# Post02

*Shangchen Jiang*

*2017.11.30*

# Topic: Data Manipulation with dplyr() & tidyr()

## Introduction:

This is a “blog post” talking about the dplyr() and tidyr() packages. We’ve already talked about dplyr() during lectures and labs. So I’ll just kind of review a little bit of it, and talk more on tidyr(). The tidyr() package is doing similar things as dplyr() and it’s designed specifically for data tidying and works well with dplyr() data pipelines.

## Motivation:

It’s kind of cool working with the dplyr() since it makes it so easy for users to manipulate the data by selecting certain rows or columns from a data frame, or adding a few more rows or columns. It works fast and efficiently. I came across the tidyr() package by chance and thought it would be interesting to explore what tidyr() can do. Some says dplyr() and tidyr() can basically be thought of as a single package. We’ll see.

```r
#let's load the packages we are going to use first
library(readr)
#note that we should load the plyr() packge before dplyr() to avoid problems
library(tidyr); library(dplyr)
```

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

```
## 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
```

```r
library(ggplot2)
```

## Let’s prepare some data from the R built-in datasets. I chose to use the airquality of New York City (one of the built-in datasets)

```r
#here we are just using the R built-in dataset "airquality"
data("airquality")
#and we can see the summary stats of this dataset
summary(airquality)
```

```
##      Ozone           Solar.R           Wind             Temp
##  Min.   :  1.00   Min.   :  7.0   Min.   : 1.700   Min.   :56.00
##  1st Qu.: 18.00   1st Qu.:115.8   1st Qu.: 7.400   1st Qu.:72.00
##  Median : 31.50   Median :205.0   Median : 9.700   Median :79.00
##  Mean   : 42.13   Mean   :185.9   Mean   : 9.958   Mean   :77.88
##  3rd Qu.: 63.25   3rd Qu.:258.8   3rd Qu.:11.500   3rd Qu.:85.00
##  Max.   :168.00   Max.   :334.0   Max.   :20.700   Max.   :97.00
##  NA's   :37       NA's   :7
##      Month            Day
##  Min.   :5.000   Min.   : 1.0
##  1st Qu.:6.000   1st Qu.: 8.0
##  Median :7.000   Median :16.0
##  Mean   :6.993   Mean   :15.8
##  3rd Qu.:8.000   3rd Qu.:23.0
##  Max.   :9.000   Max.   :31.0
##
```

# Basics:

## Let’s first review a little bit we’ve learned done class on dplyr()

## Here’s how we can use the dplyr() package to sort out parts of the data frame according to our specifications. Firstly we are going to use the filter() function.

```
#say we only want to see the airquality in June. So we use the filter function in dplyr() package to filter out al
l the data from June
june_airquality <- filter(airquality, Month == 6)
head(june_airquality, 10)
```

```
##    Ozone Solar.R Wind Temp Month Day
## 1     NA     286  8.6   78     6   1
## 2     NA     287  9.7   74     6   2
## 3     NA     242 16.1   67     6   3
## 4     NA     186  9.2   84     6   4
## 5     NA     220  8.6   85     6   5
## 6     NA     264 14.3   79     6   6
## 7     29     127  9.7   82     6   7
## 8     NA     273  6.9   87     6   8
## 9     71     291 13.8   90     6   9
## 10    39     323 11.5   87     6  10
```

## We can also sort the data in June according to the level of ozone in decreasing order so that we can see what are the days where the ozone level is the highest. Here we are using the arrange() function.

```
#here we want to use the arrange function to do the job
ozone_rank <- arrange(june_airquality, desc(Ozone))
head(ozone_rank, 10)
```

```
##    Ozone Solar.R Wind Temp Month Day
## 1     71     291 13.8   90     6   9
## 2     39     323 11.5   87     6  10
## 3     37     284 20.7   72     6  17
## 4     29     127  9.7   82     6   7
## 5     23     148  8.0   82     6  13
## 6     21     191 14.9   77     6  16
## 7     20      37  9.2   65     6  18
## 8     13     137 10.3   76     6  20
## 9     12     120 11.5   73     6  19
## 10    NA     286  8.6   78     6   1
```

