

6 Referencias

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 cotidianos (preprocesamiento, modelización, visualización)

- 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.

- 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?

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

```
str(animais)
```

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
```

```
names(animaitos)
```

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



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

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

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

```
[1] 5
```


El paquete purrr

- Muchas operaciones de R funcionan en forma vectorizada; cuando se aplican a vectores, se ejecutan 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

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(animales,typeof)
```

```
$animales
[1] "character"
```

```
$acciones
[1] "character"
```

```
$nombres
[1] "list"
```

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


También podemos usar `purrr` con vectores

```
v ← c(1,3)
map(v, rnorm)
```

```
[[1]]
[1] -0.2195367
```

```
[[2]]
[1] -0.9003695  1.1688635  1.1648648
```

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

```
[[1]]
[1] 5.352383
```

```
[[2]]
[1] 1.995069 4.317735 5.200050 4.401210 4.983714
```

Especificando 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(animales,typeof)
```

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

```
map_int(animales, 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] 3.673015
```

```
[[2]]
[1] 3.339782 1.161005 6.541389 6.352696 6.403535
```

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] 1.823143 1.018668
```

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256	257	258	259	260	261	262	263	264	265	266	267	268	269	270	271	272	273	274	275	276	277	278	279	280	281	282	283	284	285	286	287	288	289	290	291	292	293	294	295	296	297	298	299	300	301	302	303	304	305	306	307	308	309	310	311	312	313	314	315	316	317	318	319	320	321	322	323	324	325	326	327	328	329	330	331	332	333	334	335	336	337	338	339	340	341	342	343	344	345	346	347	348	349	350	351	352	353	354	355	356	357	358	359	360	361	362	363	364	365	366	367	368	369	370	371	372	373	374	375	376	377	378	379	380	381	382	383	384	385	386	387	388	389	390	391	392	393	394	395	396	397	398	399	400	401	402	403	404	405	406	407	408	409	410	411	412	413	414	415	416	417	418	419	420	421	422	423	424	425	426	427	428	429	430	431	432	433	434	435	436	437	438	439	440	441	442	443	444	445	446	447	448	449	450	451	452	453	454	455	456	457	458	459	460	461	462	463	464	465	466	467	468	469	470	471	472	473	474	475	476	477	478	479	480	481	482	483	484	485	486	487	488	489	490	491	492	493	494	495	496	497	498	499	500	501	502	503	504	505	506	507	508	509	510	511	512	513	514	515	516	517	518	519	520	521	522	523	52
--	---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	----

For "None of the above" "None of the above" "None of the above" "None of the above"

[redacted] "0010 00 00" "0010 00 00" "0010 00 00"


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

```

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

```

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

cancion	autor	pelicula
0	0	1

- 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
- II Usando nest (y groupby)
- III Como resultado de una operación.


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



```
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]>
```




```
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
```


Aplicar map a un tibble realiza operaciones por columna. Con mutate + map hacemos magia por filas.

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]>

1. *Journal of the American Medical Association*, 1997; 278: 1019-1024.

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) %>%  
  ungroup()
```


	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
```

	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]>
```

```
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 pudo	El que no hace palmas tiene ~
7	Los Palmer~	La cola	A la una, a la dos, a la one~

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

- 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)
```

```
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	canción	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)
```

- 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(orden = 1:6, datos) %>%
  nest(datos = !orden) %>%
  ungroup() %>%
  mutate(plot = map2(datos, orden,
    function(.x, .y) {
      ggplot(.x, aes(x = x, y = y)) +
        geom_point() +
        stat_smooth(
          method = "lm", se = FALSE,
          formula = y ~ poly(x,
            degree=.y,
            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
```

```
6      6 <tibble [10 x 2]>
```

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

```
# A tibble: 6 x 3
```

orden datos

plot

```
<int> <list>
```

```
<list>
```

```
1 1 <tibble [10 x 2]> <gg>
```

```
2      2 <tibble [10 x 2]> <gg>
```

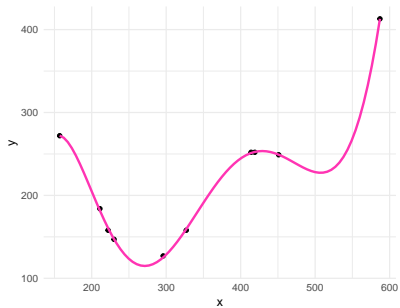
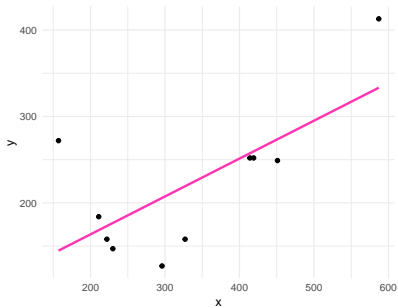
```
3      3 <tibble [10 x 2]> <gg>
```

```
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

```
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)))
```

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

data

```
# A tibble: 3 x 3
```

```
# Groups: fold [3]
```

fold data

dummy

```
<int> <list>
```

<db1>

```
1      1 <tibble [11 x 11]>      1
```

```
2      2 <tibble [11 x 11]>      1
```

```
3      3 <tibble [10 x 11]>      1
```

```
1 <tibble [11 x 11]>
2 <tibble [11 x 11]>
3 <tibble [10 x 11]>
1 <tibble [11 x 11]>
2 <tibble [11 x 11]>
3 <tibble [10 x 11]>
1 <tibble [11 x 11]>
2 <tibble [11 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 [~

- `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`


```
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]>
```


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](#).

