Herramientas Computacionales, Algoritmos y Machine Learning (HCML)

Clase 7: Aplicación de funciones (`apply` / `purrr`)

Constanza Prado – Alex Antequeda – Diego Muñoz

Clase 7: Aplicación de funciones

Familia `apply`

- Función `apply`.
- Función `tapply`.
- Función `lapply`.
- Función `sapply`.

Paquete `purrr`

- Introducción.
- Función `map` y variantes.
- Función `map2` y `pmap`.
- Función `walk`, `nest` y otras.

Actividad

Material complementario

Familia `apply`

Familia `apply`

La familia de funciones `apply` son comandos de R base que permiten aplicar una función para distintos valores del argumento de manera automática. A continuación, se presentará la función `apply` y sus distintas versiones:

Función `apply()`

`apply(X = , MARGIN = , FUN = )`: Se aplica una función FUN a una matriz X. La operación puede ser aplicada por fila (MARGIN = 1) o por columna (MARGIN = 2).

Ejemplo:

```
apply(X = iris[,1:4], MARGIN = 2, FUN = sum)
```

```
## Sepal.Length  Sepal.Width Petal.Length  Petal.Width
##        876.5        458.6        563.7        179.9
```

```
apply(X = iris[,1:4], MARGIN = 1, FUN = sum) %>% head(10)
```

```
##  [1] 10.2  9.5  9.4  9.4 10.2 11.4  9.7 10.1  8.9  9.6
```

Familia `apply`

Función `tapply()`

`tapply(X = , INDEX = , FUN = )`: Se aplica una función FUN al objeto X segregando por el objeto INDEX.

Ejemplo:

```
tapply(X = iris$Sepal.Length, INDEX = iris$Species, FUN = mean)
```

```
##     setosa versicolor  virginica
##      5.006      5.936      6.588
```

Familia `apply`

Función `lapply()`

`lapply(X = , FUN = )`: Se aplica una función FUN a la lista X. Se retorna un objeto en formato lista.

Ejemplo:

```
Lista = list("Elemento 1" = 1:10,
             "Elemento 2" = 11:20,
             "Elemento 3" = 21:30)

lapply(X = Lista, FUN = median)
```

```
## $`Elemento 1`
## [1] 5.5
##
## $`Elemento 2`
## [1] 15.5
##
## $`Elemento 3`
## [1] 25.5
```

Familia `apply`

Función `sapply()`

`sapply(X = , FUN = )`: Se aplica una función FUN a la lista X. Se retorna un objeto en formato vector.

Ejemplo:

```
Lista = list("Elemento 1" = 1:10,
             "Elemento 2" = 11:20,
             "Elemento 3" = 21:30)

sapply(X = Lista, FUN = median)
```

```
## Elemento 1 Elemento 2 Elemento 3
##        5.5       15.5       25.5
```

Paquete `purrr`

Paquete `purrr`

El paquete `purrr` contiene múltiples funciones para trabajar con vectores/listas y funciones, con el objetivo de mejorar la manera en que se aplican funciones a vectores, listas, etc. Lo anterior, permite ahorrar posibles procesos con iteraciones, transformándolo en un código más limpio y de fácil lectura.

Este paquete es fundamental para la llamada "programación funcional", la cual busca una programación limpia basada en la aplicación y composición de funciones. Se busca que el cálculo se base en funciones puras, estas son funciones que siempre retornarán el mismo resultado, sin posibilidad de que elementos externos o internos modifiquen su funcionamiento.

Familia `map`

La familia **`map`** es la equivalente al **`lapply`**. Permite aplicar funciones a elementos de listas o vectores.

Función `map()`

La función **`map`** permite aplicar funciones a los elementos de vectores/listas. La función `map` siempre retorna una lista.

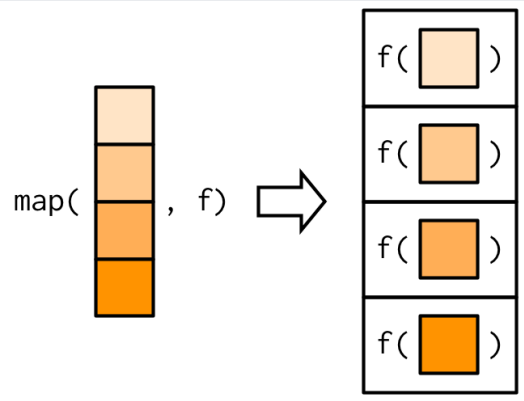
```
Lista = list("Objeto 1" = 0:100,
             "Objeto 2" = 35:75)

purrr::map(Lista, mean)
```

```
## $`Objeto 1`
## [1] 50
##
## $`Objeto 2`
## [1] 55
```

