Manipulación de datos - tidyr

Josman

14 de octubre de 2014

Una herramienta de gran utilidad para la manipulación de datos en R es el paquete tidyr. Se les llama datos "tidy" a aquellos con los que es fácil trabajar. Este tipo de manipulaciones nos servirá mucho para poder visualizar los datos de manera más eficiente. Debemos recordar que lo más importante de los datos tidy es que:

- Cada columna es una variable.
- Canda renglón es una observación.

Para convertir datos desordenados en datos tidy, primero debemos identificar las variables en nuestra base de datos, después usamos las herramientas del paquete para moverlos en columnas. El paquete tidyr provee tres principales funciones para el arreglo de datos: gather(), separate() y spread().

Función gather()

La función gather() toma multiples columnas y las reune indicando valores clave, toma datos anchos para convertirlos en datos largos. A continuación mostramos un ejemplo de cómo utilizarlo. Se presenta un experimento en el que se dieron dos drogas distintas a 3 personas y se registró su ritmo cardiaco:

```
library(tidyr)
library(dplyr)

messy <- data.frame(
  name = c("Wilbur", "Petunia", "Gregory"),
  a = c(67, 80, 64),
  b = c(56, 90, 50)
)
messy
```

```
##      name  a  b
## 1  Wilbur 67 56
## 2 Petunia 80 90
## 3 Gregory 64 50
```

Sin pensar en cómo está escrito en la base de datos, podemos identificar 3 variables: nombre, droga y ritmo cardiaco; pero únicamente el nombre está escrito como columna. Usaremos gather() para arreglar los datos como queremos que se presenten:

```
messy %>%
  gather(drug, heartrate, a:b)
```

```
##      name drug heartrate
## 1  Wilbur    a        67
## 2 Petunia    a        80
## 3 Gregory    a        64
## 4  Wilbur    b        56
## 5 Petunia    b        90
## 6 Gregory    b        50
```

Función separate()

A veces, dos variables pueden encontrarse agrupadas en una misma columna, separate() nos permite dividir los datos. A continuación mostramos datos del tiempo que gastan las personas en sus teléfonos ya sea en sus casas o en el trabajo. A cada persona se le asignó de manera aleatoria al grupo de tratamiento o al grupo de control.

```
set.seed(10)
messy <- data.frame(
  id = 1:4,
  trt = sample(rep(c('control', 'treatment'), each = 2)),
  work.T1 = runif(4),
  home.T1 = runif(4),
  work.T2 = runif(4),
  home.T2 = runif(4)
)

messy
```

```
##   id       trt    work.T1   home.T1   work.T2    home.T2
## 1  1 treatment 0.08513597 0.6158293 0.1135090 0.05190332
## 2  2   control 0.22543662 0.4296715 0.5959253 0.26417767
## 3  3 treatment 0.27453052 0.6516557 0.3580500 0.39879073
## 4  4   control 0.27230507 0.5677378 0.4288094 0.83613414
```

Primero usaremos la función gather() para comvertir las columnas work.T1, home.T1, work.T2 and home.T2 en parejas clave-valor del tiempo.

```
tidier <- messy %>%
  gather(key, time, -id, -trt)
tidier %>% head(8)
```

```
##   id       trt     key       time
## 1  1 treatment work.T1 0.08513597
## 2  2   control work.T1 0.22543662
## 3  3 treatment work.T1 0.27453052
## 4  4   control work.T1 0.27230507
## 5  1 treatment home.T1 0.61582931
## 6  2   control home.T1 0.42967153
## 7  3 treatment home.T1 0.65165567
## 8  4   control home.T1 0.56773775
```

Ahora usaremos separate() para dividir las claves en lugar y tiempo.

```
tidy <- tidier %>%
  separate(key, into = c("location", "time"), sep = "\\.")
tidy %>% head(8)
```

```
##   id       trt location time       time
## 1  1 treatment     work   T1 0.08513597
## 2  2   control     work   T1 0.22543662
## 3  3 treatment     work   T1 0.27453052
```

```
## 4  4   control     work   T1 0.27230507
## 5  1 treatment     home   T1 0.61582931
## 6  2   control     home   T1 0.42967153
## 7  3 treatment     home   T1 0.65165567
## 8  4   control     home   T1 0.56773775
```

