Sección 4 - dplyr, purrr, tidy

Ana Cecilia Plavan, Andrés Andruskevicius

Funciones para la manipulación de datos:

- mutate(): Agrega nuevas variables
- select(): Selecciona variables por su nombre
- filter(): filtra variables basado en condiciones
- arrange(): Cambia el orden de columnas

- summarise()
- group_by()
- pull()
- relocate()

Función summarize() - summarise()

- Crea un nuevo data frame con las columnas deseadas

```
summarise(X=dataframe, COL)
```

  - COL: Columnas nuevas a retornar en el data frame resultado.

```
> murders%>% summarise(suma=sum(total),valorMedio=median(total),
prom=mean(total))
  suma valorMedio   prom
1 9403         97 184.37
```

Función group_by()

- Agrupar por una o mas variables obteniendo

```
group_by(X=dataframe, COND)
```

  - COND: Condiciones lógicas a filtrar.

```
> murders%>% group_by(region)
# A tibble: 51 x 5
# Groups:   region [4]
   state                abb   region
   <chr>                <chr> <fct>
 1 Alabama              AL    South
 2 Alaska               AK    West
 3 Arizona              AZ    West
 4 Arkansas             AR    South
 5 California           CA    West
 6 Colorado             CO    West
 7 Connecticut          CT    North~
 8 Delaware             DE    South
 9 District of Co~      DC    South
10 Florida              FL    South
# ... with 41 more rows, and 2
#   more variables:
#   population <dbl>,
#   total <dbl>
```

Resumir datos

- Funciones summarize() y group_by()

```
> murders%>% group_by(region) %>% summarise(suma=sum(total),
valorMedio=median(total),prom=mean(total))
# A tibble: 4 x 4
  region          suma valorMedio  prom
  <fct>          <dbl>      <dbl> <dbl>
1 Northeast       1469         97  163.
2 South           4195        207  247.
3 North Central   1828         80  152.
4 West            1911         36  147
```

- Summarise se aplica a cada grupo por separado

Funciones útiles para trabajar en consola

- Pull: Extraer un vector de un dataFrame para trabajar localmente con los datos
- Similar al operador '.'

pull{base}

Función pull()

- Acceder a una columna del dataframe

```
pull(X=dataframe, columna , name = NULL)
```

  - columna: Vector a extraer
    - Entero n (1:last_col):
      - Positivo: posición del vector, contada desde la izquierda
      - Negativo: posición del vector, contada desde la derecha
    - Literal: Nombre del vector
  - name: Nuevo nombre del vector

Ejemplo función pull()

```
> notas
    alumnos calificacion edades
1       Ana            D       9
2     Pablo            B       8
3     Pedro          MBS       9
4 Marianela          MBS       9
5      <NA>            B       7
6     Maria         <NA>      NA
7      Pepe         <NA>      NA
```

```
> head(heights)
     sex height
1   Male     75
2   Male     70
3   Male     68
4   Male     74
5   Male     61
6 Female     65
```

```
> result<-notas %>% mutate(sexo=heights$sex[1:7],altura=heights$height[1:7]) %>%
  summarise(mean(altura))
> result
  mean(altura)
1       68.429
> class(result)
[1] "data.frame"
```

```
    alumnos calificacion edades   sexo altura
1       Ana            D       9   Male     75
2     Pablo            B       8   Male     70
3     Pedro          MBS       9   Male     68
4 Marianela          MBS       9   Male     74
5      <NA>            B       7   Male     61
6     Maria         <NA>      NA Female     65
7      Pepe         <NA>      NA Female     66
```

```
> result<-pull(result)
> class(result)
[1] "numeric"
```

Paquete dplyr

- En adición a data frames/tibbles:
  - dtplyr: Grandes data sets de datos
  - dbplyr: Datos almacenados en bases de datos relacionales (SQL)
  - sparklyr: Apache Spark

```
# Todo tidyverse:
install.packages("tidyverse")

# Instalar solo dplyr:
install.packages("dplyr")
```

- Datos con un formato ordenado y estructurado donde:
  - Cada columna es una variable
  - Cada fila es una observación
  - Cada celda contiene un único valor

```
#>       country year fertility
#> 1     Germany 1960      2.41
#> 2 South Korea 1960      6.16
#> 3     Germany 1961      2.44
#> 4 South Korea 1961      5.99
#> 5     Germany 1962      2.47
#> 6 South Korea 1962      5.79
```

```
#>       country 1960 1961 1962
#> 1     Germany 2.41 2.44 2.47
#> 2 South Korea 6.16 5.99 5.79
```