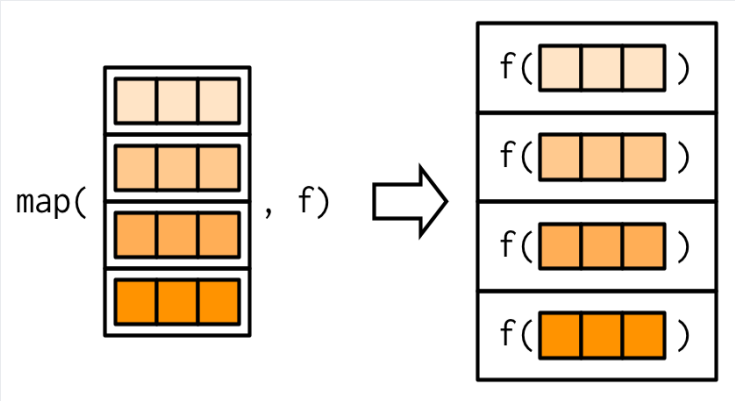
```
purrr::map(c("Uno" = 1,
             "Cuatro" = 4,
             "Nueve" = 9), sqrt)
```

```
## $Uno
## [1] 1
##
## $Cuatro
## [1] 2
##
## $Nueve
## [1] 3
```

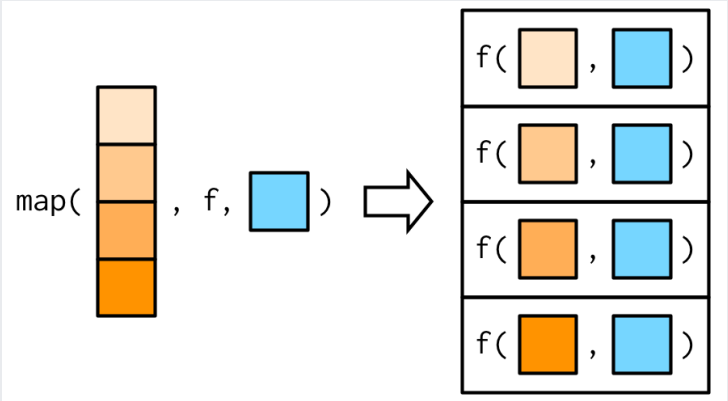
Ilustraciones, uso función `map()`



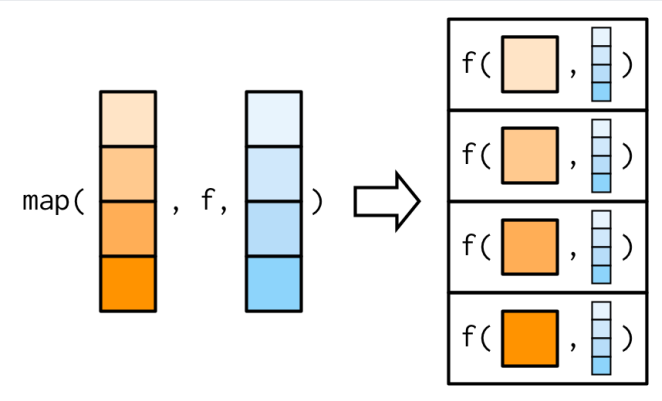
Aplicado a un vector



Aplicado a una lista



Aplicado a un vector y a un elemento



Aplicado a dos vectores

Fuente : Advanced R

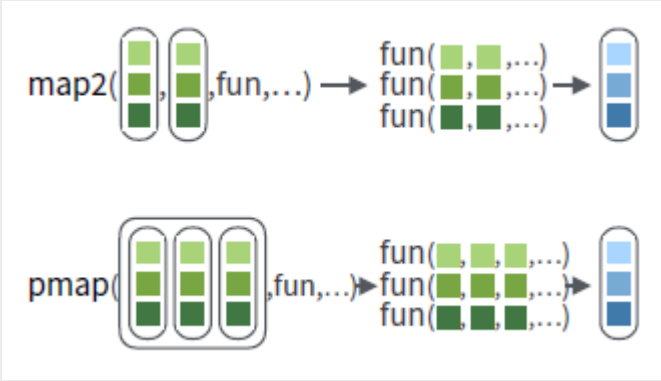
Otras funciones derivadas de `map`

Para cada función existen variantes que modifican el tipo de objeto que retorna la función. Estas son,

Función	Resultados
`map`	Lista
`map_chr`	Vector de carácteres
`map_dbl`	Vector numérico (double)
`map_dfc`	data frame (columnas)
`map_dfr`	data frame (filas)
`map_int`	Vector de enteros (integer)
`map_lgl`	Vector lógico

Otras funciones derivadas de `map`

- `map2(x, y, f, ...)`: Aplica una función a pares de elementos a partir de dos listas/vectores. Entrega una lista.
- `pmap(..., f, ...)`: Es una generalización de map2 a un número indeterminado p de argumentos enlistados. Entrega una lista.



