Menos for, más purrr:
Programación funcional con R

Nacho Evangelista
evangelistaignacio@gmail.com

18 de febrero de 2021

Contenido

Motivación

- Buscamos evitar duplicación de código. La duplicación hace que los errores y *bugs* sean más frecuentes. También se hace más difícil modificar el código.
- Un estilo funcional implica descomponer un problema grande en partes y resolver cada una de estas partes con una función o combinación de funciones.
- La idea es arrancar con porciones de código pequeños y fáciles de entender (funciones). Combinar estos bloques en estructuras más complejas.
- El estilo funcional permite obtener soluciones eficientes y elegantes a problemas modernos.

- Nuestro objetivo es reemplazar los *for loops* o estructuras de repetición.
- En lugar de eso, trataremos de trabajar con operaciones vectorizadas
- Por ejemplo: en una estructura funcional, no es necesario crear una lista vacía para ir guardando resultados ni ir llevando control de un índice. El código es más conciso.
- En R base existen las funciones de la familia `apply`. En el paquete `purrr`, estas funciones se reemplazan por la familia `map`, más fáciles de usar. Se llaman funcionales: reciben una función como argumento.

Listas

El bloque fundamental de `purrr` son las listas:

- Un vector es un objeto que guarda elementos individuales del mismo tipo
- Un dataframe es una estructura que guarda varios vectores de la misma longitud pero de distinto tipo.
- Una lista es una estructura que permite guardar objetos de distinto tipo y longitud.

¿Cómo se definen?

```r
animalitos ← list(
  animales = c("perro", "gato", "elefante", "vaca"),
  acciones = c("ladrar", "maullar", "barritar", "mugir"),
  nombres = list("Dalmata" = "Pongo", "Marley"),
  5)
```

```r
str(animalitos)
```

```
List of 4
 $ animales: chr [1:4] "perro" "gato" "elefante" "vaca"
 $ acciones: chr [1:4] "ladrar" "maullar" "barritar" "mugir"
 $ nombres :List of 2
  ..$ Dalmata: chr "Pongo"
  ..$        : chr "Marley"
 $          : num 5
```

```r
names(animalitos)
```

```
[1] "animales" "acciones" "nombres"  ""
```

¿Cómo acceder a sus elementos?

```
animalitos[["animales"]]
```

```
[1] "perro"    "gato"     "elefante" "vaca"
```

```
animalitos$acciones
```

```
[1] "ladrar"   "maullar"  "barritar" "mugir"
```

```
animalitos$"nombres"
```

```
$Dalmata
[1] "Pongo"

[[2]]
[1] "Marley"
```

```
animalitos[[1]]
```

```
[1] "perro"    "gato"     "elefante" "vaca"
```

```
animalitos[[4]]
```

```
[1] 5
```

El paquete purrr

- Muchas operaciones de R funcionan en forma vectorizada; aplicadas a vectores, algunas funciones se aplican elemento a elemento (una suerte de iteración)

```
v ← c(2,5,7)
exp(v)
```

```
[1]    7.389056  148.413159 1096.633158
```

- Muchas funciones no tienen esa capacidad

```
meses ← c("Ene","Feb","Mar","Abr","May","Jun",
          "Jul","Ago","Sep","Oct","Nov","Dic")
which(meses=="Sep")
which(meses==c("Sep","Ene"))
```

```
[1] 9
[1] 9
```

Idea clave

El paquete `purrr` sirve para vectorizar operaciones y por lo tanto, para iterar

La función `map`

- La función (funcional) más básica de `purrr` es `map`:
  1. Toma un vector y una función,
  2. aplica la función a cada elemento del vector,
  3. devuelve los resultados en una lista.

Función `map`

`map(.x, .f, ...)`

- `.x` es la lista (o el vector) sobre la que aplicaremos la función,
- `.f` es la función en cuestión,
- `...` son argumentos adicionales de `.f`

- `map(1:3,f)` es equivalente a `list(f(1), f(2), f(3))`

Ejemplos

Apliquemos algunas funciones a nuestra lista de animalitos

```
map(animalitos,typeof)
```

```
$animales
[1] "character"

$acciones
[1] "character"

$nombres
[1] "list"

[[4]]
[1] "double"
```

```
map(animalitos,length)
```

```
$animales
[1] 4

$acciones
[1] 4

$nombres
[1] 2

[[4]]
[1] 1
```

También podemos usar purrr con vectores

```
v ← c(1,3)
map(v,rnorm)
```

```
[[1]]
[1] -1.206372

[[2]]
[1]  0.9604448 -0.6479977  0.5383806
```

```
# agregamos parámetros adicionales de la función rnorm
v ← c(1,5)
map(v,rnorm,mean=4,sd=2)
```

```
[[1]]
[1] 4.078548

[[2]]
[1] 3.1004876 0.7231275 5.9310111 2.7001499 3.2164765
```

