Data Wrangling

Martín Montané

04-07-2018

Objetivo

La clase anterior realizamos algunas de las etapas de un típico proyecto de Ciencia de Datos: leímos los datos y realizamos un breve análisis descriptivo, principalmente a través de algunas visualizaciones. Sin embargo, en la práctica esto no suele suceder de esta manera e inmediatamente después de leer los datos tenemos que realizar una serie de transformaciones que, como regla general, ocupa la mayor parte del tiempo. Este documento tiene el objetivo de cubrir las herramientas que **tidyverse** nos ofrece para realizar estas transformaciones, conocidas de manera coloquial como *Data Wrangling*.

Vamos a intentar cumplir este objetivo en dos partes. En primer lugar, vamos a pasar lista de las funciones que los van a acompañar de ahora en adelante para hacer las transformaciones de datos necesarias. En segundo lugar, vamos a usarlas para realizar el mismo procesamiento de datos que hice yo para transformar los datos tal como son descargados desde *Properati* hasta tener los datasets que trabajamos la clase pasada.

El dataset *gapminder*

En 2007 Hans Rosling (1948-2017), un médico sueco, dio una de las charlas Ted más famosas[1]. Su presentación mostraba la evolución de tres variables en el tiempo: PIB per cápita, expectativa de vida al nacer y población. Uno de los principales mensajes de su charla es que, aunque no lo notemos, el mundo ha mejorado considerablemente - y continúa haciéndolo. Vamos a trabajar con estos optimistas datos en nuestra clase de hoy.

R tiene una librería en la cual podemos cargar el dataset completo que presentó Rosling en aquella ocación. Sólo hay que escribir las siguientes líneas:

```
install.packages("gapminder")
```

```
library(gapminder) # Recordar que hay que tener instalado el paquete gapminder
print(gapminder) # Prueben ejecutar "gapminder" a secas, es decir sin el método print()
```

```
## # A tibble: 1,704 x 6
##    country     continent  year lifeExp      pop gdpPercap
##    <fct>       <fct>     <int>   <dbl>    <int>     <dbl>
##  1 Afghanistan Asia       1952    28.8  8425333      779.
##  2 Afghanistan Asia       1957    30.3  9240934      821.
##  3 Afghanistan Asia       1962    32.0 10267083      853.
##  4 Afghanistan Asia       1967    34.0 11537966      836.
##  5 Afghanistan Asia       1972    36.1 13079460      740.
##  6 Afghanistan Asia       1977    38.4 14880372      786.
##  7 Afghanistan Asia       1982    39.9 12881816      978.
##  8 Afghanistan Asia       1987    40.8 13867957      852.
##  9 Afghanistan Asia       1992    41.7 16317921      649.
## 10 Afghanistan Asia       1997    41.8 22227415      635.
## # ... with 1,694 more rows
```

[1] Una presentación sumamente interesante puede consultarse en https://www.youtube.com/watch?v=jbkSRLYSojo

```
# ¿Es distinto?
gapminder_df <- gapminder # Creamos un dataframe con los datos de gapminder
str(gapminder)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    1704 obs. of  6 variables:
##  $ country  : Factor w/ 142 levels "Afghanistan",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ continent: Factor w/ 5 levels "Africa","Americas",..: 3 3 3 3 3 3 3 3 3 3 ...
##  $ year     : int  1952 1957 1962 1967 1972 1977 1982 1987 1992 1997 ...
##  $ lifeExp  : num  28.8 30.3 32 34 36.1 ...
##  $ pop      : int  8425333 9240934 10267083 11537966 13079460 14880372 12881816 13867957 16317921 22
##  $ gdpPercap: num  779 821 853 836 740 ...
```

¿Qué clase de objeto es? Si leen en la parte superior de lo que devuelve R van a ver que dice *tibble* y no *Data Frame*. Un tibble es muy similar a un data frame a fines prácticos, con algunas diferencias en el método print(). Uno de esos detalles que nos reporta el tipo de vector. Por ejemplo, podemos ver que los vectores *country* y *continent* son factores, *year* y *pop* son integers (vectores númericos que toman solo números reales enteros) y *lifeExp* y *gdpPercap* son vectores double (dbl), es decír vectores númericos con números décimales. La descripción de las variables es relativamente obvia:

- country: Nombre de país
- continent: Nombre del continente
- year: año de la observación
- lifeExp: expectativa de vida al nacer (en años)
- pop: cantidad de habitantes
- gdpPercap: Producto Interno Bruto (PIB) por habitante

Transformaciones de los datos

Recordemos que precisamos de las funciones de **tidyverse** para transformar nuestros datos. Para esto, solo tenemos que aplicar el comando `require()` o `library()`, los dos cumplen con nuestro objetivo:

```
require(tidyverse) # Pueden usar library(tidyverse), el resultado debería ser el mismo.
```

Listo: comencemos con nuestra sesión de **Data wrangling**

Transformaciones básicas

Selección de columnas: select()

En el tutorial anterior ya introducimos de manera informal al método `select()`. El comando es realmente simple: debemos pasarle los nombres de las variables que deseamos retener. Conservar solo algunas de las variables de un Data Frame es una operación que se realiza muy frecuentemente, así que practiquemos con dos ejemplos. Es importante recordar el papel que cumple el **pipe** (%>%): todo lo que está antes es pasado a lo que le sigue para ser procesado.

```
gapminder_df %>% select(country) # Seleccionamos solo la variable de país
```

```
## # A tibble: 1,704 x 1
##    country
##    <fct>
##  1 Afghanistan
##  2 Afghanistan
##  3 Afghanistan
##  4 Afghanistan
```

```
##  5 Afghanistan
##  6 Afghanistan
##  7 Afghanistan
##  8 Afghanistan
##  9 Afghanistan
## 10 Afghanistan
## # ... with 1,694 more rows
```

```
gapminder_df %>% select(country, continent) # Seleccionamos las variables country y continent
```

```
## # A tibble: 1,704 x 2
##    country     continent
##    <fct>       <fct>
##  1 Afghanistan Asia
##  2 Afghanistan Asia
##  3 Afghanistan Asia
##  4 Afghanistan Asia
##  5 Afghanistan Asia
##  6 Afghanistan Asia
##  7 Afghanistan Asia
##  8 Afghanistan Asia
##  9 Afghanistan Asia
## 10 Afghanistan Asia
## # ... with 1,694 more rows
```

Como siempre, podemos asignar estos nuevos data frames a otros objetos mediante el método de asignación `<-`. Además, una función que nos va a servir de ahora en adelante es `unique()`. Esta función toma un *objeto* como argumento y devuleve un nuevo *objeto* que contiene los casos únicos. Bajandolo a tierra, junto con la creación de un nuevo objeto veamos cuantos países y continentes hay en nuestro dataset:

```
gapminder_subset <- gapminder_df %>% select(country, continent)
str(gapminder_subset)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    1704 obs. of  2 variables:
##  $ country  : Factor w/ 142 levels "Afghanistan",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ continent: Factor w/ 5 levels "Africa","Americas",..: 3 3 3 3 3 3 3 3 3 3 ...
```

```
paises <- unique(gapminder_subset$country)
length(paises) # Length() nos devuelve la cantidad de elementos que tiene un vector
```

```
## [1] 142
```

```
length(unique(gapminder_subset$continent)) # Podemos combinar las funciones
```

```
## [1] 5
```

Existen 142 países y 5 continentes en nuestro dataset.

