Introduce Cleansing Techniques

Hanh Nguyen

We often wish to tidy and reshape a dataset so that we can create certain plots. Here I introduce the two packages **tidyr** and **reshape2** to help the need and also to see how functions in **tidyr** and **reshape2** overlap and differ.
We first compare the functions gather(), separate() and spread(), from tidyr, with the functions melt(), colsplit() and dcast(), from reshape2.

The original dataset

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

tidyr package

gather {tidyr}: takes multiple columns and collapses into key-value pairs, duplicating all other columns as needed. You use gather() when you notice that you have columns that are not variables.
Simply put, gather() takes wide-format data and turns it into long-format data

```
iris.tidyr <- iris %>%
  gather(key,value,-Species)
```

```
##   Species          key value
## 1  setosa Sepal.Length   5.1
## 2  setosa Sepal.Length   4.9
## 3  setosa Sepal.Length   4.7
## 4  setosa Sepal.Length   4.6
## 5  setosa Sepal.Length   5.0
## 6  setosa Sepal.Length   5.4
```

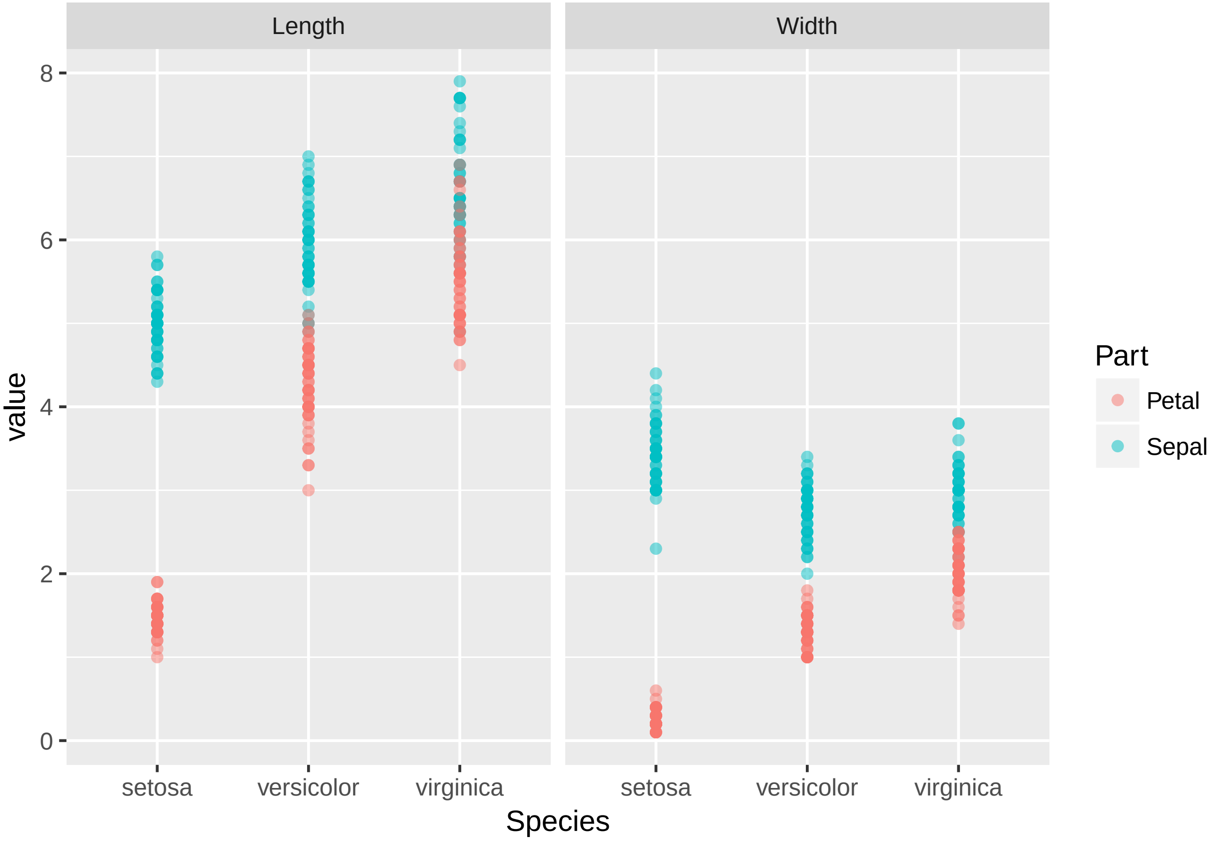
Our next step is to split the column key into two different columns: Part of a flower (Sepal or Petal) and Measure of that part (Length or Width), hence we use separate() function.
separate {tidyr}: turns a single character column into multiple columns.

```
iris.tidyr <- iris %>%
  gather(key,value,-Species) %>%
  separate(key,into=c("Part","Measure"),sep="\\.")
```

```
##   Species  Part Measure value
## 1  setosa Sepal  Length   5.1
## 2  setosa Sepal  Length   4.9
## 3  setosa Sepal  Length   4.7
## 4  setosa Sepal  Length   4.6
## 5  setosa Sepal  Length   5.0
## 6  setosa Sepal  Length   5.4
```