Variando el tipo de salida

Si la función que aplicamos devuelve un único elemento, podemos usar las variantes `map_lgl`, `map_int`, `map_dbl` o `map_chr` y obtener un vector como resultado

```
map_chr(animalitos,typeof)
```

```
   animales    acciones     nombres
"character" "character"      "list"    "double"
```

```
map_int(animalitos,length)
```

```
animales acciones  nombres
       4         4         2         1
```

Funciones propias

Además de usar funciones predefinidas, podemos definir nuestras propias funciones

```
sumar_diez ← function(x) return(x+10)
map_dbl(c(-10,4,7),sumar_diez)
```

```
[1]  0 14 17
```

```
map_dbl(c(-10,4,7),function(x) return(x+10))
```

```
[1]  0 14 17
```

```
# función anónima (el argumento es siempre .x)
map_dbl(c(-10,4,7), ~.x+10)
```

```
[1]  0 14 17
```

La definición de una función anónima permite ser más explícito en el pasaje de parámetros

```
v ← c(1,5)
map(v,~rnorm(.x,mean=4,sd=2)) # el argumento es siempre .x!
```

```
[[1]]
[1] 0.6427472


[[2]]
[1] -1.1118872  3.8886009  1.7938746  5.0563910 -0.6634644
```

Comparar con

```
v ← c(1,5)
map(v,rnorm,mean=4,sd=2)
```

También podemos hacer

```
v ← c(1,8)
map(v,~rnorm(n=2,mean=.x,sd=1)) # el argumento es siempre .x!
```

```
[[1]]
[1]  0.7028858 -2.0754148
```

Dos argumentos: map2

Queremos extraer las primeras n letras de algunas palabras, donde n es una cantidad variable

```
palabras ← c("recorcholis","nacimiento","rosedal","artista","ion")
cantidad_letras ← c(2,1,3,2,2)
# los argumentos son .x y .y
map2_chr(palabras,cantidad_letras,~substr(.x,1,.y))
```

```
[1] "re"  "n"   "ros" "ar"  "io"
```

Queremos generar dos secuencias de fechas

```
inicio ← as.Date(c("2018-01-03","2019-03-06"))
fin ← as.Date(c("2018-01-06","2019-03-08"))
map2(inicio,fin,~seq.Date(.x,.y,by="1 day"))
```

```
[[1]]
[1] "2018-01-03" "2018-01-04" "2018-01-05" "2018-01-06"

[[2]]
[1] "2019-03-06" "2019-03-07" "2019-03-08"
```

Pero yo nunca usé listas...

```
pelis ← tibble(
cancion = c("Strange Things", "Life is a Highway", "I'm a Believer"),
autor = c("Randy Newman", "Rascal Flatts", "Smash Mouth"),
pelicula = c("Toy Story", "Cars", NA)
)
```

```
typeof(pelis)
```

```
[1] "list"
```

```
as.list(pelis)
```

```
$cancion
[1] "Strange Things"    "Life is a Highway" "I'm a Believer"

$autor
[1] "Randy Newman"  "Rascal Flatts" "Smash Mouth"

$pelicula
[1] "Toy Story" "Cars"      NA
```

```
map_chr(pelis,typeof)
```

```
    cancion       autor    pelicula
"character" "character" "character"
```

```
map_int(pelis,~sum(is.na(.x)))
```

```
 cancion    autor pelicula
       0        0        1
```

Hasta acá...

Resumen

- Las listas son estructuras de datos muy versátiles
- Un dataframe/tibble es una lista
- `purrr` sirve para iterar
- `map` es el caballito de batalla del paquete `purrr`
- Se puede especificar el tipo de salida con las variantes `map_`
- Para iterar a lo largo de dos listas se usa `map2`
- Hay varias formas de pasar una función como argumento de `map`

I. En la definición del `tibble`

```
T1 ← tibble(
  v1 = 1:3,
  v2 = c("a","b","c"),
  v3 = list(c("A","B","C")),
  v4 = 10
)
```

```
# A tibble: 3 x 4
     v1 v2    v3           v4
  <int> <chr> <list>    <dbl>
1     1 a     <chr [3]>    10
2     2 b     <chr [3]>    10
3     3 c     <chr [3]>    10
```

```r
T2 ← tibble(
  v1 = 1:3,
  v2 = c("a","b","c"),
  v3 = list(c("A","B","C"),"B")
)
```

```
Error : Tibble columns must have compatible sizes.
* Size 3: Existing data.
* Size 2: Column ‘v3‘.
i Only values of size one are recycled.
```

```r
T3 ← tibble(
  v1 = 1:3,
  v2 = c("a","b","c"),
  v3 = list(c("A","B","C"),"perro",5)
)
```

```
# A tibble: 3 x 3
     v1 v2    v3
  <int> <chr> <list>
1     1 a     <chr [3]>
2     2 b     <chr [1]>
3     3 c     <dbl [1]>
```

```r
T4 ← tibble(
  v1 = 1:3,
  v2 = c("a","b","c"),
  v3 = list(rnorm(1),rnorm(10),rnorm(100))
)
```

```
# A tibble: 3 x 3
     v1 v2    v3
  <int> <chr> <list>
1     1 a     <dbl [1]>
2     2 b     <dbl [10]>
3     3 c     <dbl [100]>
```

```
animalitos ← list(animales = c("perro","gato","elefante","vaca"),
                  acciones = c("ladrar","maullar","barritar","mugir"),
                  nombres = list(c("Pongo","Marley","Golfo"),
                                 c("Pelusa","Tom"),
                                 c("Tantor","Dumbo"),
                                 c("Oscar")))

as_tibble(animalitos)
```

```
# A tibble: 4 x 3
  animales acciones nombres
  <chr>    <chr>    <list>
1 perro    ladrar   <chr [3]>
2 gato     maullar  <chr [2]>
3 elefante barritar <chr [2]>
4 vaca     mugir    <chr [1]>
```