Selección de casos: `filter()`

Cuando queramos analizar nuestros datos según ciertas características que tengan nuestras observaciones, es preciso poder seleccionar los casos según los valores que toman en una o más variables. **tidyverse** (en rigor, uno de sus paquetes: **dplyr**) nos ofrece el método `filter()` para realizar esta clase de transformaciones. Por ejemplo: ¿Cuáles son las observaciones que corresponden al año 2002?

```
gapminder_df %>%  filter(year == 2002)
```

```
## # A tibble: 142 x 6
```

```
##    country     continent  year lifeExp       pop gdpPercap
##    <fct>       <fct>     <int>   <dbl>     <int>     <dbl>
##  1 Afghanistan Asia       2002    42.1  25268405      727.
##  2 Albania     Europe     2002    75.7   3508512     4604.
##  3 Algeria     Africa     2002    71.0  31287142     5288.
##  4 Angola      Africa     2002    41.0  10866106     2773.
##  5 Argentina   Americas   2002    74.3  38331121     8798.
##  6 Australia   Oceania    2002    80.4  19546792    30688.
##  7 Austria     Europe     2002    79.0   8148312    32418.
##  8 Bahrain     Asia       2002    74.8    656397    23404.
##  9 Bangladesh  Asia       2002    62.0 135656790     1136.
## 10 Belgium     Europe     2002    78.3  10311970    30486.
## # ... with 132 more rows
```

Como podemos ver, este comando nos devuelve un tibble (o data frame) con las observaciones correspondientes al año 2002. Recordamos: **Las funciones que aplicamos a un objeto (en este caso gapminder) tienen que estar mediadas por lo que se conoce como pipe (%>%)**. Detengámonos para analizar qué fue lo que hicimos en mayor detalle.

La función filter toma **operadores lógicos** como argumentos. Estos operadores devuelven TRUE o FALSE dependiendo si una comparación se verifica o no. En nuestro caso particular, el operador lógico utilizado fue `==`, que singifica **exactamente igual a**. Puede parecer raro que usemos doble igual en lugar de un solo igual, pero recuerden que en R el `=` se encuentra reservado para la asignación, al igual que `<-`.

Utilizando el mismo operador lógico podemos hacer todavía más cosas. Por ejemplo, podemos filtrar con respecto a otras variables o **combinar condiciones de filtrado** mediante el mismo método

```
gapminder_df %>% filter(country == "Argentina")
```

```
## # A tibble: 12 x 6
##    country   continent  year lifeExp      pop gdpPercap
##    <fct>     <fct>     <int>   <dbl>    <int>     <dbl>
##  1 Argentina Americas   1952    62.5 17876956     5911.
##  2 Argentina Americas   1957    64.4 19610538     6857.
##  3 Argentina Americas   1962    65.1 21283783     7133.
##  4 Argentina Americas   1967    65.6 22934225     8053.
##  5 Argentina Americas   1972    67.1 24779799     9443.
##  6 Argentina Americas   1977    68.5 26983828    10079.
##  7 Argentina Americas   1982    69.9 29341374     8998.
##  8 Argentina Americas   1987    70.8 31620918     9140.
##  9 Argentina Americas   1992    71.9 33958947     9308.
## 10 Argentina Americas   1997    73.3 36203463    10967.
## 11 Argentina Americas   2002    74.3 38331121     8798.
## 12 Argentina Americas   2007    75.3 40301927    12779.
```

```
gapminder_df %>% filter(continent == "Americas")
```

```
## # A tibble: 300 x 6
##    country   continent  year lifeExp      pop gdpPercap
##    <fct>     <fct>     <int>   <dbl>    <int>     <dbl>
##  1 Argentina Americas   1952    62.5 17876956     5911.
##  2 Argentina Americas   1957    64.4 19610538     6857.
##  3 Argentina Americas   1962    65.1 21283783     7133.
##  4 Argentina Americas   1967    65.6 22934225     8053.
##  5 Argentina Americas   1972    67.1 24779799     9443.
##  6 Argentina Americas   1977    68.5 26983828    10079.
##  7 Argentina Americas   1982    69.9 29341374     8998.
```

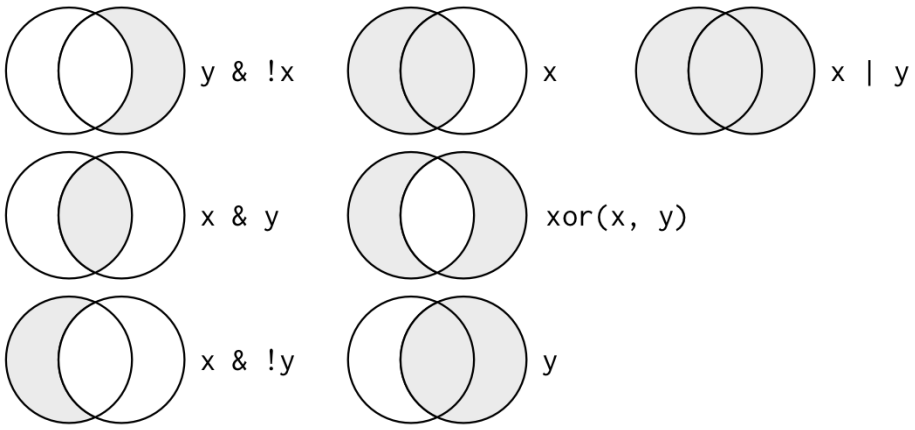


Figura 1: Representación gráfica de todas las operaciones lógicas posibles. Fuente: R for Data Science

```
##  8 Argentina Americas   1987    70.8 31620918     9140.
##  9 Argentina Americas   1992    71.9 33958947     9308.
## 10 Argentina Americas   1997    73.3 36203463    10967.
## # ... with 290 more rows
```

```
# Combinando filtros
gapminder_df %>% filter(country == "Argentina", year == 2007)
```

```
## # A tibble: 1 x 6
##   country   continent  year lifeExp      pop gdpPercap
##   <fct>     <fct>     <int>   <dbl>    <int>     <dbl>
## 1 Argentina Americas   2007    75.3 40301927    12779.
```

Como podemos ver en el último de los ejemplos, se pueden combinar más de un operador lógico separado por comas. Cada condición se concatena a la anterior como un AND lógico ¿Qué signfica esto? Que la función `select()` toma cada uno de los operadores lógicos y busca que todos se cumplan de manera SIMULTÁNEA. En nuestro último caso, filtramos los datos que correspondían tanto a Argentina como al año 2007, lo que devolvió un objeto de una sola fila: los valores correspondientes para Argentina en el año 2007.