Dataframe

```
 alumnos<-
data.frame(nombres=c("Juan","Pedro","Paco","Ana"
,NA,"Maria","Cecilia",NA),calificacion=c("MBS",N
A,"B","B","MBS","MBS","D",NA),edad=c(13,12,NA,10
,13,10,12,NA))
```

```
> alumnos
  nombres calificacion edad
1    Juan          MBS   13
2   Pedro         <NA>   12
3    Paco            B   NA
4     Ana            B   10
5    <NA>          MBS   13
6   Maria          MBS   10
7 Cecilia            D   12
8    <NA>         <NA>   NA
> class(alumnos)
[1] "data.frame"
```

Tibbles

- Tipo especial de data frames
- Las funciones goup_by y summarize retornan este tipo

```
>alumnos2<-tibble(nombres=c("Juan","Pedro","Paco",
"Ana",NA,"Maria","Cecilia",NA),calificacion=c("MBS",NA,"B"
,"B","MBS","MBS","D",NA),edad=c(13,12,NA,10,13,10,12,NA))
> class(alumnos2)
[1] "tbl_df"     "tbl"        "data.frame"
```

```
> alumnos2
# A tibble: 8 x 3
  nombres calificacion  edad
  <chr>   <chr>        <dbl>
1 Juan    MBS             13
2 Pedro   NA              12
3 Paco    B               NA
4 Ana     B               10
5 NA      MBS             13
6 Maria   MBS             10
7 Cecilia D               12
8 NA      NA              NA
```

1. tibbles tienen una mejor visualización en consola

- Se muestran más datos de las variables
- Adapta el contenido a la pantalla

```
> murders<-as_tibble(murders)
> murders
# A tibble: 51 x 5
   state                abb   region     population total
   <chr>                <chr> <fct>           <dbl> <dbl>
 1 Alabama              AL    South         4779736   135
 2 Alaska               AK    West           710231    19
 3 Arizona              AZ    West          6392017   232
 4 Arkansas             AR    South         2915918    93
 5 California           CA    West         37253956  1257
 6 Colorado             CO    West          5029196    65
 7 Connecticut          CT    Northeast     3574097    97
 8 Delaware             DE    South          897934    38
 9 District of Columbia DC    South          601723    99
10 Florida              FL    South        19687653   669
# ... with 41 more rows
```

2. Subset de tibbles

```
> murders$population
 [1]  4779736   710231  6392017  2915918 37253956  5029196  3574097   897934
 [9]   601723 19687653  9920000  1360301  1567582 12830632  6483802  3046355
[17]  2853118  4339367  4533372  1328361  5773552  6547629  9883640  5303925
[25]  2967297  5988927   989415  1826341  2700551  1316470  8791894  2059179
[33] 19378102  9535483   672591 11536504  3751351  3831074 12702379  1052567
[41]  4625364   814180  6346105 25145561  2763885   625741  8001024  6724540
[49]  1852994  5686986   563626
> class(murders)
[1] "data.frame"
> class(as_tibble(murders$population))
[1] "tbl_df"     "tbl"        "data.frame"
> class(murders$population)
[1] "numeric"
```

- Wanings en caso de no encontrar los datos:

```
> murders$POPulation
NULL
> murders<-as_tibble(murders)
> murders$POPulation
NULL
Warning message:
Unknown or uninitialised column: `POPulation`.
```

3. Tibbles pueden contener datos complejos

- Data frames contenían vectores numéricos, textos o lógicos.

- Tibbles pueden contener objetos como listas o funciones.

```
> tibble(id = c(1, 2, 3), func = c(mean, median, sd))
# A tibble: 3 x 2
     id func
  <dbl> <list>
1     1 <fn>
2     2 <fn>
3     3 <fn>
```

3. El operador '.'

- Extraer el valor de una variable

```
> filter(murders, region == "South") %>%
+     mutate(rate = total / population * 10^5) %>%
+     summarize(median = median(rate))
# A tibble: 1 x 1
  median
   <dbl>
1   3.40
> filter(murders, region == "South") %>%
+     mutate(rate = total / population * 10^5) %>%
+     summarize(median = median(rate)) %>% .$median
[1] 3.398069
```

3. El paquete purrr

- Paquete para controlar la clase del resultado de funciones

```
compute_s_n <- function(n){
   x <- 1:n
   sum(x)
}
> compute_s_n(25)
[1] 325
```

- Utilizando función sappy:

```
> n <- 1:25
> s_n <- sapply(n, compute_s_n)
> s_n
 [1]   1   3   6  10  15  21  28  36  45  55  66  78  91 105 120
136 153 171
[19] 190 210 231 253 276 300 325
> class(s_n)
[1] "integer"
```