II. Usando nest

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am
Mazda RX4	21.0	6	160	110	3.90	2.620	16.46	0	1
Mazda RX4 Wag	21.0	6	160	110	3.90	2.875	17.02	0	1
Datsun 710	22.8	4	108	93	3.85	2.320	18.61	1	1
Hornet 4 Drive	21.4	6	258	110	3.08	3.215	19.44	1	0
Hornet Sportabout	18.7	8	360	175	3.15	3.440	17.02	0	0
Valiant	18.1	6	225	105	2.76	3.460	20.22	1	0

	gear	carb
Mazda RX4	4	4
Mazda RX4 Wag	4	4
Datsun 710	4	1
Hornet 4 Drive	3	1
Hornet Sportabout	3	2
Valiant	3	1

```
mtcars %>%
  group_by(gear,carb) %>%
  nest()
```

```
# A tibble: 11 x 3
# Groups:   gear, carb [11]
    gear  carb data
   <dbl> <dbl> <list>
 1     4     4 <tibble [4 x 9]>
 2     4     1 <tibble [4 x 9]>
 3     3     1 <tibble [3 x 9]>
 4     3     2 <tibble [4 x 9]>
 5     3     4 <tibble [5 x 9]>
 6     4     2 <tibble [4 x 9]>
 7     3     3 <tibble [3 x 9]>
 8     5     2 <tibble [2 x 9]>
 9     5     4 <tibble [1 x 9]>
10     5     6 <tibble [1 x 9]>
# ... with 1 more row
```

```
mtcars %>%
  nest(datos = !c(gear,carb))
```

```
# A tibble: 11 x 3
    gear  carb datos
   <dbl> <dbl> <list>
 1     4     4 <tibble [4 x 9]>
 2     4     1 <tibble [4 x 9]>
 3     3     1 <tibble [3 x 9]>
 4     3     2 <tibble [4 x 9]>
 5     3     4 <tibble [5 x 9]>
 6     4     2 <tibble [4 x 9]>
 7     3     3 <tibble [3 x 9]>
 8     5     2 <tibble [2 x 9]>
 9     5     4 <tibble [1 x 9]>
10     5     6 <tibble [1 x 9]>
# ... with 1 more row
```

III. Como resultado de una operación

Queremos unir estas dos tablas reemplazando los códigos del casting por los personajes.

```
peleas ← tibble::tribble(
 ~pelea, ~horario,                        ~casting,
      1,  "20:30",         "gsf901,fez195,yfm179",
      2,  "20:50",                "thf028,yfm179",
      3,  "19:40", "jfa348,fez195,gky651,wpx281",
      4,  "21:00",                "thf028,fez195")
```

```
luchadores ← tibble::tribble(
   ~codigo,            ~nombre,
  "gsf901",  "Vicente Viloni",
  "thf028",     "Hip Hop Man",
  "wpx281",         "La Masa",
  "fez195", "Fulgencio Mejía",
  "jfa348",        "Mc Floyd",
  "phb625",     "Mario Morán",
  "gky651",      "Rulo Verde",
  "yfm179",    "Steve Murphy")
```

```
peleas %>%
  mutate(casting_split = strsplit(casting, split = ",")) %>%
  select(-horario,-casting) %>%
  unnest(casting_split) %>%
  left_join(luchadores, by = c("casting_split" = "codigo"))
```

```
# A tibble: 11 x 3
   pelea casting_split nombre
   <dbl> <chr>         <chr>
 1     1 gsf901        Vicente Viloni
 2     1 fez195        Fulgencio Mejía
 3     1 yfm179        Steve Murphy
 4     2 thf028        Hip Hop Man
 5     2 yfm179        Steve Murphy
 6     3 jfa348        Mc Floyd
 7     3 fez195        Fulgencio Mejía
 8     3 gky651        Rulo Verde
 9     3 wpx281        La Masa
10     4 thf028        Hip Hop Man
# ... with 1 more row
```

¿Por qué necesito purrr?