La Figura 1 muestra todas las combinaciones booleanas (o lógicas) posibles dados dos conjuntos. La función `filter()` aplica el operador lógico `&` a cada una de las condiciones que imponemos. No se preocupen: podemos generar el resto de las condiciones booleanas usando **operadores booleanos** tales como `|`, que representa **OR**, o `!` que representa **NOT**, es decir que lo podemos usar para negar una condición. Si todo esto suena complejo es totalmente razonable. De cualquiera manera, algunos ejemplos (y mucha práctica) va a hacer que todo sea muy intuitivo. Veamos algunos ejemplos

```
# Datos de argentina pero que NO incluyan al año 2007
gapminder_df %>% filter(country == "Argentina", !year == 2007) # ! representa la negación
```

```
## # A tibble: 11 x 6
##    country   continent  year lifeExp      pop gdpPercap
##    <fct>     <fct>     <int>   <dbl>    <int>     <dbl>
##  1 Argentina Americas   1952    62.5 17876956     5911.
##  2 Argentina Americas   1957    64.4 19610538     6857.
##  3 Argentina Americas   1962    65.1 21283783     7133.
##  4 Argentina Americas   1967    65.6 22934225     8053.
##  5 Argentina Americas   1972    67.1 24779799     9443.
##  6 Argentina Americas   1977    68.5 26983828    10079.
##  7 Argentina Americas   1982    69.9 29341374     8998.
##  8 Argentina Americas   1987    70.8 31620918     9140.
```

```
##  9 Argentina Americas   1992    71.9 33958947     9308.
## 10 Argentina Americas   1997    73.3 36203463    10967.
## 11 Argentina Americas   2002    74.3 38331121     8798.
```

```
# Uno de los inconvenientes cuando queremos filtrar por más de un criterio de una misma variable
# es que tenemos que hacer lo siguiente
gapminder_df %>% filter(country == "Argentina", year == 2002 | year == 2007)
```

```
## # A tibble: 2 x 6
##   country   continent  year lifeExp      pop gdpPercap
##   <fct>     <fct>     <int>   <dbl>    <int>     <dbl>
## 1 Argentina Americas   2002    74.3 38331121     8798.
## 2 Argentina Americas   2007    75.3 40301927    12779.
```

Bastante más simple que en la explicación anterior ¿No? Vamos a complejizarlo levemente introduciendo al operador **%in%**. Este operador *%in%* es muy útil para matchear múltiples condiciones de manera simple. Lo que hace es devolver TRUE en todos los elementos de un vector que cumplen con alguno de los valores contenidos en el vector de la derecha. En el siguiente caso devuelve las observaciones que corresponden a los años 2002 o 2007.

```
# Podemos solucionarlo mediante el siguiente método
gapminder_df %>% filter(year %in% c(2002,2007))
```

```
## # A tibble: 284 x 6
##    country     continent  year lifeExp      pop gdpPercap
##    <fct>       <fct>     <int>   <dbl>    <int>     <dbl>
##  1 Afghanistan Asia       2002    42.1 25268405      727.
##  2 Afghanistan Asia       2007    43.8 31889923      975.
##  3 Albania     Europe     2002    75.7  3508512     4604.
##  4 Albania     Europe     2007    76.4  3600523     5937.
##  5 Algeria     Africa     2002    71.0 31287142     5288.
##  6 Algeria     Africa     2007    72.3 33333216     6223.
##  7 Angola      Africa     2002    41.0 10866106     2773.
##  8 Angola      Africa     2007    42.7 12420476     4797.
##  9 Argentina   Americas   2002    74.3 38331121     8798.
## 10 Argentina   Americas   2007    75.3 40301927    12779.
## # ... with 274 more rows
```

```
# Es una forma de concatenar condiciones de tipo | (OR)
gapminder_df %>% filter(year == 2002 | year == 2007) # Da el mismo resultado
```

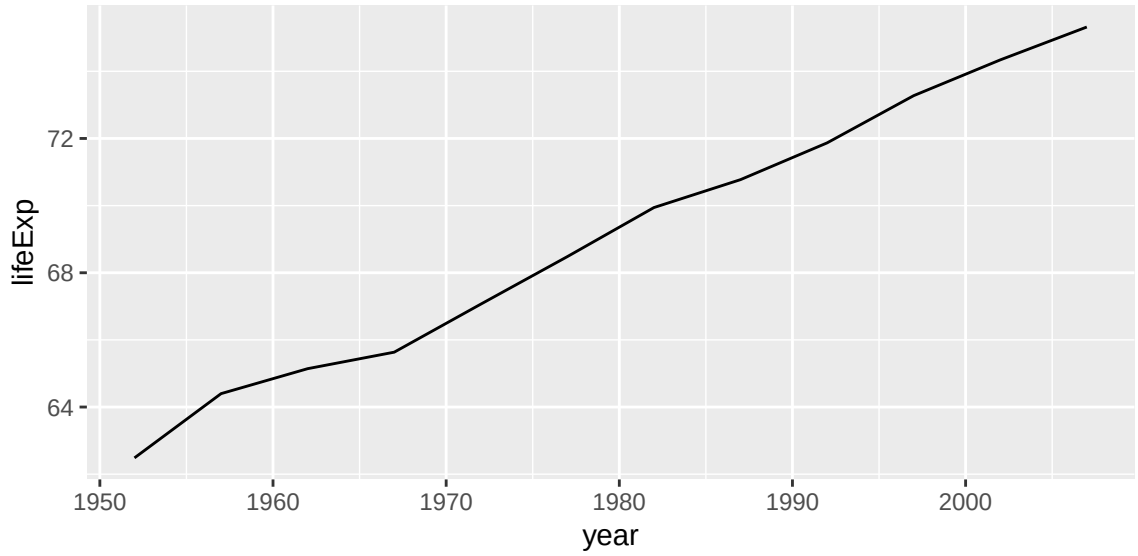
```
## # A tibble: 284 x 6
##    country     continent  year lifeExp      pop gdpPercap
##    <fct>       <fct>     <int>   <dbl>    <int>     <dbl>
##  1 Afghanistan Asia       2002    42.1 25268405      727.
##  2 Afghanistan Asia       2007    43.8 31889923      975.
##  3 Albania     Europe     2002    75.7  3508512     4604.
##  4 Albania     Europe     2007    76.4  3600523     5937.
##  5 Algeria     Africa     2002    71.0 31287142     5288.
##  6 Algeria     Africa     2007    72.3 33333216     6223.
##  7 Angola      Africa     2002    41.0 10866106     2773.
##  8 Angola      Africa     2007    42.7 12420476     4797.
##  9 Argentina   Americas   2002    74.3 38331121     8798.
## 10 Argentina   Americas   2007    75.3 40301927    12779.
## # ... with 274 more rows
```