With this dataset structure, we now can create a plot as shown below.

```
iris.tidyr %>%
  ggplot(aes(x = Species, y = value, col = Part)) +
  geom_point(alpha =0.5) +
  facet_grid(. ~ Measure)
```



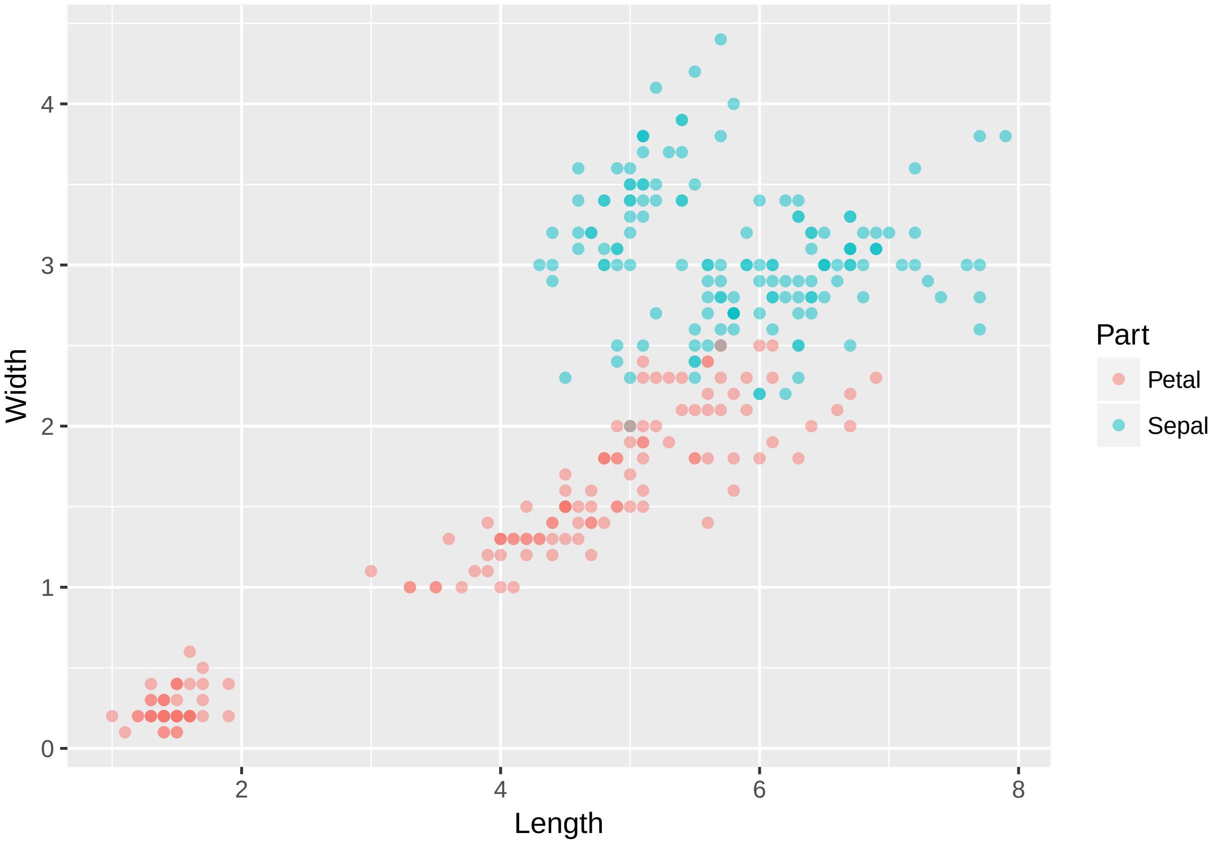
spread {tidyr}: spreads a key-value pair across multiple columns. In contrast to gather(), spread() takes long-format data and turns it into wide-format data.

```
iris$Flower <- 1:nrow(iris)
iris.tidyr <- iris %>%
  gather(key, value, - Species, - Flower) %>%
  separate(key, c("Part", "Measure"), "\\.") %>%
  spread(Measure, value)
```

```
##   Species Flower  Part Length Width
## 1  setosa      1 Petal    1.4   0.2
## 2  setosa      1 Sepal    5.1   3.5
## 3  setosa      2 Petal    1.4   0.2
## 4  setosa      2 Sepal    4.9   3.0
## 5  setosa      3 Petal    1.3   0.2
## 6  setosa      3 Sepal    4.7   3.2
```