¿Y ahora qué?

Con las columnas lista se abre ante nosotros un universo de posibilidades y `purrr` es la herramienta ideal para explorarlo

1. Identificar el mes

Queremos obtener el número de mes a partir de la abreviatura

```
datos ← tibble::tribble(
 ~id,~dia,  ~mes, ~año,
   1,  15, "Sep", 2019,
   2,   6, "oct", 2021,
   3,   3, "Ene", 2020,
   4,  31, "dic", 2019)
```

Idea

1. Armar un vector con las abreviaturas de los meses
2. Usar la función `which` junto con `map`

```
meses ← c("Ene","Feb","Mar","Abr","May","Jun",
          "Jul","Ago","Sep","Oct","Nov","Dic")
```

```
which(meses=="Sep")
```

```
[1] 9
```

```
which(meses==c("Sep","Ene"))
```

```
[1] 9
```

```
datos %>%
  mutate(mes_n = map(mes,~which(meses==.x)))
```

```
# A tibble: 4 x 5
     id   dia mes      año mes_n
  <dbl> <dbl> <chr> <dbl> <list>
1     1    15 Sep    2019 <int [1]>
2     2     6 oct    2021 <int [0]>
3     3     3 Ene    2020 <int [1]>
4     4    31 dic    2019 <int [0]>
```

```
datos %>%
  mutate(mes_n = map(mes,~which(toupper(meses)==toupper(.x))))
```

```
# A tibble: 4 x 5
     id   dia mes      año mes_n
  <dbl> <dbl> <chr> <dbl> <list>
1     1    15 Sep    2019 <int [1]>
2     2     6 oct    2021 <int [1]>
3     3     3 Ene    2020 <int [1]>
4     4    31 dic    2019 <int [1]>
```

```
datos %>%
  mutate(mes_n = map_int(mes,~which(toupper(meses)==toupper(.x))))
```

```
# A tibble: 4 x 5
     id   dia mes      año mes_n
  <dbl> <dbl> <chr> <dbl> <int>
1     1    15 Sep    2019     9
2     2     6 oct    2021    10
3     3     3 Ene    2020     1
4     4    31 dic    2019    12
```

2. Secuencia de fechas

Contamos con los movimientos de dos empresas. Interesa tener la serie temporal de eventos para cada empresa y producto.

```
datos ← tibble::tribble(
  ~empresa, ~producto,       ~fecha, ~evento,
       "A",       "A1", "02/06/2018",     112,
       "A",       "A1", "06/06/2018",     141,
       "A",       "A1", "13/07/2018",     119,
       "A",       "A2", "01/05/2018",      53,
       "A",       "A2", "04/05/2018",      67,
       "B",       "B1", "01/07/2018",     127,
       "B",       "B1", "05/07/2018",     301,
       "B",       "B1", "10/07/2018",      98,
       "B",       "B1", "11/07/2018",     167)
datos$fecha ← as.Date(datos$fecha,format = "%d/%m/%Y")
```

Idea:

1. Determinar la primera y última fecha de cada grupo
2. Generar una secuencia de fechas (`seq.Date`) para cada grupo y construir una tabla con todas las fechas
3. Unir esta tabla con la original

```
fechas_todas ←
  datos %>%
  group_by(empresa, producto) %>%
  summarise(fecha_inicial = min(fecha),
            fecha_final = max(fecha)) %>%
  mutate(fechas = map2(fecha_inicial,
                       fecha_final,
                       ~seq.Date(.x,.y,by="1 day"))) %>%
  select(-fecha_inicial,-fecha_final)
```

```
`summarise()` regrouping output by 'empresa' (override with `
```

```
datos %>%
  group_by(empresa, producto) %>%
  summarise(fecha_inicial = min(fecha),
            fecha_final = max(fecha))
```

```
`summarise()` regrouping output by 'empresa' (override with `
```

```
# A tibble: 3 x 4
# Groups:   empresa [2]
  empresa producto fecha_inicial fecha_final
  <chr>   <chr>    <date>        <date>
1 A       A1       2018-06-02    2018-07-13
2 A       A2       2018-05-01    2018-05-04
3 B       B1       2018-07-01    2018-07-11
```

```
datos %>%
group_by(empresa, producto) %>%
summarise(fecha_inicial = min(fecha),
          fecha_final = max(fecha)) %>%
mutate(fechas = map2(fecha_inicial,
                     fecha_final,
                     ~seq.Date(.x,.y,by="1 day")))
```

```
`summarise()` regrouping output by 'empresa' (override with `
```

```
# A tibble: 3 x 5
# Groups:   empresa [2]
  empresa producto fecha_inicial fecha_final fechas
  <chr>   <chr>    <date>        <date>      <list>
1 A       A1       2018-06-02    2018-07-13  <date [42]>
2 A       A2       2018-05-01    2018-05-04  <date [4]>
3 B       B1       2018-07-01    2018-07-11  <date [11]>
```

```
fechas_todas %>%
  unnest(fechas) %>%
  left_join(datos, by = c("empresa","producto","fechas" = "fecha"))
```

```
# A tibble: 57 x 4
# Groups:   empresa [2]
   empresa producto fechas     evento
   <chr>   <chr>    <date>      <dbl>
 1 A       A1       2018-06-02    112
 2 A       A1       2018-06-03     NA
 3 A       A1       2018-06-04     NA
 4 A       A1       2018-06-05     NA
 5 A       A1       2018-06-06    141
 6 A       A1       2018-06-07     NA
 7 A       A1       2018-06-08     NA
 8 A       A1       2018-06-09     NA
 9 A       A1       2018-06-10     NA
10 A       A1       2018-06-11     NA
# ... with 47 more rows
```