Hagamos algo útil con lo que aprendimos hasta ahora, aun haciendo referencia a una librería que todavía no

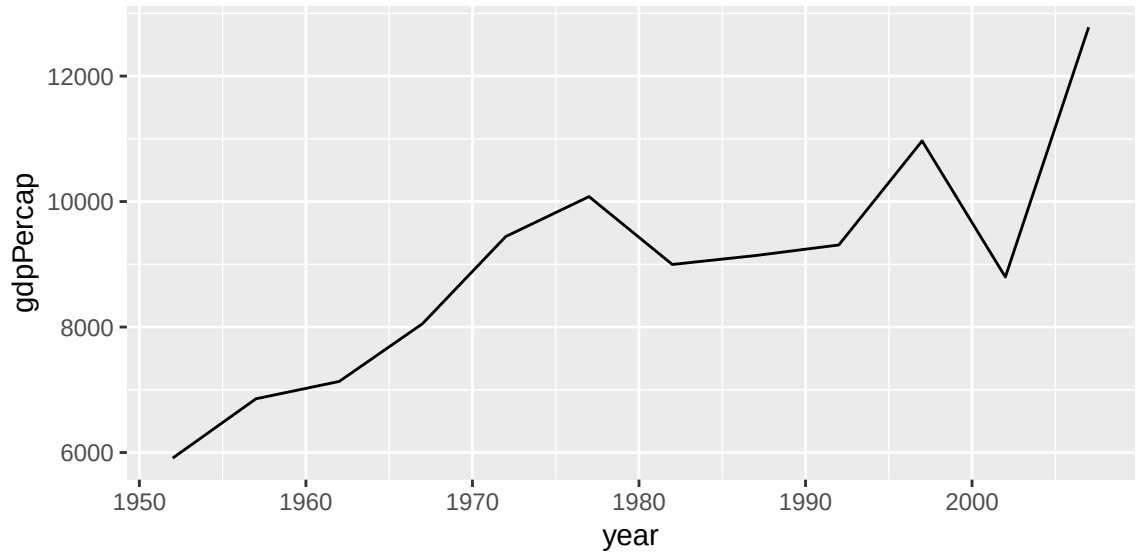
usamos como **ggplot**. Este paquete es uno de los más utilizados para realizar gráficos y funciona de una manera similar a **tmap**, el paquete que vimos por arriba en la anterior clase.

```
gapminder_argentina <- gapminder_df %>% filter(country == "Argentina")
```

```
# Expectativa de vida al nacer en Argentina 1952-2007
ggplot(gapminder_argentina) + geom_line(aes(x = year, y = lifeExp))
```




```
# PIB per cápita en Argentina 1952-2007
ggplot(gapminder_argentina) + geom_line(aes(x = year, y = gdpPercap))
```



Creo que podemos sacar nuestras primeras conclusiones para Argentina: la expectativa de vida parece haber crecido bastante estable durante todo el período bajo análisis, pero la evolución del PIB per cápita bien puede parecerse a lo que registra un sismógrafo !

Ordenando: la función arrange()

La función *arrange* nos permite ordenar un dataset en orden ascendente o descendente en base a los valores de las variables. Está parte del proceso suele ser relevante para algunas transformaciones de datos y para la inspección visual de valores extremos.

```
gapminder_df %>% arrange(lifeExp) # En sentido ascendente (del valor más bajo al más alto)
```

```
## # A tibble: 1,704 x 6
##    country      continent  year lifeExp     pop gdpPercap
##    <fct>        <fct>     <int>   <dbl>   <int>     <dbl>
##  1 Rwanda       Africa     1992    23.6 7290203      737.
##  2 Afghanistan  Asia       1952    28.8 8425333      779.
##  3 Gambia       Africa     1952    30    284320      485.
##  4 Angola       Africa     1952    30.0 4232095     3521.
##  5 Sierra Leone Africa     1952    30.3 2143249      880.
##  6 Afghanistan  Asia       1957    30.3 9240934      821.
##  7 Cambodia     Asia       1977    31.2 6978607      525.
##  8 Mozambique   Africa     1952    31.3 6446316      469.
##  9 Sierra Leone Africa     1957    31.6 2295678     1004.
## 10 Burkina Faso Africa     1952    32.0 4469979      543.
## # ... with 1,694 more rows
```

```
gapminder_df %>% arrange(desc(lifeExp)) # En sentido descendente (del valor más alto al más bajo)
```

```
## # A tibble: 1,704 x 6
##    country          continent  year lifeExp       pop gdpPercap
##    <fct>            <fct>     <int>   <dbl>     <int>     <dbl>
##  1 Japan            Asia       2007    82.6 127467972    31656.
##  2 Hong Kong, China Asia       2007    82.2   6980412    39725.
##  3 Japan            Asia       2002    82   127065841    28605.
##  4 Iceland          Europe     2007    81.8    301931    36181.
##  5 Switzerland      Europe     2007    81.7   7554661    37506.
##  6 Hong Kong, China Asia       2002    81.5   6762476    30209.
##  7 Australia        Oceania    2007    81.2  20434176    34435.
##  8 Spain            Europe     2007    80.9  40448191    28821.
##  9 Sweden           Europe     2007    80.9   9031088    33860.
## 10 Israel           Asia       2007    80.7   6426679    25523.
## # ... with 1,694 more rows
```

