerald Green  31     points1     33
## 6         Al Horford  30     points2    293
## 7       Amir Johnson  29     points2    186
## 8      Avery Bradley  26     points2    251
## 9  Demetrius Jackson  22     points2      2
## 10      Gerald Green  31     points2     56
## 11        Al Horford  30     points3     86
## 12      Amir Johnson  29     points3     27
## 13     Avery Bradley  26     points3    108
## 14 Demetrius Jackson  22     points3      1
## 15      Gerald Green  31     points3     39
```

Visualization of unite()



Legends/note: In the screenshots, cells with the same shades of blue share the same content.

(This picture is screenshot from online resources to illustrate the visualization idea of the method. They are not necessarily charts or graphs generated from ggplot or other graphic packages. Therefore I can't really provide labeled axes, but I have provided label above the picture, and legends below)

Like I have said, the `unite()` method is merging two column variables into one, this graph expresses the same idea of what I generated above in the data frame.

(Note I have written every function's result to a .csv file under the `output` folder, so you can further compare how these outputs look like and have a clearer perspective of what I have perfomed up there.)

Discussion & Summary for data reshaping & wrangling

What I found for data reshaping here are four basic methods: `gather()`, `spread()`, `separate()`, and `unite()`. `gather()` transfers data frame in wide format into long format, and `spread()` does the opposite. `separate()` makes one variables into two while `unite()` gets them back to one. These four functions together manage to organize data into different formats of rows and columns and transform data to more appropriate format. In this way, they help us perform basic reshaping and wrangling of data for further use.

End of Data reshaping & wrangling. Now let's have a look at data formatting, which is to organize data in columns and rows according to certain organization guidelines, to tidy up datasets according to observations, variables, and types, making it easy to manage and analyze.

Section 2: Data Formatting

There are many guidelines for formatting and organizing data. I will provide some of them that I regard as most important ones.

1. Use column labels.

The header (column labels at the very top row of data) can give us a clear idea of what that column is about. We should mark the column labels in first rows of every range of data.

2. Avoid leading or trailing spaces.

Inserting spaces at beginning or the end of a cell might easily lead to errors and affect sorting or searching for the cells. We can also increase indent command within the cell rather than typing spaces.

3. Extend data formats and formulas.

We should extend consistent formatting and formulas when inserting new data into a data range. If we don't do this, the format might seem messy and some columns might dismatch each other, causing troubles and errors.

4. Use cell borders and keep different data range separate.

We should leave at least one blank column and one blank row between a data range and the other to mark the separation of two different and unrelated data. In this way, we can easily detect and select the range for data analysis.

5. Display all rows and columns in a range.

Some data columns or rows might be neglected due to being displayed at the corners or being hidden. Before processing and making changes to a range of data, we should make sure that all data are displayed fully without some hidden ones being excluded.

6. Put similar items in the same column.

We can design every row to have similar items in the same column. Therefore, we putting data together, it will seem more organized and tidy and ready to use.

Discussion & Summary of Data Formatting

Data formatting basically requires us to put data in a neat and tidy way in order to make the following processing and analyzing process eazier. There are many rules and guidelines that we can apply, as I listed some of them above. Everyone has a different taste for organizing and formatting data, but as long as the data doesn't look messy and is easy to use, then well done on data formatting!

Summary

Data manipulation is a great skill to learn, in that it enables us to have a better understanding of data and of performing data processing and analytics on correct and neat data formats.

A quick recap: in the data reshaping & wrangling section, I introduced four basic methods in tidyr:

1. `gather()`, which is used to reshape "wide" format to "long" format
2. `spread()`, which is used to reshape long format to wide format
3. `separate()`, which is used to split a variable into two
4. `unite()`, which is used to merge two variables into one

In the data formatting section, I included some guidelines for ensuring data are in a tidy and ready-to-use format:

1. Use column labels.
2. Avoid leading or trailing spaces.
3. Extend data formats and formulas.
4. Use cell borders and keep different data range separate.
5. Display all rows and columns in a range.
6. Put similar items in the same column.

What I included in this post are basic methods and rules in data manipulation, and I have included references in the following. If you are interested in learning more in this area and exploring more examples in this realm, I will recommend you to click on the following links!

References:

1. UC Business Analytics R Programming Guide
2. RStudio Cheat Sheet
3. Analytics Vidhya
4. UCLA IDRE
5. Data Science with R by Garrett Grolemund
6. Data Manipulation
7. 5 tips for data manipulation in Excel