3. Abrir varios archivos a la vez

En el directorio de trabajo hay varios archivos que debemos abrir y leer.

```
list.files(pattern="archivo")
```

```
[1] "archivo_1.csv" "archivo_2.csv" "archivo_3.csv"
```

Idea:

1. Listar los archivos y construir un `tibble`
2. Leer cada archivo con `read.csv`
3. *Desanidar*

```
list.files(pattern="archivo") %>%
  tibble(archivos = .) %>%
  mutate(contenido = map(archivos,read.csv)) %>%
  unnest(contenido)
```

```
list.files(pattern="archivo") %>%
  tibble(archivos = .) %>%
  mutate(contenido = map(archivos,read.csv))
```

```
# A tibble: 3 x 2
  archivos      contenido
  <chr>         <list>
1 archivo_1.csv <df[,5] [6 x 5]>
2 archivo_2.csv <df[,5] [6 x 5]>
3 archivo_3.csv <df[,5] [6 x 5]>
```

```r
list.files(pattern="archivo") %>%
  tibble(archivos = .) %>%
  mutate(contenido = map(archivos,read.csv)) %>%
  unnest(contenido)
```

```
# A tibble: 18 x 6
   archivos      name    cat   fecha         v1    v2
   <chr>         <fct>   <fct> <fct>      <dbl> <int>
 1 archivo_1.csv ocgux74 B     2021-05-24   8.3    19
 2 archivo_1.csv lecjd73 A     2021-03-16   2.1    60
 3 archivo_1.csv hzprt72 D     2021-04-30   5.8    24
 4 archivo_1.csv epwoz07 A     2021-02-15   8.4    65
 5 archivo_1.csv zfdas14 D     2020-12-21   5      46
 6 archivo_1.csv ywqiu85 D     2021-06-06   0.4    99
 7 archivo_2.csv shjqu73 D     2021-01-26   5.6    94
 8 archivo_2.csv oncxv34 A     2021-03-20   2.1    23
 9 archivo_2.csv yzqml54 D     2021-03-05   2.3    15
10 archivo_2.csv ohsaq71 C     2021-07-13   4.4    31
# ... with 8 more rows
```

4. Múltiples salidas

Queremos analizar frases de canciones y determinar: a) cantidad de palabras, b) cantidad de preposiciones

```
# A tibble: 7 x 3
  banda        cancion           frase
  <chr>        <chr>             <chr>
1 Los Wachit~ Este es el pasi~ El que no hace palmas es un ~
2 La Base     Sabor sabrosón   Según la moraleja, el que no~
3 Damas Grat~ Me va a extrañar ATR perro cumbia cajeteala p~
4 Altos Cumb~ No voy a llorar  Andy, fijate que volvieron, ~
5 Los Pibes ~ Llegamos los Pi~ Llegamos los pibes chorros q~
6 La Liga     Se re pudrió     El que no hace palmas tiene ~
7 Los Palmer~ La cola          A la una, a la dos, a la one~
```

Idea:

1. Definir una función que devuelva ambas cantidades
2. Aplicarla a cada frase

```r
analizar_frase ← function(cancion){
  preposiciones ← c("a", "ante", "bajo", "cabe", "con",
                    "contra", "de", "desde", "durante",
                    "en", "entre", "hacia", "hasta", "mediante",
                    "para", "por", "según", "sin", "so", "sobre",
                    "tras", "versus", "vía")

  palabras ← strsplit(cancion," ") %>% unlist

  cant_palabras ← length(palabras)

  cant_preposiciones ← sum(palabras %in% preposiciones)

  return(list(cant_palabras = cant_palabras,
              cant_preposiciones = cant_preposiciones))
  }
```

```
datos %>%
  mutate(resultado = map(frase,analizar_frase)) %>%
  unnest_wider(resultado)
```

```
datos %>%
  mutate(resultado = map(frase,analizar_frase))
```

```
# A tibble: 7 x 4
  banda     cancion         frase                  resultado
  <chr>     <chr>           <chr>                  <list>
1 Los Wach~ Este es el pa~ El que no hace palma~ <named lis~
2 La Base   Sabor sabrosón Según la moraleja, e~ <named lis~
3 Damas Gr~ Me va a extra~ ATR perro cumbia caj~ <named lis~
4 Altos Cu~ No voy a llor~ Andy, fijate que vol~ <named lis~
5 Los Pibe~ Llegamos los ~ Llegamos los pibes c~ <named lis~
6 La Liga   Se re pudrió   El que no hace palma~ <named lis~
7 Los Palm~ La cola         A la una, a la dos, ~ <named lis~
```

```
datos %>%
  mutate(resultado = map(frase,analizar_frase)) %>%
  unnest_wider(resultado)
```

```
# A tibble: 7 x 5
  banda   cancion   frase     cant_palabras cant_preposicio~
  <chr>   <chr>     <chr>             <int>            <int>
1 Los Wa~ Este es ~ El que n~             8                0
2 La Base Sabor sa~ Según la~            12                0
3 Damas ~ Me va a ~ ATR perr~             6                0
4 Altos ~ No voy a~ Andy, fi~             8                0
5 Los Pi~ Llegamos~ Llegamos~            10                1
6 La Liga Se re pu~ El que n~             9                1
7 Los Pa~ La cola   A la una~            15                2
```