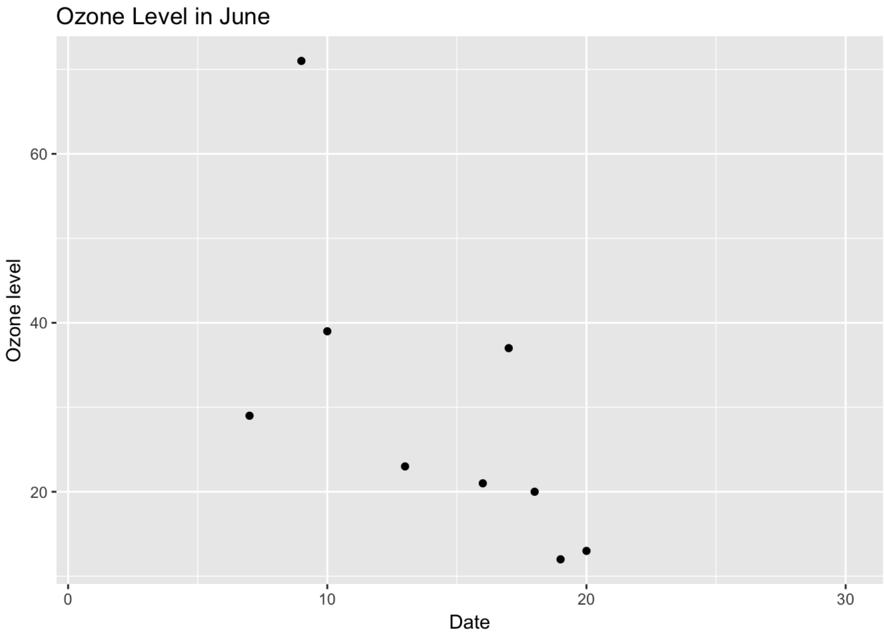
## It might be annoying to keep all the data you don’t need around. As we are only interested in the ozone level, let’s just look at it only, with respect to its date of recording. This is the time when we call the select() funtion

```
#we can select() the only things we want from the data frame
just_ozone <- select(june_airquality, Ozone, Month, Day)
head(just_ozone, 10)
```

```
##    Ozone Month Day
## 1     NA     6   1
## 2     NA     6   2
## 3     NA     6   3
## 4     NA     6   4
## 5     NA     6   5
## 6     NA     6   6
## 7     29     6   7
## 8     NA     6   8
## 9     71     6   9
## 10    39     6  10
```

```
#and we can just use ggplot to construct a relationship graph between the data and the ozone level
ggplot(just_ozone, aes(y = just_ozone$Ozone, x = just_ozone$Day)) +
  geom_point()+
  ggtitle('Ozone Level in June')+
  labs(x = 'Date', y = 'Ozone level')
```

```
## Warning: Removed 21 rows containing missing values (geom_point).
```




It’s plain to see that with the dplyr() package, it’s so easy for us to manipulate the data on hand. Aside from the filter(), arrange(), and select() functions, we’ve also learnt group_by(), summarise(), mutate() which are all very handy and effortless to work with!

## *NEW!* Now tidyr()!

Now we are going to talk about tidyr() and see what it can do to data frames. Now let’s try thr USArrests package where we can see the number of arrests for each category of crime in each state

```
#load the data first
data("USArrests")
#for easier manipulation, let's add the state names in a column
USArrests$State <- rownames(USArrests)
USArrests <- USArrests[, c(5, 1:4)]
#we can also see the summary stats of this data set
summary(USArrests)
```

```
##     State               Murder          Assault         UrbanPop
##  Length:50          Min.   : 0.800   Min.   : 45.0   Min.   :32.00
##  Class :character   1st Qu.: 4.075   1st Qu.:109.0   1st Qu.:54.50
##  Mode  :character   Median : 7.250   Median :159.0   Median :66.00
##                     Mean   : 7.788   Mean   :170.8   Mean   :65.54
##                     3rd Qu.:11.250   3rd Qu.:249.0   3rd Qu.:77.75
##                     Max.   :17.400   Max.   :337.0   Max.   :91.00
##       Rape
##  Min.   : 7.30
##  1st Qu.:15.07
##  Median :20.10
##  Mean   :21.23
##  3rd Qu.:26.18
##  Max.   :46.00
```

Firstly, let’s look at the gather() function. The gather() function takes in a data frame and reshape it from wide format into long format. Let’s try it on our arrest data.

```
long_arrest <- USArrests %>% gather(Type, Number, -State)
head(long_arrest, 10)
```

```
##          State   Type Number
## 1      Alabama Murder   13.2
## 2       Alaska Murder   10.0
## 3      Arizona Murder    8.1
## 4     Arkansas Murder    8.8
## 5   California Murder    9.0
## 6     Colorado Murder    7.9
## 7  Connecticut Murder    3.3
## 8     Delaware Murder    5.9
## 9      Florida Murder   15.4
## 10     Georgia Murder   17.4
```

Note that this result show that the gather() function sort of shifts the data frame around, and creates a new variable "Type" instead of listing out the four types individually.