Función spread()

La función spread() toma dos columnas (clave-valor) y las extiende en múltiples columnas, convirtiendo los datos largos en datos anchos. Mostraremos un pequeño ejemplo de finanzas:

```
stocks <- data.frame(
  time = as.Date('2009-01-01') + 0:9,
  X = rnorm(10, 0, 1),
  Y = rnorm(10, 0, 2),
  Z = rnorm(10, 0, 4)
)

stocks
```

```
##          time           X          Y          Z
## 1  2009-01-01  1.10177950 -1.1926213 -7.4149618
## 2  2009-01-02  0.75578151 -4.3705737 -0.3117843
## 3  2009-01-03 -0.23823356 -1.3497319  3.8742654
## 4  2009-01-04  0.98744470 -4.2381224  0.7397038
## 5  2009-01-05  0.74139013 -2.5303960 -5.5197743
## 6  2009-01-06  0.08934727 -0.7473231 -5.7420574
## 7  2009-01-07 -0.95494386 -1.3751109  1.4483489
## 8  2009-01-08 -0.19515038 -1.7443177 -7.0363470
## 9  2009-01-09  0.92552126 -0.2035220 -1.2981760
## 10 2009-01-10  0.48297852 -0.5075611 -2.6062520
```

Primero usamos la función gather() para acomodar los datos.

```
stocksm <- stocks %>% gather(stock, price, -time)
stocksm %>% head(8)
```

```
##         time stock       price
## 1 2009-01-01     X  1.10177950
## 2 2009-01-02     X  0.75578151
## 3 2009-01-03     X -0.23823356
## 4 2009-01-04     X  0.98744470
## 5 2009-01-05     X  0.74139013
## 6 2009-01-06     X  0.08934727
## 7 2009-01-07     X -0.95494386
## 8 2009-01-08     X -0.19515038
```

Ahora que ya tenemos datos de este tipo, veamos qué podemos hacer con spread().

```
# Expandimos los datos según el tipo de stock
stocksm %>% spread(stock,price)
```

```
##          time           X          Y          Z
## 1  2009-01-01  1.10177950 -1.1926213 -7.4149618
## 2  2009-01-02  0.75578151 -4.3705737 -0.3117843
## 3  2009-01-03 -0.23823356 -1.3497319  3.8742654
## 4  2009-01-04  0.98744470 -4.2381224  0.7397038
## 5  2009-01-05  0.74139013 -2.5303960 -5.5197743
## 6  2009-01-06  0.08934727 -0.7473231 -5.7420574
## 7  2009-01-07 -0.95494386 -1.3751109  1.4483489
## 8  2009-01-08 -0.19515038 -1.7443177 -7.0363470
## 9  2009-01-09  0.92552126 -0.2035220 -1.2981760
## 10 2009-01-10  0.48297852 -0.5075611 -2.6062520
```

```
# Expandimos los datos según la fecha
stocksm %>% spread(time, price)
```

```
##   stock 2009-01-01 2009-01-02 2009-01-03 2009-01-04 2009-01-05  2009-01-06
## 1     X   1.101780  0.7557815 -0.2382336  0.9874447  0.7413901  0.08934727
## 2     Y  -1.192621 -4.3705737 -1.3497319 -4.2381224 -2.5303960 -0.74732311
## 3     Z  -7.414962 -0.3117843  3.8742654  0.7397038 -5.5197743 -5.74205745
##   2009-01-07 2009-01-08 2009-01-09 2009-01-10
## 1 -0.9549439 -0.1951504  0.9255213  0.4829785
## 2 -1.3751109 -1.7443177 -0.2035220 -0.5075611
## 3  1.4483489 -7.0363470 -1.2981760 -2.6062520
```

Con esto tenemos una introducción para crear bases de datos inteligentes que nos permitirán realizar las tareas con datos de manera más rápida y eficaz.