5. Múltiples plots

Queremos construir un conjunto de plots mostrando los ajustes de polinomios de distinto orden a los puntos del dataset.

```
datos ← tibble::tribble(
 ~‘x‘,  ~‘y‘,
  211,   184,
  230,   147,
  587,   413,
  414,   252,
  419,   252,
  157,   272,
  327,   158,
  222,   158,
  451,   249,
  296,   127)
```

Idea:

1. Combinamos los datos con cada uno de los posibles órdenes del polinomio
2. `group_by` + `nest`
3. Aplicar una función que cree el gráfico utilizando `map`

```
plots ← crossing(poly = 1:6, datos) %>%
  nest(datos = !poly) %>%
  ungroup() %>%
  mutate(plot = map2(datos, poly,
                     function(data, poly) {
                       ggplot(data, aes(x = x, y = y)) +
                         geom_point() +
                         stat_smooth(
                           method = "lm", se = FALSE,
                           formula = y ~ poly(x, poly, raw = TRUE),
                           colour = "maroon1") +
                         theme_minimal()
                     }))
```

```
crossing(poly = 1:6, datos)
```

```
# A tibble: 60 x 3
    poly     x     y
   <int> <dbl> <dbl>
 1     1   157   272
 2     1   211   184
 3     1   222   158
 4     1   230   147
 5     1   296   127
 6     1   327   158
 7     1   414   252
 8     1   419   252
 9     1   451   249
10     1   587   413
# ... with 50 more rows
```

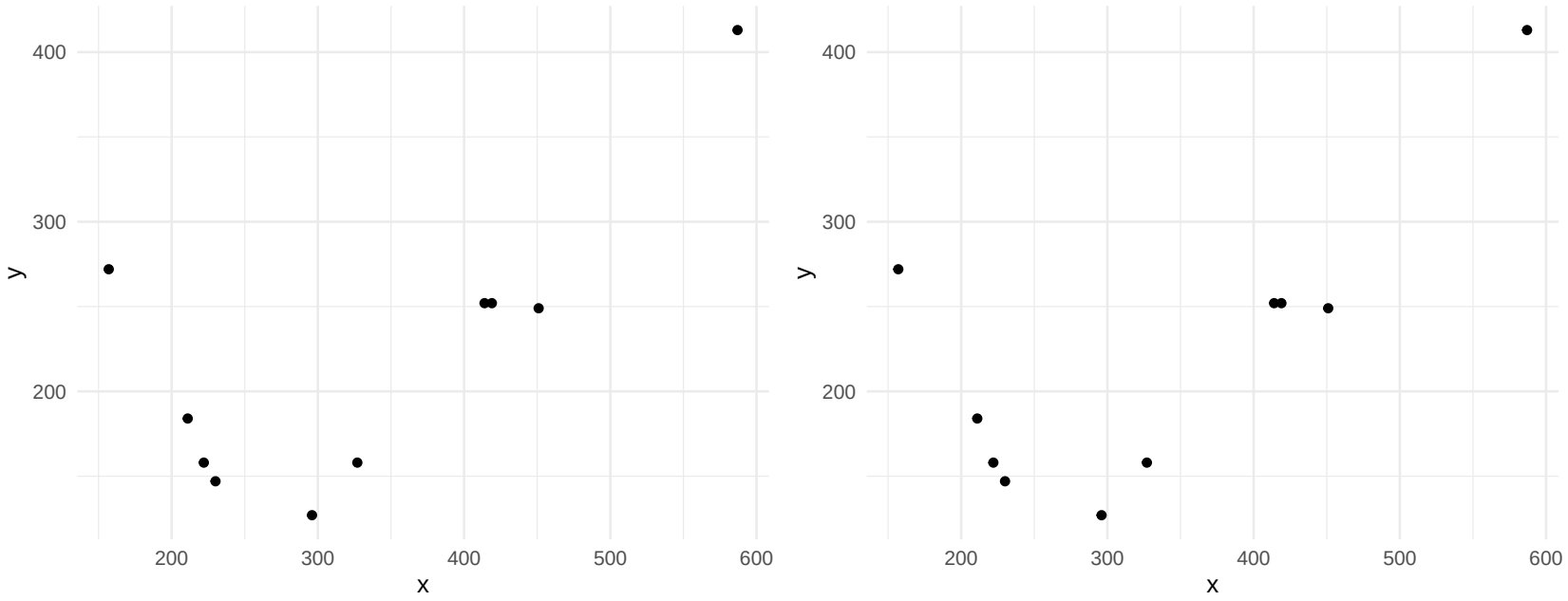
```
crossing(poly = 1:6, datos) %>%
  nest(datos = !poly)
```

```
# A tibble: 6 x 2
   poly datos
  <int> <list>
1     1 <tibble [10 x 2]>
2     2 <tibble [10 x 2]>
3     3 <tibble [10 x 2]>
4     4 <tibble [10 x 2]>
5     5 <tibble [10 x 2]>
6     6 <tibble [10 x 2]>
```

```
plots ← crossing(poly = 1:6, datos) %>%
  nest(datos = !poly) %>%
  ungroup() %>%
  mutate(plot = map2(datos, poly,
                     function(data, poly) {
                       ggplot(data, aes(x = x, y = y)) +
                         geom_point() +
                         stat_smooth(
                           method = "lm", se = FALSE,
                           formula = y ~ poly(x, poly, raw = TRUE),
                           colour = "maroon1") +
                         theme_minimal()
                     }))
```

```
# A tibble: 6 x 3
   poly datos             plot
  <int> <list>            <list>
1     1 <tibble [10 x 2]> <gg>
2     2 <tibble [10 x 2]> <gg>
3     3 <tibble [10 x 2]> <gg>
4     4 <tibble [10 x 2]> <gg>
5     5 <tibble [10 x 2]> <gg>
6     6 <tibble [10 x 2]> <gg>
```

```
library(patchwork)
plots$plot[[1]] + plots$plot[[6]]
```



