Post 1

Robby Grewal

10/31/17

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

```
##
## The downloaded binary packages are in
##   /var/folders/sl/qgw5jvlx7f5g4vj68nkdg_bw0000gn/T//RtmpPiElEF/downloaded_packages
```

```
library(tidyr)
```

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

Post 1: Tidy Data with the Tidyr Package

Introduction

The purpose of exploring the Tidyr package is to learn how to approach data sets that aren't well-formatted, or just messy in general. The Tidyr package is used to help "clean up" data so that it can be better manipulated and worked with.

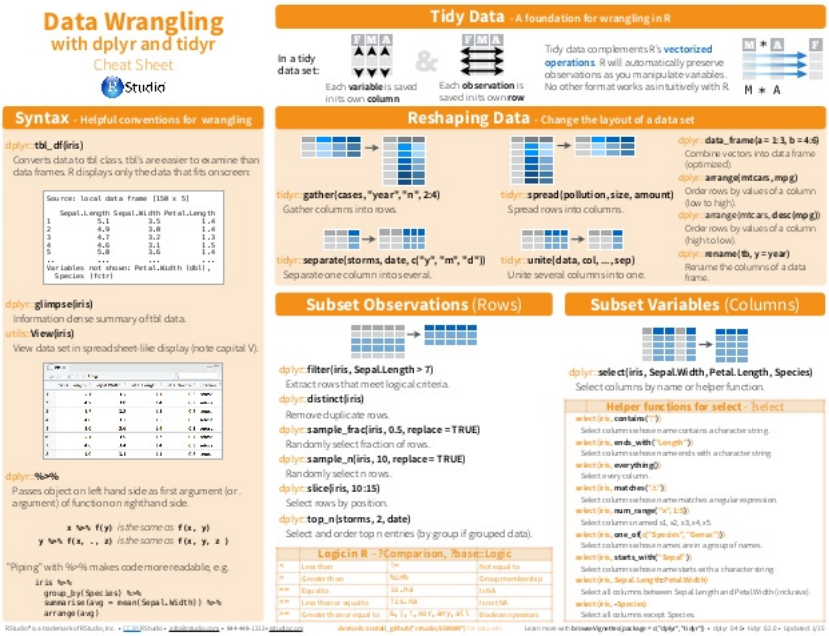
My motivation behind this post is that I work in a breast cancer lab and I often end up with large amounts of data processing I have to do afterwards. I've been using R to do this, and Tidyr is a package I use often in the beginning of my projects in order to get clean data. All of the data sets we have worked with in class have also been clean, well-organized data, and in the real world, this is often not what we are presented with, especially when working on our own research and not pre-existing sources of data. In this post, we will go over what is considered clean versus messy data, and also go over 3 basic function in the "Tidyr package". This package, and overall skill to tidy up data, is invaluable for any data scientist.

Background on Messy Data and Tidyr

What is clean data? A data set is clean if it follows these three principles: 1. Each Observation is in its own row. An observation is a value that displays a relationship between variables. 2. Each Variable is in its own column. A variable is a measurable property. 3. Each value has its own set, or "cell" if you think about it like a spreadsheet.

All tidy data follows these principles, but there are many different ways a data set could be messy, but in general, if they break one of the three principles in any way, they are messy. This means when column headers are values, multiple variables are stored in one column, or variables are in rows, the data is not clean. Those are the most common ways data is found to be messy.

The package Tidyr is used to clean data so the rest of work becomes easier through easier manipulation and visualization. The methods that we will go over in this post are Gather, Spread, Separate, and Unite, and these are four of the basic and major functions in this package. The entire documentation of this package can be found here



Tidyr Cheatsheet

- Gather(): Takes multiple columns and makes them key-value pairs.
- Spread(): Takes key & value columns and spreads them into multiple columns
- Separate(): Used to separate one column into multiple columns
- Unite(): Takes multiple columns and unites them together.

Examples of Tidyr Functions

After installing and loading Tidyr, here are examples of each of the functions outlined above. Note that all tables have hypothetical data used to illustrate messy data, and how to fix it with these functions.

gather(data, key = “key”, value = “value”, …)

This function is used when the column headers of your table are values and not variables. This will take multiple columns and collapse them into key-value pairs. The format of this function is as follows: gather(data, key = “key”, value = “value”, …)

“…” corresponds to a selection of columns. If nothing is specified, all columns are selected.

Here is an example of a table that has values as column headers.

```
Tb1 <- data.frame(
  School = c("UCLA", "Cal", "USC"),
  '2005' = c(16000, 18000, 9000),
  '2010' = c(18000, 20000, 12000),
  check.names = FALSE
)

Tb1
```

```
##   School  2005  2010
## 1   UCLA 16000 18000
## 2    Cal 18000 20000
## 3    USC  9000 12000
```

As you can see, this table has values for columns two and three. They are meant to represent the year and the value is the number of students attending the university. However, that is not easy to understand from this table as you can’t tell what these column headers mean. We can use the gather function to clean this up.

```
gathertb1 <- gather(Tb1, "Year", "Number of Students", 2:3)
gathertb1
```

```
##   School Year Number of Students
## 1   UCLA 2005              16000
## 2    Cal 2005              18000
## 3    USC 2005               9000
## 4   UCLA 2010              18000
## 5    Cal 2010              20000
## 6    USC 2010              12000
```

This function, written as is, takes columns 2 and 3, specified in the function, and gives them names corresponding to “Years”, and “Number of Students” This data set is much easier to work with using packages such as Dplyr and Ggplot2, and it is also easier to understand and read visually.

spread(data, key, value)

This function is used when you have variables in rows and columns. Effectively, this performs the opposite function as gather. You would use this when you have values in a column that should actually be a column name.