Estas funciones también tienen alternativas que permiten modificar el tipo de resultado obtenido anteriormente, añadiendo alguno de los sufijos `_chr`, `_dbl`, `_dfc`, etc.

Ejemplo utilizando la función `map2`

Se crea una función que genera un cálculo simple con 2 argumentos x, y. Esta función llamará `Ejemplo`:

```
Ejemplo = function(x,y){
  log(x) + y^2
}
```

Se aplica la función `Ejemplo` a dos vectores de largo 3, a través de la función map2, obteniendo una lista:

```
map2(1:3, 11:13, Ejemplo)
```

```
## [[1]]
## [1] 121
##
## [[2]]
## [1] 144.6931
##
## [[3]]
## [1] 170.0986
```

Se aplica la función `Ejemplo` a dos vectores de largo 3, a través de la función map2_dbl, obteniendo un vector:

```
map2_dbl(1:3, 11:13, Ejemplo)
```

```
## [1] 121.0000 144.6931 170.0986
```

Actividad

1. Usando la función `map_dbl` obtenga los primeros 10 números pares.

2. Recuerde el ejemplo de la clase anterior, donde se quería retornar un mensaje de felicitaciones o de reprobación dependiendo de la nota de un alumno.

```
case_when(nota < 1 ~ "Error, ingrese un número entre 1 y 7.",
          nota > 7 ~ "Error, ingrese un número entre 1 y 7.",
          is.numeric(nota) == FALSE ~ "Error, ingrese un número.",
          nota >= 4 ~ "¡Felicitaciones!",
          nota <4 ~ "Reprobaste")
```

Cree una función que retorne una tabla en formato data.frame con la nota y mensaje del alumno. Suponga ahora que se tienen las notas de 10 alumnos, las que se encuentran en el código del final del ejercicio . ¿Cuál sería el resultado de aplicar la función creada con `map`? ¿Qué ventajas se tendrían al utilizar `map_dfr`?

```
notas <- c(6.8, 5.5, 3.2, 4.3, 2.1,
           6.5, 4.7, 5.9, 6.2, 3.8)
```

Otras funciones paquete `purrr` y complementarias.

Función `nest`

La función `nest()` de `tidyr` permite transformar una base de datos a una *base anidada*. Es decir, agrupa la base de datos original de tal manera que cada fila representa una base de datos diferente. Obteniendo una "base de datos de bases de datos".

```
iris %>% group_by(Species) %>% nest()
```

```
## # A tibble: 3 x 2
## # Groups:   Species [3]
##   Species    data
##   <fct>      <list>
## 1 setosa     <tibble[,4] [50 x 4]>
## 2 versicolor <tibble[,4] [50 x 4]>
## 3 virginica  <tibble[,4] [50 x 4]>
```

Usualmente, se realiza un `group_by()` para poder anidar por una variable. De ser necesario, el comando `group_nest()` permite anidar directamente por una variable. En este caso, la base de datos **no** queda agrupada, por lo que es útil cuando no se necesita la agrupación para realizar otros cálculos.

El comando `unnest()` permite transformar una base anidada en una normal.

Otras funciones paquete `purrr` y complementarias.

Funciones `walk`

Las funciones **`walk()`**, **`walk2()`** y **`pwalk()`** son muy similares a las funciones `map` vistas anteriormente. Sin embargo, estas son utilizadas cuando se desea aplicar *una acción* en vez de generación. Algunos ejemplos de acciones son: `save()`, `ggsave()`, `write_csv()`, `print()`, etc.

Ejemplo funciones `walk` y `nest`

En el siguiente ejemplo, se utilizará la base de datos `paises` de la librería `datos`. El objetivo será guardar una base de datos para cada continente en archivos `csv`.

```
glimpse(paises)
```

```
## Rows: 1,704
## Columns: 6
## $ pais              <fct> "Afganistán", "Afganistán", "Afganistán", "Afgani
## $ continente        <fct> Asia, Asia, Asia, Asia, Asia, Asia, Asia, Asia, A
## $ anio              <int> 1952, 1957, 1962, 1967, 1972, 1977, 1982, 1987, 1
## $ esperanza_de_vida <dbl> 28.801, 30.332, 31.997, 34.020, 36.088, 38.438, 3
## $ poblacion         <int> 8425333, 9240934, 10267083, 11537966, 13079460, 1
## $ pib_per_capita    <dbl> 779.4453, 820.8530, 853.1007, 836.1971, 739.9811,
```

Ejemplo funciones `walk` y `nest`

Para el guardado, se utilizará la función **`write_csv()`** de `readr`, la cual tiene los siguientes argumentos:

```
write_csv(x = base_datos ,file = "ruta/nombre_archivo.csv")
```

Este comando guarda el objeto de R `base_datos` en el archivo `nombre_archivo.csv` en la ruta seleccionada.