6. K-fold cross validation

```
K ← 3
data ← mtcars %>%
  mutate(fold = rep(1:K,length.out=nrow(.))) %>%
  arrange(fold) %>%
  group_by(fold) %>%
  nest() %>%
  mutate(dummy = 1)
```

```
train_test ← data %>%
  inner_join(data, by="dummy") %>%
  select(-dummy) %>%
  filter(fold.y ≠ fold.x) %>%
  group_by(fold.x) %>%
  summarise(test = list(first(data.x)),
            train = list(bind_rows(data.y)))
```

```
`summarise()` ungrouping output (override with `.groups` argu
```

```
train_test %>%
  mutate(modelo = map(train,~lm(mpg ~ wt,data=.x)),
         pred = map2(modelo,test,~predict(.x,.y)),
```

```
data
```

```
# A tibble: 3 x 3
# Groups:   fold [3]
   fold data                dummy
  <int> <list>              <dbl>
1     1 <tibble [11 x 11]>     1
2     2 <tibble [11 x 11]>     1
3     3 <tibble [10 x 11]>     1
```

```
data %>%
  inner_join(data, by="dummy") %>%
  select(-dummy)
```

```
# A tibble: 9 x 4
  fold.x data.x              fold.y data.y
   <int> <list>                <int> <list>
1      1 <tibble [11 x 11]>      1 <tibble [11 x 11]>
2      1 <tibble [11 x 11]>      2 <tibble [11 x 11]>
3      1 <tibble [11 x 11]>      3 <tibble [10 x 11]>
4      2 <tibble [11 x 11]>      1 <tibble [11 x 11]>
5      2 <tibble [11 x 11]>      2 <tibble [11 x 11]>
6      2 <tibble [11 x 11]>      3 <tibble [10 x 11]>
7      3 <tibble [10 x 11]>      1 <tibble [11 x 11]>
8      3 <tibble [10 x 11]>      2 <tibble [11 x 11]>
9      3 <tibble [10 x 11]>      3 <tibble [10 x 11]>
```

```
data %>%
  inner_join(data, by="dummy") %>%
  select(-dummy) %>%
  filter(fold.y != fold.x) %>%
  group_by(fold.x)
```

```
# A tibble: 6 x 4
# Groups:   fold.x [3]
  fold.x data.x              fold.y data.y
   <int> <list>               <int> <list>
1      1 <tibble [11 x 11]>      2 <tibble [11 x 11]>
2      1 <tibble [11 x 11]>      3 <tibble [10 x 11]>
3      2 <tibble [11 x 11]>      1 <tibble [11 x 11]>
4      2 <tibble [11 x 11]>      3 <tibble [10 x 11]>
5      3 <tibble [10 x 11]>      1 <tibble [11 x 11]>
6      3 <tibble [10 x 11]>      2 <tibble [11 x 11]>
```

```
train_test
```

```
# A tibble: 3 x 3
  fold.x test                train
   <int> <list>              <list>
1      1 <tibble [11 x 11]> <tibble [21 x 11]>
2      2 <tibble [11 x 11]> <tibble [21 x 11]>
3      3 <tibble [10 x 11]> <tibble [22 x 11]>
```

```
train_test %>%
  mutate(modelo = map(train,~lm(mpg ~ wt,data=.x)))
```

```
# A tibble: 3 x 4
  fold.x test                train                modelo
   <int> <list>              <list>               <list>
1      1 <tibble [11 x 11]> <tibble [21 x 11]> <lm>
2      2 <tibble [11 x 11]> <tibble [21 x 11]> <lm>
3      3 <tibble [10 x 11]> <tibble [22 x 11]> <lm>
```

```
train_test %>%
  mutate(modelo = map(train,~lm(mpg ~ wt,data=.x)),
         pred = map2(modelo,test,~predict(.x,.y)))
```

```
# A tibble: 3 x 5
  fold.x test                train                modelo pred
   <int> <list>              <list>               <list> <list>
1      1 <tibble [11 x 11~ <tibble [21 x 1~ <lm>   <dbl [11~
2      2 <tibble [11 x 11~ <tibble [21 x 1~ <lm>   <dbl [11~
3      3 <tibble [10 x 11~ <tibble [22 x 1~ <lm>   <dbl [10~
```

```
train_test %>%
  mutate(modelo = map(train,~lm(mpg ~ wt,data=.x)),
         pred = map2(modelo,test,~predict(.x,.y)),
         real = map(test,"mpg"))
```

```
# A tibble: 3 x 6
  fold.x test           train          modelo pred     real
   <int> <list>         <list>         <list> <list>   <list>
1      1 <tibble [11 ~ <tibble [21 ~ <lm>   <dbl [1~ <dbl [~
2      2 <tibble [11 ~ <tibble [21 ~ <lm>   <dbl [1~ <dbl [~
3      3 <tibble [10 ~ <tibble [22 ~ <lm>   <dbl [1~ <dbl [~
```