Exigiendo tipos de datos específicos

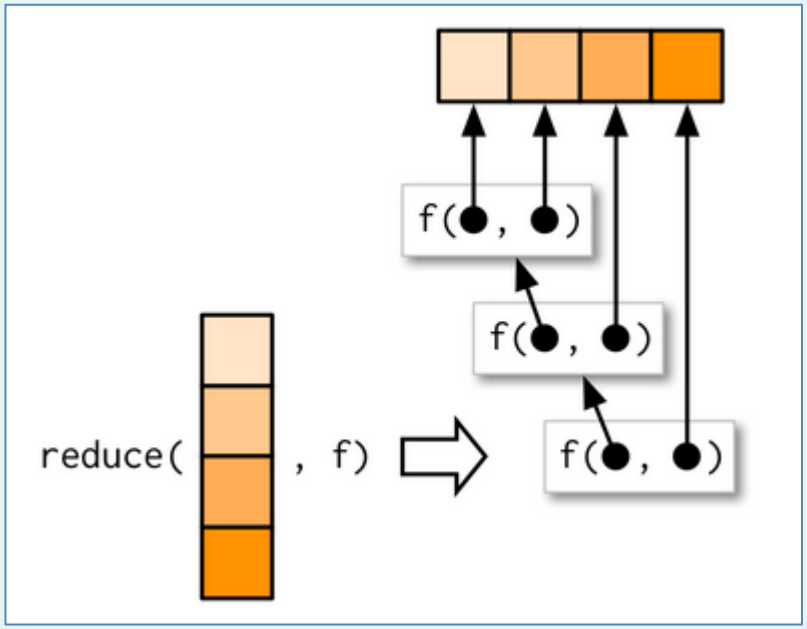
```
> map_dbl(n,compute_s_n)
 [1]   1   3   6  10  15  21  28  36  45  55  66  78
[13]  91 105 120 136 153 171 190 210 231 253 276 300
[25] 325
> class(map_dbl(n,compute_s_n))
[1] "numeric"
> class(map_chr(n,compute_s_n))
[1] "character"
> map_chr(n,compute_s_n)
 [1] "1"   "3"   "6"   "10"  "15"  "21"  "28"  "36"
 [9] "45"  "55"  "66"  "78"  "91"  "105" "120" "136"
[17] "153" "171" "190" "210" "231" "253" "276" "300"
[25] "325"
```

Con funciones anónimas

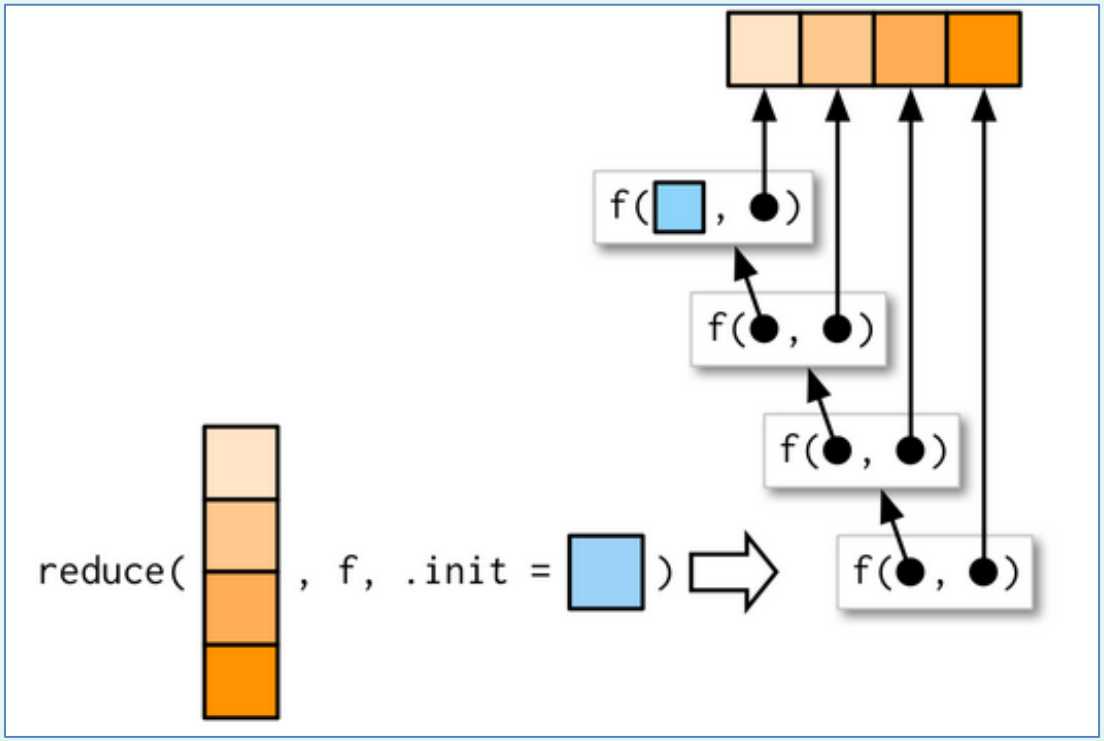
```
> map_dbl(murders, function(x) length(unique(x)))
    state       abb    region population     total
       51        51         4         51        47
```