Veamos una de las principales funcionalidades de tidyverse al combinar algunos de los comandos que aprendimos hasta ahora. Vamos a filtrar el dataset para el continente de América y el año 2007 y luego (recordar: está a la derecha del último **pipe**) ordenar las observaciones según la expectativa de vida al nacer, de mayor a menor:

```
gapminder_df %>% filter(continent == "Americas", year == 2007) %>% arrange(desc(lifeExp))
```

```
## # A tibble: 25 x 6
##    country       continent  year lifeExp       pop gdpPercap
##    <fct>         <fct>     <int>   <dbl>     <int>     <dbl>
##  1 Canada        Americas   2007    80.7  33390141    36319.
##  2 Costa Rica    Americas   2007    78.8   4133884     9645.
##  3 Puerto Rico   Americas   2007    78.7   3942491    19329.
##  4 Chile         Americas   2007    78.6  16284741    13172.
##  5 Cuba          Americas   2007    78.3  11416987     8948.
##  6 United States Americas   2007    78.2 301139947    42952.
##  7 Uruguay       Americas   2007    76.4   3447496    10611.
```

```
##  8 Mexico        Americas   2007    76.2 108700891    11978.
##  9 Panama        Americas   2007    75.5   3242173     9809.
## 10 Argentina     Americas   2007    75.3  40301927    12779.
## # ... with 15 more rows
```

Creando y modificando variables: mutate()

Crear nuevas variables en base a los valores de otras variables que ya existen suele ser una parte necesaria para enriquecer el análisis. En el marco de **tidyverse** la forma de lograrlo es a través del *verb* (verb es tan solo otra forma de llamar a las funciones en el contexto de **tidyverse**) mutate(). Imagemos, por ejemplo, que queremos la población medida en millones de personas para hacer más fácil su lectura:

```
new_gapminder <- gapminder %>% mutate(pop = pop / 1000000)
head(new_gapminder, n = 3) # Head nos permite ver solo una determinada cantidad de filas
```

```
## # A tibble: 3 x 6
##   country     continent  year lifeExp   pop gdpPercap
##   <fct>       <fct>     <int>   <dbl> <dbl>     <dbl>
## 1 Afghanistan Asia       1952    28.8  8.43      779.
## 2 Afghanistan Asia       1957    30.3  9.24      821.
## 3 Afghanistan Asia       1962    32.0 10.3       853.
```

La sintaxis es simple: del lado izquierdo de la igualdad escribimos el nombre de la variable y del lado izquierdo definimos su valor (en este caso, el valor de pop dividido por un millón). Al utilizar el nombre de una variable que anteriormente ya existía no creamos una nueva, sino que reemplazamos a *pop*. Si escribimos el nombre de una variable que no existe R agrega esa variable al dataset:

```
# Calcuando el PIB
gapminder_df %>% mutate(gdp = gdpPercap * pop)
```

```
## # A tibble: 1,704 x 7
##    country     continent  year lifeExp      pop gdpPercap          gdp
##    <fct>       <fct>     <int>   <dbl>    <int>     <dbl>        <dbl>
##  1 Afghanistan Asia       1952    28.8  8425333      779.  6567086330.
##  2 Afghanistan Asia       1957    30.3  9240934      821.  7585448670.
##  3 Afghanistan Asia       1962    32.0 10267083      853.  8758855797.
##  4 Afghanistan Asia       1967    34.0 11537966      836.  9648014150.
##  5 Afghanistan Asia       1972    36.1 13079460      740.  9678553274.
##  6 Afghanistan Asia       1977    38.4 14880372      786. 11697659231.
##  7 Afghanistan Asia       1982    39.9 12881816      978. 12598563401.
##  8 Afghanistan Asia       1987    40.8 13867957      852. 11820990309.
##  9 Afghanistan Asia       1992    41.7 16317921      649. 10595901589.
## 10 Afghanistan Asia       1997    41.8 22227415      635. 14121995875.
## # ... with 1,694 more rows
```

```
# Combinando verbs
gapminder_df %>% mutate(pop = pop / 1000000) %>% filter(year == 2007) %>% arrange(gdpPercap)
```

```
## # A tibble: 142 x 6
##    country                 continent  year lifeExp   pop gdpPercap
##    <fct>                   <fct>     <int>   <dbl> <dbl>     <dbl>
##  1 Congo, Dem. Rep.        Africa     2007    46.5 64.6       278.
##  2 Liberia                 Africa     2007    45.7  3.19      415.
##  3 Burundi                 Africa     2007    49.6  8.39      430.
##  4 Zimbabwe                Africa     2007    43.5 12.3       470.
##  5 Guinea-Bissau           Africa     2007    46.4  1.47      579.
```

```
##  6 Niger                   Africa     2007    56.9 12.9       620.
##  7 Eritrea                 Africa     2007    58.0  4.91      641.
##  8 Ethiopia                Africa     2007    52.9 76.5       691.
##  9 Central African Republic Africa    2007    44.7  4.37      706.
## 10 Gambia                  Africa     2007    59.4  1.69      753.
## # ... with 132 more rows
```

Resumiendo y transformando datos en base a grupos

Muchas veces es necesario resumir diversas variables de un dataset en base a grupos. En nuestro ejemplo, preguntas que precisarían de agrupar datos serían algunas como las siguientes:

1. ¿Cuál es la expectativa de vida al nacer **promedio** por continente para cada uno de los años?
2. ¿Cuál es el país más pobre y más rico, medido por PIB per cápita, de cada continente para cada uno de los años?
3. ¿Cuál es la diferencia en la expectativa de vida al nacer para cada país con respecto a la media del continente para cada año?