Here is an example of a table that

```
tb2 <- data.frame(
  School = c(rep("UCLA", 4), rep("Cal", 4), rep("USC", 4)),
  year = c(2005, 2005, 2010, 2010, 2005, 2005, 2010, 2010, 2005, 2005, 2010, 2010),
  grad_status = c(rep(c("Undergraduates", "Post-Graduates"), 3)),
  value = c(10000, 6000, 11000, 7000, 12000, 6000, 14000, 6000, 6000, 3000, 8000, 4000)
)
tb2
```

```
##    School year    grad_status value
## 1    UCLA 2005 Undergraduates 10000
## 2    UCLA 2005 Post-Graduates  6000
## 3    UCLA 2010 Undergraduates 11000
## 4    UCLA 2010 Post-Graduates  7000
## 5     Cal 2005 Undergraduates 12000
## 6     Cal 2005 Post-Graduates  6000
## 7     Cal 2010 Undergraduates 14000
## 8     Cal 2010 Post-Graduates  6000
## 9     USC 2005 Undergraduates  6000
## 10    USC 2005 Post-Graduates  3000
## 11    USC 2010 Undergraduates  8000
## 12    USC 2010 Post-Graduates  4000
```

We can see in this table that the correct variable would be number of undergraduates, and number of post-graduates. In this table, the appropriate column names are actually in rows, and we would use the spread function to solve this.

```
spread_tb2 <- spread(tb2, key = grad_status, value = value)
spread_tb2
```

```
##   School year Post-Graduates Undergraduates
## 1    Cal 2005           6000          12000
## 2    Cal 2010           6000          14000
## 3   UCLA 2005           6000          10000
## 4   UCLA 2010           7000          11000
## 5    USC 2005           3000           6000
## 6    USC 2010           4000           8000
```

As we can see, this table is much cleaner than before, and variables are clearly labeled and understood, so it would be much easier to manipulate this data set than the previous one.

separate(data, col, into , sep = “”)

This function is used when many variables are in a single column and you want to separate them out. You can specify what separates the two variables using the “sep” field in the separate function. It will separate on any non alphanumeric character. Say you have two variables in a column separated by a period. This is what the initial table would look like.

```
table3 <- data.frame(
  x = c("a.b"),
  y = c(10000)
)
table3
```

```
##     x     y
## 1 a.b 10000
```

To separate variables a & b, we can use the separate function.

```
separate_tb3 <- separate(table3, col = x, c("c", "d"))
separate_tb3
```

```
##   c d     y
## 1 a b 10000
```

Here, we can see that the column variable “a.b”, which are two variables combined, are separated into their own columns, with the names that we specify in the function. A and B are used in place of variable names, but an example of this would be a variable such as Years and Months, if it were written in a concatenated manner such as “2017/8”. In this case, the separate function would separate at the “/”, since it is not alphanumeric.

unite(data, col, …, sep = “”)

This function is used to combine multiple columns into a single column. This would be used to combine messy data where one variable is stored across multiple columns. The sep argument can be used to specify how the columns will be separated. The default is an underscore. This is effectively the opposite of separate

Here we see a table with two column values, postgrads and undergrads.

```
tb4 <-  data.frame(
  School = c("UCLA", "Cal", "USC"),
  Undergrads = c(10000, 12000, 8000),
  Postgrads = c(4000, 6000, 3000)
  )

tb4
```

```
##   School Undergrads Postgrads
## 1   UCLA      10000      4000
## 2    Cal      12000      6000
## 3    USC       8000      3000
```

Currently, this is considered to be a Tidy data set. However, for the purposes of demonstrating this function, we will combine the Undergrads and postgrads column into a single one, separated by “:”. We

```
unite_table <- unite(tb4, "Undergrads:Postgrads", 2:3, sep = ":")
unite_table
```

```
##   School Undergrads:Postgrads
## 1   UCLA           10000:4000
## 2    Cal           12000:6000
## 3    USC            8000:3000
```

We can see that this combined the two columns into a single one using the separator specified. This can be used to combine columns that all contain a single variable, or to concatenate variables like shown above.

Conclusion & Take Home Message

The Tidyr package can be used to take messy data sets and convert them to “tidy” data sets that are more easily understood, visualized, manipulated, and modeled. Tidying data is a crucial step in any data science application, and learning how to use packages such as Tidyr is an invaluable skill. These are four basic functions in Tidyr, and there are more for other cases of messy data. These four functions help fix messy

data that violates the three main princples of tidy data, as they directly fix the problems that violate these mistakes. Tidy data is very important to the rest of the data science process and should be considered a fundamental skill, moreso than working with already-tidy data, since most data we will encounter will not be tidy.

References

- R Documentation
- Wikipedia
- Wickham Hadley Documentation
- Basic Tidyr Examples
- Data Processing with dplyr and tidyr
- Data Tidying
- Easily Tidy Data with `spread()` and `gather()` Functions