Más de dos argumentos: `pmap`

Cuando queremos vectorizar operaciones sobre una lista (o vector) usamos `map_`; para iterar sobre dos listas, `map2_`; para más de dos listas se usa `pmap_`

Función `map`

`pmap(.l, .f, ...)`

- `.l` es una lista de listas sobre la que aplicaremos la función,
- `.f` es la función en cuestión; si la definimos en forma anónima, sus argumentos son `..1`, `..2`, `..3`, etc.
- `...` son argumentos adicionales de `.f`

```
bolilleros ← list(1:6,1:50,1:100)
cant ← list(6,5,4)
con_reemplazo ← list(TRUE,FALSE,FALSE)

pmap(list(bolilleros,cant,con_reemplazo),
     ~sample(x = ..1, size = ..2, replace = ..3))
```

```
[[1]]
[1] 3 6 3 3 1 6

[[2]]
[1] 47 41 37 39 21

[[3]]
[1] 93 80 19 86
```

possibly

Ocasionalmente hay funciones que pueden fallar y es necesario manejar los errores

```
c(list.files(pattern="archivo"),"archivo_4.csv") %>%
  tibble(archivos = .) %>%
  mutate(contenido = map(archivos,read.csv)) %>%
  unnest(contenido)
```

```
Error : Problem with ‘mutate()‘ input ‘contenido‘.
x cannot open the connection
i Input ‘contenido‘ is ‘map(archivos, read.csv)‘.
```

Para salvar los errores existe en `purrr` la función `possibly`. Recibe como primer argumento una función y devuelve una nueva función que tiene una salida específica en caso de que falle (parámetro `otherwise`).

```
possibly_read.csv ← possibly(read.csv,otherwise = data.frame())

c(list.files(pattern="archivo"),"archivo_4.csv") %>%
  tibble(archivos = .) %>%
  mutate(contenido = map(archivos,possibly_read.csv))
```

```
# A tibble: 4 x 2
  archivos      contenido
  <chr>         <list>
1 archivo_1.csv <df[,5] [6 x 5]>
2 archivo_2.csv <df[,5] [6 x 5]>
3 archivo_3.csv <df[,5] [6 x 5]>
4 archivo_4.csv <df[,0] [0 x 0]>
```

Resumen

Referencias

- Bryan, Jenny. (2019) *Purrr tutorial*. Link.
- Barterm, Rebecca. (2019). *Learn to purrr*. Link.
- Wickham, Hadley (2019). *Advanced R – 2nd Edition*. Capítulo 9. CRC Press. Link.
- Baumer B., Kaplan D. y Horton N. (2021). *Modern Data Science with R – 2nd Edition*. Capítulo 7. CRC Press. Link.