Para responder estas preguntas, *tidyverse* brinda dos funciones: *group_by()*, para agrupar observaciones según categorías de una variable, y *summarise()*, para aplicar alguna transformación sobre cada conjunto de datos. Veamos cómo podríamos resolver estas tres preguntas con estas funciones y otras que ya hemos aprendido.

```
# Primera pregunta
gapminder_df %>% group_by(year, continent) %>% summarise(mean_lifeExp = mean(lifeExp))
```

```
## # A tibble: 60 x 3
## # Groups:   year [?]
##     year continent mean_lifeExp
##    <int> <fct>            <dbl>
##  1  1952 Africa            39.1
##  2  1952 Americas          53.3
##  3  1952 Asia              46.3
##  4  1952 Europe            64.4
##  5  1952 Oceania           69.3
##  6  1957 Africa            41.3
##  7  1957 Americas          56.0
##  8  1957 Asia              49.3
##  9  1957 Europe            66.7
## 10  1957 Oceania           70.3
## # ... with 50 more rows
```

```
# Segunda pregunta
gapminder_df %>% group_by(year, continent) %>%
  summarise(poor_country = min(gdpPercap),
            rich_country = max(gdpPercap),
            poor_country_nom = country[gdpPercap == poor_country],
            rich_country_nom = country[gdpPercap == rich_country])
```

```
## # A tibble: 60 x 6
## # Groups:   year [?]
##     year continent poor_country rich_country poor_country_nom
##    <int> <fct>            <dbl>        <dbl> <fct>
##  1  1952 Africa            299.        4725. Lesotho
##  2  1952 Americas         1398.       13990. Dominican Republic
##  3  1952 Asia              331       108382. Myanmar
```

```
##  4  1952 Europe            974.       14734. Bosnia and Herzegovina
##  5  1952 Oceania         10040.       10557. Australia
##  6  1957 Africa            336.        5487. Lesotho
##  7  1957 Americas         1544.       14847. Dominican Republic
##  8  1957 Asia              350       113523. Myanmar
##  9  1957 Europe           1354.       17909. Bosnia and Herzegovina
## 10  1957 Oceania         10950.       12247. Australia
## # ... with 50 more rows, and 1 more variable: rich_country_nom <fct>
```

```
# Tercera pregunta
gapminder_df %>% group_by(year, continent) %>%
  mutate(dif_lifeExp = lifeExp - mean(lifeExp))
```

```
## # A tibble: 1,704 x 7
## # Groups:   year, continent [60]
##    country     continent  year lifeExp      pop gdpPercap dif_lifeExp
##    <fct>       <fct>     <int>   <dbl>    <int>     <dbl>       <dbl>
##  1 Afghanistan Asia       1952    28.8  8425333      779.       -17.5
##  2 Afghanistan Asia       1957    30.3  9240934      821.       -19.0
##  3 Afghanistan Asia       1962    32.0 10267083      853.       -19.6
##  4 Afghanistan Asia       1967    34.0 11537966      836.       -20.6
##  5 Afghanistan Asia       1972    36.1 13079460      740.       -21.2
##  6 Afghanistan Asia       1977    38.4 14880372      786.       -21.2
##  7 Afghanistan Asia       1982    39.9 12881816      978.       -22.8
##  8 Afghanistan Asia       1987    40.8 13867957      852.       -24.0
##  9 Afghanistan Asia       1992    41.7 16317921      649.       -24.9
## 10 Afghanistan Asia       1997    41.8 22227415      635.       -26.3
## # ... with 1,694 more rows
```

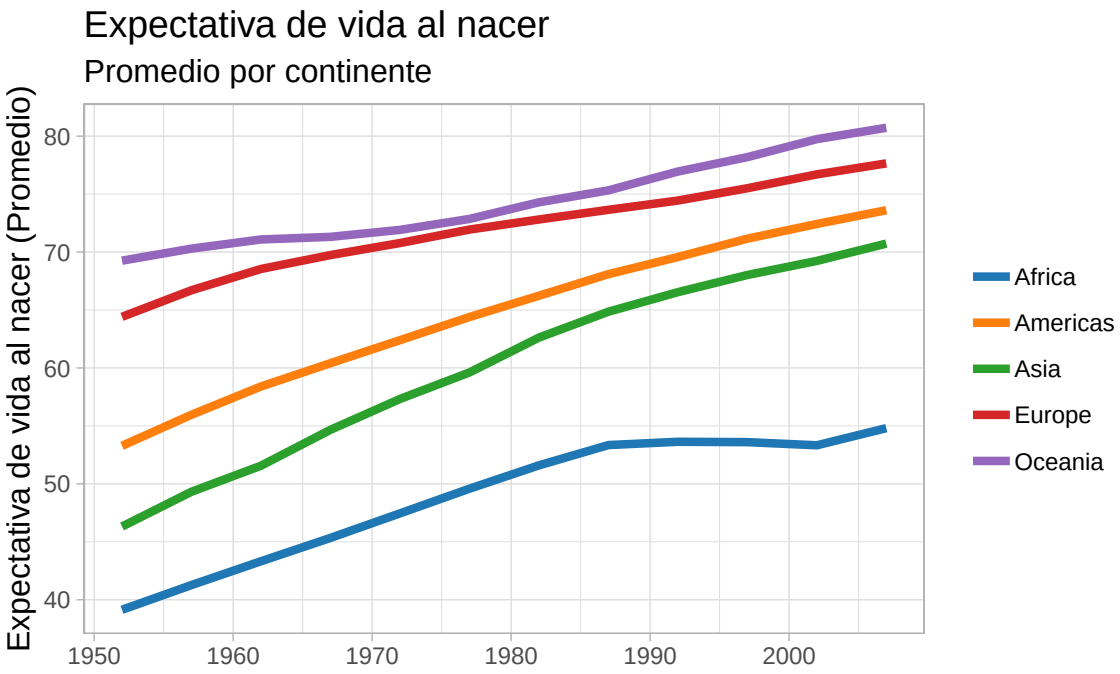
Las tres respuesta comienzan de la misma manera, es decir con la función *group_by()*. Entre los paréntesis hay que colocar los nombres de las variables por las cuáles se quiere agrupar, en nuestro caso las variables *year* y *continent*. De esta manera todo lo que siga después de la próxima pipe se aplicará sobre cada uno de los grupos generados por las combinaciones año y continente.

En el caso de la primera respuesta, usamos la función *summarise()*, que devuelve un Data Frame (en rigor, tibble) con resúmenes para cada uno de los grupos. Debido a que existen 12 "fotos" de las variables (una cada cinco años) y se definen 5 continentes (África, América, Asia, Europa y Oceanía), el data frame que devuelve tiene 60 filas (12 países multiplicado por 5 continentes da 60 grupos). Dentro de los paréntesis de *summarise()* podemos crear tantas variables como queramos. En el primer caso, creamos una variable que se llama **mean_lifeExp** que recibe el resultado de aplicar *mean()* a la variable *lifeEXp*. Esta función es aplicada a cada uno de los grupos que definimos anteriormente.

En el segundo caso hacemos algo similar a lo anterior, pero definimos cuatro variables. *poor_country* busca el valor mínimo del PIB per cápita para cada grupo a través de *min()*, *rich_country* hace lo opuesto a través de la función *max()*, *poor_country_nom* busca el nombre que corresponde al PIB per cápita más bajo y *rich_country_nom* busca el nombre que corresponde al PIB per cápita más alto. Estas últimas dos variables se generan a través de filtrar el vector *country* y buscar el valor que corresponde al PIB per cápita más bajo o alto, según corresponda, mediante el código `country[gdpPercap == poor_country]`.

Finalmente, la tercera parte del código combina las funciones *group_by()* y *mutate()*, que ya vimos anteriormente. La novedad es que ahora podemos utilizarla para crear nuevas variables en el dataset basadas en agregaciones de otras variables. En este ejemplo, creamos la variable **dif_lifeExp**, que toma la diferencia entre la expectativa de vida al nacer para cada observación y la media de cada grupo (en nuestro caso, año y continente).