With this dataset structure, we now can create a plot as shown below.

```
iris.tidyr %>%
  ggplot(aes(x=Length,y=Width,col=Part)) +
  geom_point(alpha=0.5)
```



reshape2 package

melt {reshape2}: converts an object into a molten data frame, giving same result with the gather() function from tidyr.
However, gather() cannot handle matrices or arrays, while melt() can!

```
iris.re <- iris %>%
  melt(id.vars="Species")
```

```
##   Species     variable value
## 1  setosa Sepal.Length   5.1
## 2  setosa Sepal.Length   4.9
## 3  setosa Sepal.Length   4.7
## 4  setosa Sepal.Length   4.6
## 5  setosa Sepal.Length   5.0
## 6  setosa Sepal.Length   5.4
```

colsplit {reshape2}: splits variable names that is a combination of multiple variables.
Again, we can achieve the same result with separate() function from tidyr, however, colsplit() operates only on a single column so we use cbind() to insert the new two columns in the data frame. While separate() performs all the operation at once.

```
iris$Flower <- 1:nrow(iris)
iris.re <- iris %>%
  melt(id.vars=c("Species","Flower"))
iris.re = cbind(Species=iris.re[,1],
              Flower=iris.re[,2],
                colsplit(iris.re[,3],"\\.",c("Part","Measure")),
                value=iris.re[,4])
```

```
##   Species Flower  Part Measure value
## 1  setosa      1 Sepal  Length   5.1
## 2  setosa      2 Sepal  Length   4.9
## 3  setosa      3 Sepal  Length   4.7
## 4  setosa      4 Sepal  Length   4.6
## 5  setosa      5 Sepal  Length   5.0
## 6  setosa      6 Sepal  Length   5.4
```

Again, the same result produced by spread() from tidyr can be obtained using dcast() from reshape2 by specifying the correct formula.
cast {reshape2}: casts a molten data frame into an array or data frame.

```
iris.re = dcast(iris.re, formula=Flower+Species+Part ~Measure)
```

```
##   Flower Species  Part Length Width
## 1      1  setosa Petal    1.4   0.2
## 2      1  setosa Sepal    5.1   3.5
## 3      2  setosa Petal    1.4   0.2
## 4      2  setosa Sepal    4.9   3.0
## 5      3  setosa Petal    1.3   0.2
## 6      3  setosa Sepal    4.7   3.2
```

Sources:
https://blog.rstudio.org/2014/07/22/introducing-tidyr/
http://www.milanor.net/blog/reshape-data-r-tidyr-vs-reshape2/