Para cumplir este objetivo, se deben realizar los siguientes pasos:

1. Crear una carpeta llamada `Bases_Continentes` para guardar las bases de datos. Use el comando **`dir.create()`**.

2. Utilizar la función **`group_nest()`** para crear una base de datos anidada según la variable continente.

Ejemplo funciones `walk` y `nest`

3. Cree una nueva variable llamada `file` que contenga la ruta, nombre del continente y extensión del archivo a guardar (ej. "Bases_Continentes/Asia.csv"). Utilice el comando `paste0()` para unir los strings.

4. Elimine la variable correspondiente a continentes y cambie el nombre de la variable `data` a `x`.

5. Utilice la función `pwalk` para aplicar la función `write_csv()` a las columnas de la base de datos.

La base de datos es renombrada para que las columnas respectivas contengan los nombres de los argumentos de la función `write_csv()`, de esa forma, al usar el comando `pwalk()`, no habrán problemas en la ejecución.

Solución: Ejemplo funciones `walk` / `nest`.

En el siguiente bloque de código, se encuentra la solución del ejemplo anterior.

```
## Pregunta 1 ##
dir.create("Bases_Continentes")

paises %>%
  ## Pregunta 2 ##
  group_nest(continente) %>%

  ## Pregunta 3 ##
  mutate(file = paste0("Bases_Continentes/", continente, ".csv")) %>%

  ## Pregunta 4 ##
  select(file, x = data) %>%

  ## Pregunta 5 ##
  pwalk(write_csv)
```

Actividad Final

La base de datos `esperanza.xlsx`(*) contiene información de la esperanza de vida junto con distintas variables de resumen de salud a nivel nacional de 102 países desde los años 2000 a 2014. El objetivo de esta actividad será crear un procedimiento en R que permita generar y guardar tablas de resumen anual, con el fin de encontrar diferencias entre países desarrollados y en vías de desarrollo por año. El procedimiento debe generar archivos .csv independientes.

1.- Defina una función que, utilizando un año en particular de la base de datos `esperanza.xlsx`, genere una tabla de resumen de las diferencias entre países desarrollados y en vías de desarrollo. En particular, la tabla debe contener la siguiente información:

- Número de países desarrollados/en vías de desarrollo.
- Media de la esperanza de vida.
- Media del consumo de alcohol.
- Media de los años de escolaridad.
- Media de las muertes por VIH-SIDA.

(*) En la página 23 de este documento, se encuentra una descripción de las variables de la base de datos.

Actividad Final

2.- Crear una carpeta llamada `Bases_Anuales` para guardar las bases de datos. Use el comando `dir.create()`.

3.- Utilizar la función `group_nest()` para crear una base de datos anidada según la variable year.

4.- Cree una nueva variable llamada `file` que contenga la ruta, nombre del continente y extensión del archivo a guardar (ej. "Bases_Anuales/2004.csv"). Utilice el comando `paste0()` para unir los strings.

5.- Utilice la función `map` para generar una nueva variable llamada `tabla_resumen`, que guarde para cada año la tabla generada con la función de la pregunta 1.

6.- Elimine las variables year y data. Cambie el nombre de la variable `tabla_resumen` a `x`.

7.- Utilice la función `pwalk` para aplicar la función `write_csv()` a las columnas de la base de datos.

Descripción de las variables

Variable	Descripción
country	País.
year	Año.
status	Estatus del país, desarrollado o en vías de desarrollo.
life_expectancy	Esperanza de vida en años.
adult_mortality	Probabilidad de morir entre los 15 y 60 años (por 1000 habitantes).
infant_deaths	Número de muertes infantiles por 1000 habitantes.
alcohol	Alcohol consumido per-cápita en litros (mayores a 15 años).
hepatitis_b	Inmunizacion de hepatitis-B en niños de 1 año (%)
bmi	índice de Masa Corporal (IMC) promedio.
under-five deaths	Número de muertes bajo 5 años por 1000 habitantes.
polio	Inmunización de polio en niños de 1 año (%).
diphtheria	Inmunización de la difteria, tuétano y pertuosis (%).
hiv_aids	Muertes por 1000 nacimientos por VIH/SIDA.
schooling	Número de años de escolaridad.

Referencias y material complementario

- Página web del paquete **`purrr`** (en inglés).
- Cheatsheet de `purrr` (en inglés).
- Cheatsheet de `purrr` (en español).
- Cheatsheet de R Avanzado (en inglés).

¡Gracias!