El siguiente gráfico muestra una visualización basada en este dataset que acabamos de generar, usando **ggplot**



Fuente: elaboración propia con base en datos de Gapminder

Transformando la presentación de los datos: gather y spread

Los datos pueden venir presentados en dos formatos: largo o ancho. Los datasets con formato largo tienen pocas columnas y muchas filas, mientras los de formato ancho poseen muchas filas y pocas columnas. Sin embargo, en ambas representaciones los datos son exactamente los mismos.

Wide Format

	1	2	3
a	0.1	0.2	0.3
b	0.2	0.4	0.6

Long Format

a	1	0.1
b	1	0.2
a	2	0.2
b	2	0.4
a	3	0.3
b	4	0.6

En diversas situaciones es preferible tener una de las dos representaciones. Por ejemplo, para graficar con la librería `ggplot2` muchas veces es conveniente contar con representaciones en formato largo de los datos. *Tidyverse* ofrece dos métodos que sirven para este propósito: gather() y spread().

- gather(): toma un conjunto de variables (vectores/columnas) y las colapsa en una sola columna con valores que resumen los datos de ese conjunto de variables. Hace que el data frame sea más largo
- spread(): toma dos variables y las descompone entre múltiples variables (hace que el data frame sea más ancho)

```r
gapminder_sub <- gapminder_df %>% select(country,year,pop)
gapminder_sub <- gapminder_sub %>% mutate(pop = pop / 1000000) # pop en millones
head(gapminder_sub,n = 5) # Muestra las primeras cinco filas
```

```
## # A tibble: 5 x 3
##   country      year   pop
##   <fct>       <int> <dbl>
## 1 Afghanistan  1952  8.43
## 2 Afghanistan  1957  9.24
## 3 Afghanistan  1962 10.3
## 4 Afghanistan  1967 11.5
## 5 Afghanistan  1972 13.1
```

```r
wide <- gapminder_sub %>% spread(key = year,value = pop)
wide
```

```
## # A tibble: 142 x 13
##    country     `1952` `1957` `1962` `1967` `1972` `1977` `1982`  `1987`
##    <fct>        <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>   <dbl>
##  1 Afghanistan  8.43   9.24  10.3   11.5   13.1   14.9   12.9    13.9
##  2 Albania      1.28   1.48   1.73   1.98   2.26   2.51   2.78    3.08
##  3 Algeria      9.28  10.3   11.0   12.8   14.8   17.2   20.0    23.3
##  4 Angola       4.23   4.56   4.83   5.25   5.89   6.16   7.02    7.87
##  5 Argentina   17.9   19.6   21.3   22.9   24.8   27.0   29.3    31.6
##  6 Australia    8.69   9.71  10.8   11.9   13.2   14.1   15.2    16.3
##  7 Austria      6.93   6.97   7.13   7.38   7.54   7.57   7.57    7.58
##  8 Bahrain      0.120  0.139  0.172  0.202  0.231  0.297  0.378   0.455
##  9 Bangladesh  46.9   51.4   56.8   62.8   70.8   80.4   93.1   104.
## 10 Belgium      8.73   8.99   9.22   9.56   9.71   9.82   9.86    9.87
## # ... with 132 more rows, and 4 more variables: `1992` <dbl>,
## #   `1997` <dbl>, `2002` <dbl>, `2007` <dbl>
```

¿Qué es lo que hicimos? En el parámetro *key* hay que asignar la variable que queremos que queremos que deje ser un solo vector y pase a tener una columna por cada categoría que tiene. En este caso, elegimos la variable *year*, por lo que dicha variable desaparece del dataset y ahora cada categoría es una nueva columna. Luego, el parámetro *value* toma la variable que irá a cada una de la de las columnas, en este caso elegimos *pop*. Ahora vemos que la cantidad de columnas es mayor que en el dataset original y las filas son solo 142, que es la cantidad de países que tiene el dataset. Con la función gather podemos recrear el dataset original:

```r
original <- wide %>% gather(key = year,
                            value = "pop",
                            ... = 2:13,
                            convert = TRUE) %>%
  arrange(country, year)

head(original, n = 5) # Muestra las primeras cinco filas
```

```
## # A tibble: 5 x 3
##   country      year   pop
##   <fct>       <int> <dbl>
## 1 Afghanistan  1952  8.43
## 2 Afghanistan  1957  9.24
## 3 Afghanistan  1962 10.3
## 4 Afghanistan  1967 11.5
## 5 Afghanistan  1972 13.1
```

La función *gather()* requiere un poco más de atención. A diferencia de *spread()*, en *gather* tenemos que 1) definir el nombre de la nueva variable que tendrá como categoría los nombres de otras columnas (parámetro key), 2) el nombre de la variable que tendrá los valores que estaban en variables que ahora se resumirán en la nueva columna y 3) las columnas que quieren colapsarse dentro de la nueva variable en el parámetro .... Los primeros dos parámetros no tienen gran complejidad, pero la selección de las variables puede realizarse de varias maneras. En el caso anterior, lo hicimos indicando la posición de las columnas en el Data Frame. En el siguiente código exhibimos tres métodos que logran lo mismo.

```
original2 <- wide %>% gather(key = year, value = "pop", ... = -country, convert = TRUE) %>%
  arrange(country, year)  # Todas las columnas menos country
original3 <- wide %>% gather(key = year, value = "pop", ... = -1, convert = TRUE) %>%
  arrange(country, year) # Todas las columnas menos la primera
original4 <- wide %>% gather(key = year, value = "pop",
                             ... = `1952`,`1957`,`1962`,`1967`,`1972`,`1977`, `1982`,
                             `1987`,`1992`,`1997`,`2002`,`2007`,
                            convert = TRUE) %>%
  arrange(country, year) # Todas las columnas con esos nombres
# Esta línea nos devuelve TRUE si los 5 objetos sin iguales entre sí o FALSE si hay
# al menos uno que no lo es
all(sapply(list(original, original2, original3, original4),
           function(x) identical(x, gapminder_sub)))
```

```
## [1] TRUE
```

Finalmente, hay un cuarto parámetro cuyo uso no discutimos: *convert*. Por default tiene valor FALSE y asume que los nombres de las columnas deben colapsarse como textos. En nuestro caso, los nombres de las variables hacían referencia a años, que deben ser convertidos a números en la nueva columna. Eso es exactamente lo que hace `convert = TRUE`

La *mise en place*: preparando el dataset de inmuebles

A esta altura vimos una gran cantidad de comandos para transformar datos con un interesante dataset como es **gapminder**. Sin embargo, probablemente hayan sido demasiados para procesarlos de una sola vez. Lo que vamos a hacer ahora es aplicarlo en nuestro objetivo inicial: transformar los datasetas tal como se descargan desde **Properati** hasta el dataset que trabajamos la clase pasada. Vamos a usar varios de los comandos que vimos anteriormente.

