Menos for, más purrr: Programación funcional con R

Nacho Evangelista

18 de febrero de 2020

Contenido

1. Introducción
   - Motivación
   - Listas en R
2. purrr
   - La función `map`
   - Acomodando el tipo de salida
   - Funciones anónimas
   - Múltiples argumentos
   - dataframes
3. purrr + tidyr + dplyr
   - Columnas lista y dataframes anidados
   - `mutate` + `purrr`
   - Ejemplos
4. Extra
   - Más de dos argumentos pmap
   - Puede fallar...
   - `repeat`
5. Resumen

A

- Nuestro objetivo: reemplazar los *for loops* o estructuras de repetición. Los lenguajes de programación puramente funcionales utilizan funciones para lograr los mismos resultados.
- 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.
- En una estructura funcional, no es necesario crear una lista vacía para ir guardando resultados, el código es más conciso.
- Premisa: arrancar con porciones de código pequeños y fáciles de entender (funciones). Combinar estos bloques en estructuras más complejas.
- Evitar duplicación de código. La duplicación hace que los errores y *bugs* sean más frecuentes. También se hace más difícil modificar el código.
- Principio: no repetirse a uno mismo (DRY: *don't repeat yourself*)
- R programmers prefer to solve this type of problem by applying an

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.
- https://cran.r-project.org/web/packages/listviewer/index.html

A

- Muchas operaciones de R simplemente funcionan en forma vectorizada; cuando proveemos vectores como entrada, la función se aplica elemento a elemento (una suerte de iteración)
- Muchas funciones no tienen esa capacidad
- `purrr` sirve para iterar
- La función (funcional) más básica de `purrr` es `map`: toma un vector y una función, y aplica la función a cada elemento del vector, devolviendo los resultados en una lista.
- `map(1:3,f)` es equivalente a `list(f(1), f(2), f(3))`

A

A

map2

Pero yo nunca usé listas...

Columnas lista y dataframes anidados

- Some crazy stuff starts happening when you learn that tibble columns can be lists (as opposed to vectors, which is what they usually are).
- For instance, a tibble can be “nested” where the tibble is essentially split into separate data frames based on a grouping variable, and these separate data frames are stored as entries of a list

¿Cómo se construyen?

- En la definición del `tibble`
- Usando `groupby` y `nest`
- Como resultado de una operación.

I. En la definición del `tibble`

II. Usando `groupby` y `nest`

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?

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)
```

```
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)
```

```
# 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

```
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)
```

```
# 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)
```

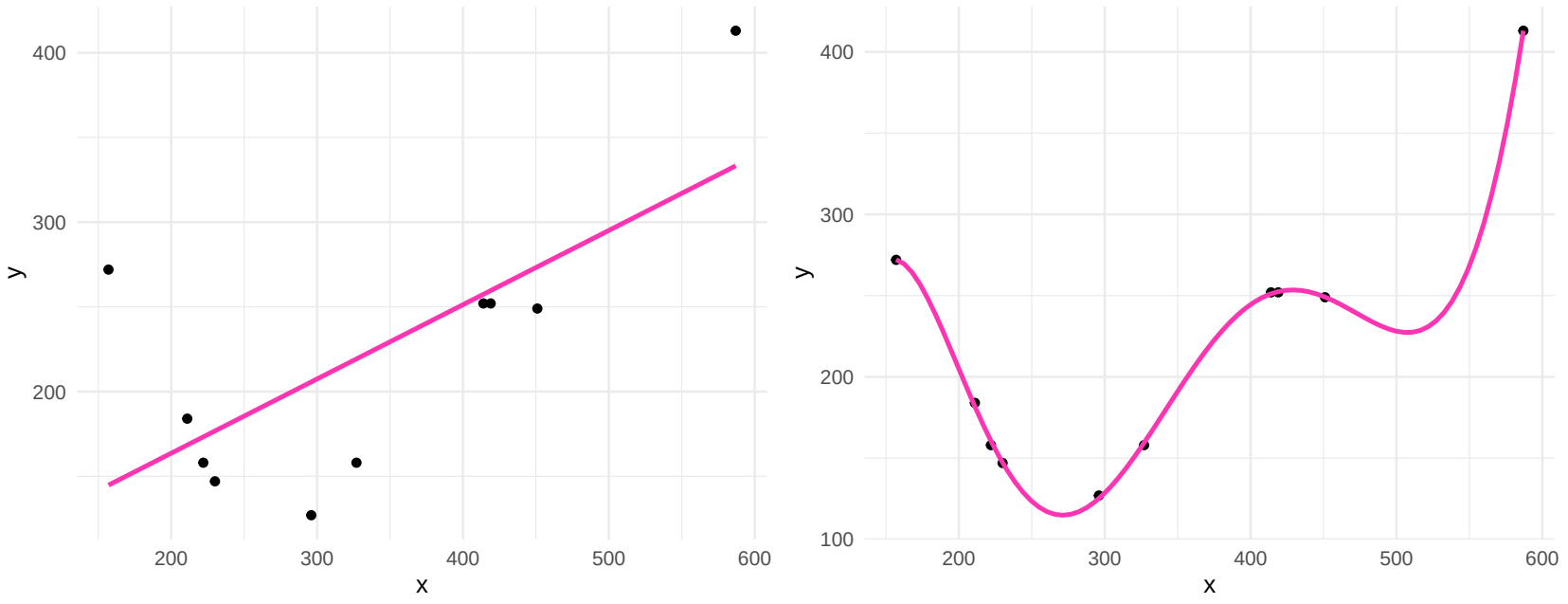
Idea:

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()
                     }))
```

```
# 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>
```

```
plots$plot[[1]] + plots$plot[[6]]
```



6. K-fold cross validation

```
# A tibble: 4 x 6
  fold.x test           train           modelo pred    real
   <int> <list>         <list>          <list> <list>  <list>
1      1 <tibble [8 x~ <tibble [24 x~ <lm>   <dbl [~ <dbl [~
2      2 <tibble [8 x~ <tibble [24 x~ <lm>   <dbl [~ <dbl [~
3      3 <tibble [8 x~ <tibble [24 x~ <lm>   <dbl [~ <dbl [~
4      4 <tibble [8 x~ <tibble [24 x~ <lm>   <dbl [~ <dbl [~
```

possibly

Resumen