## the gather() function leads us to its complement - spread() function, which turns the long format back into wide format. We can use this spread() function to turn our gathered version again back to wide format.

```
wide_arrest <-long_arrest %>% spread(Type, Number)
head(wide_arrest, 10)
```

```
##          State Assault Murder Rape UrbanPop
## 1      Alabama     236   13.2 21.2       58
## 2       Alaska     263   10.0 44.5       48
## 3      Arizona     294    8.1 31.0       80
## 4     Arkansas     190    8.8 19.5       50
## 5   California     276    9.0 40.6       91
## 6     Colorado     204    7.9 38.7       78
## 7  Connecticut     110    3.3 11.1       77
## 8     Delaware     238    5.9 15.8       72
## 9      Florida     335   15.4 31.9       80
## 10     Georgia     211   17.4 25.8       60
```

note here we have our wide version back.

## Secondly (actually, thirdly, since it's our third function from tidyr()), we are going to look at the unite() function. Using unite(), we can combine two variables into one single variable. We can try it on our data.

```
total_arrest <- USArrests %>% unite(Total, Murder, Assault, Rape, sep = ' ')
head(total_arrest, 10)
```

```
##                   State          Total UrbanPop
## Alabama         Alabama 13.2 236 21.2       58
## Alaska           Alaska   10 263 44.5       48
## Arizona         Arizona    8.1 294 31       80
## Arkansas       Arkansas  8.8 190 19.5       50
## California   California    9 276 40.6       91
## Colorado       Colorado  7.9 204 38.7       78
## Connecticut Connecticut  3.3 110 11.1       77
## Delaware       Delaware  5.9 238 15.8       72
## Florida         Florida 15.4 335 31.9       80
## Georgia         Georgia 17.4 211 25.8       60
```

It's very easy to see that although we are able to combine the variables, the numbers were just placed one by another and do not add up. So it's not a very efficient way to combine numeric data. It perhaps works better with letters and sentences that can be literally put together one after another.

## And we can guess what the next function is? Yeah, separate() to separate what we just "united".

```
sep_arrest <- total_arrest %>% separate(Total, c("Murder", "Assault", "Rape"), sep = ' ')
head(sep_arrest, 10)
```

```
##                   State Murder Assault Rape UrbanPop
## Alabama         Alabama   13.2     236 21.2       58
## Alaska           Alaska     10     263 44.5       48
## Arizona         Arizona    8.1     294   31       80
## Arkansas       Arkansas    8.8     190 19.5       50
## California   California      9     276 40.6       91
## Colorado       Colorado    7.9     204 38.7       78
## Connecticut Connecticut    3.3     110 11.1       77
## Delaware       Delaware    5.9     238 15.8       72
## Florida         Florida   15.4     335 31.9       80
## Georgia         Georgia   17.4     211 25.8       60
```

now with separate() we have our original version back!

## Take-home message:

Now we’ve seen the tidyr() package and what its functions do to a data frame. We can conclude that tidyr() is indeed useful in some ways, but it can not do everything. We might still have to use dplyr() or apply both packages to complete data processing. When trying to manipulate a set of data, it’s better to understand what is the goal we are going to achieve, and select the best packages and functions to do the job. But after all, it’s never a bad idea to learn something new!

## Reference:

1.https://cran.r-project.org/web/packages/dplyr/vignettes/dplyr.html

2.https://cran.r-project.org/web/packages/tidyr/tidyr.pdf

3.https://rpubs.com/bradleyboehmke/data_wrangling

4.http://tclavelle.github.io/dplyr-tidyr-tutorial/

5.http://tidyr.tidyverse.org/

6.https://www.r-bloggers.com/data-manipulation-with-tidyr/

7.https://www.rdocumentation.org/packages/tidyr/versions/0.7.2/topics/gather

8.https://www.rdocumentation.org/packages/tidyr/versions/0.7.2/topics/spread

9.https://www.rdocumentation.org/packages/tidyr/versions/0.7.2/topics/unite

10.https://www.rdocumentation.org/packages/tidyr/versions/0.7.2/topics/separate