Vamos a trabajar con una versión levemente modificada de los datos descargados de Properati porque vamos a procesar una muestra estratificada por barrio y año. Esto quiere decir que tomamos aleatoriamente 30 observaciones (siempre que las hayan) de cada combinación barrio y año. Esto lo hacemos para reducir la cantidad de filas, que eran más de 400mil en el dataset original y logramos reducirlas sensiblemente, sin perder generalidad en nuestra explicación.

```
barriosOriginal <- read.table(file = "https://datalab-utdt.github.io/datasets/barriosSample.csv",
                              sep = ";",
                              header = TRUE,
                              stringsAsFactors = FALSE)
dim(barriosOriginal) # Número de filas y de columnas
```

```
## [1] 4780   31
```

```
colnames(barriosOriginal) # Nombres de las columnas
```

```
##  [1] "BARRIOS"                   "COMUNA"
##  [3] "NUM_DE_BAR"                "created_on"
##  [5] "operation"                 "property_type"
```

```
##  [7] "place_name"                 "place_with_parent_names"
##  [9] "geonames_id"                "lat.lon"
## [11] "price"                      "currency"
## [13] "price_aprox_local_currency" "price_aprox_usd"
## [15] "surface_in_m2"              "price_usd_per_m2"
## [17] "floor"                      "rooms"
## [19] "expenses"                   "properati_url"
## [21] "image_thumbnail"            "description"
## [23] "title"                      "extra"
## [25] "surface_total_in_m2"        "surface_covered_in_m2"
## [27] "price_per_m2"               "id"
## [29] "country_name"               "state_name"
## [31] "year"
```

Recuerden que el dataset con el que trabajamos la clase anterior tenia tan solo 7 variables, incluyendo una que contenía datos sobre los polígonos espaciales (variable que vamos a introducirla en la próxima clase y la descartamos en esta reconstrucción). Las seis variables restantes son 1) los nombres de los barrios y 2) el valor de los precios en dólares para cada año (2013-2017).

Para empezar, vamos a quedarnos solo con las variables que son relevantes: barrios, precios en dolares y años

```
barriosOriginal <- barriosOriginal %>% select(BARRIOS, price_usd_per_m2, year)
str(barriosOriginal)
```

```
## 'data.frame':    4780 obs. of  3 variables:
##  $ BARRIOS         : chr  "AGRONOMIA" "AGRONOMIA" "AGRONOMIA" "AGRONOMIA" ...
##  $ price_usd_per_m2: num  1304 1642 1333 2800 1467 ...
##  $ year            : int  2013 2013 2013 2013 2013 2013 2013 2013 2013 2013 ...
```

Ahora, necesitamos calcular el precio PROMEDIO por barrio y año. Por suerte, ya conocemos las funciones **group_by** y **summarise**. Es todo lo que necesitamos para esta transformación:

```
barriosOriginal <- barriosOriginal %>%
                   group_by(BARRIOS, year) %>%
                   summarise(precioPromedio = mean(price_usd_per_m2))
str(barriosOriginal)
```

```
## Classes 'grouped_df', 'tbl_df', 'tbl' and 'data.frame':  240 obs. of  3 variables:
##  $ BARRIOS       : chr  "AGRONOMIA" "AGRONOMIA" "AGRONOMIA" "AGRONOMIA" ...
##  $ year          : int  2013 2014 2015 2016 2017 2013 2014 2015 2016 2017 ...
##  $ precioPromedio: num  1682 1254 1588 2051 1781 ...
##  - attr(*, "vars")= chr "BARRIOS"
##  - attr(*, "drop")= logi TRUE
```

Ya estamos bastante cerca de nuestro objetivo: contamos con un **Data Frame** donde se registra el precio promedio en USD del metro cuadrado de los inmuebles por barrio para el período 2013-2017. La diferencia con el dataset de la clase pasada es que este se presenta en formato largo y el anterior en formato ancho. La función **spread**, que ya vimos anteriormente, va a hacer lo que necesitamos

```
barriosOriginal <- barriosOriginal %>% spread(key = year, value = precioPromedio)
head(barriosOriginal, n = 3)
```

```
## # A tibble: 3 x 6
## # Groups:   BARRIOS [3]
##   BARRIOS   `2013` `2014` `2015` `2016` `2017`
##   <chr>      <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
## 1 AGRONOMIA  1682.  1254.  1588.  2051.  1781.
## 2 ALMAGRO    2024.  1922.  1989.  1891.  2270.
```

```
## 3 BALVANERA  1857.  1791.  1908.  1642.  1668.
```

Ya casi estamos ! Lo único que nos falta es cambiar el nombre de las columnas. Para eso vamos a utilizar la función `paste()`:

```
colnames(barriosOriginal)[2:6] <- paste('USDm2_',colnames(barriosOriginal)[2:6], sep="")
head(barriosOriginal, n =3)
```

```
## # A tibble: 3 x 6
## # Groups:   BARRIOS [3]
##   BARRIOS   USDm2_2013 USDm2_2014 USDm2_2015 USDm2_2016 USDm2_2017
##   <chr>          <dbl>      <dbl>      <dbl>      <dbl>      <dbl>
## 1 AGRONOMIA      1682.      1254.      1588.      2051.      1781.
## 2 ALMAGRO        2024.      1922.      1989.      1891.      2270.
## 3 BALVANERA      1857.      1791.      1908.      1642.      1668.
```

Lo que hace esta función es tomar vectores y concatenarlos entre sí. En este caso le pasamos dos vectores: uno de tipo *character* con un solo elemento ("USDm2_") y otro que contiene la posición 2 a 6 del vector *character* `colnames(barriosOriginal)`, es decir los nombres de las variables 2 a 6 del dataset `barriosOriginal`. En estos simples pasos ya construimos el dataset con el que trabajamos la clase anterior desde el formato que tenían cuando se descargaron desde el portal de datos.

Ejercicios

Para repasar los conceptos, prácticar y comprender lo poderosa que es la programación para repetir procesos rutinarios, vamos a llegar al dataset de Comunas que presentamos en la clase anterior. Recordar que el archivo que leyeron anteriormente desde internet (**barriosOriginal**) tiene una columna que indica la **comuna** a la cual pertenece la observación.