Función reduce()

- Toma un vector de tamaño n y produce la salida ejecutando una función con cada par de sus elementos.
- reduce(1:4, f) ~ f(f(f(1, 2), 3), 4)



Función reduce()



Ejemplo función reduce()

```
> l <- map(1:4, function(x) sample(1:15, 10))
> l
[[1]]
 [1] 14  6  5  7  2 10 11  4 13  1

[[2]]
 [1] 12 10  6 13  7  5 14  9  1  8

[[3]]
 [1]  6  8  5 13 15 10  3  9  7 14

[[4]]
 [1] 11  3  4  1  9  2 10 13  5  8
```

```
> comunes<-l[[1]]
> comunes
 [1] 14  6  5  7  2 10 11  4 13  1
> comunes<-intersect(comunes,l[[2]])
>
> comunes
[1] 14  6  5  7 10 13  1
> comunes<-intersect(comunes,l[[3]])
>
> comunes
[1] 14  6  5  7 10 13
> comunes<-intersect(comunes,l[[4]])
>
> comunes
[1]  5 10 13
```

Ejemplo función reduce()

```
> reduce(l,intersect)
[1]  5 10 13
> reduce(l,union)
 [1] 14  6  5  7  2 10 11  4 13  1 12  9  8 15  3
```

Función accumulate()

- En lugar de retornar solo el resultado final de reduce(), retorna también todos los resultados intermedios.

```
> accumulate(l, intersect)
[[1]]
 [1] 14  6  5  7  2 10 11  4 13  1

[[2]]
[1] 14  6  5  7 10 13  1

[[3]]
[1] 14  6  5  7 10 13

[[4]]
[1]  5 10 13
```

Condicionales de Tidyverse

- Función case_when():

  Obtenemos varias salidas posibles, a diferencia del ifelse

```
> x <- c(-2, -1, 0, 1, 2)
> case_when(x < 0 ~ "Negative",
+           x > 0 ~ "Positive",
+           TRUE  ~ "Zero")
[1] "Negative" "Negative" "Zero"     "Positive" "Positive"
```

Condicionales de Tidyverse

- Función between():

  Determinar valores por intervalos

```
> x <- c(-2, -1, 0, 1, 2)
> x >= 0 & x <= 1
[1] FALSE FALSE  TRUE  TRUE FALSE
> between(x,0, 1)
[1] FALSE FALSE  TRUE  TRUE FALSE
```

Función filter_all()

- Filtrar datos evaluando una condición para un grupo de variables:

```
filter_all(X=dataframe,condiciones)
```

- Las condiciones lógicas deben utilizar:
  - all_vars(): Todas las variables tienen que cumplir la condición lógica
  - any_vars(): Por lo menos una de las variables tiene que cumplir la condición lógica

Ejemplo función filter_all

```
> datos<-data.frame(edad=c(13,12,NA,10,13,10,12,NA),
resultado1=c(15,10,9,NA,11,15,5,NA),
resultado2=c(16,5,11,NA,14,15,NA,NA))
```

```
> datos
  edad resultado1 resultado2
1   13         15         16
2   12         10          5
3   NA          9         11
4   10         NA         NA
5   13         11         14
6   10         15         15
7   12          5         NA
8   NA         NA         NA
```

```
> datos %>%  filter_all(all_vars
(!is.na(.)))
  edad resultado1 resultado2
1   13         15         16
2   12         10          5
3   13         11         14
4   10         15         15
```

```
> datos %>%  filter_all(all_vars(.>12))
  edad resultado1 resultado2
1   13         15         16
>
> datos %>%  filter_all(any_vars(.>12))
  edad resultado1 resultado2
1   13         15         16
2   13         11         14
3   10         15         15
>
> datos %>%  filter_all(any_vars(.<12))
  edad resultado1 resultado2
1   12         10          5
2   NA          9         11
3   10         NA         NA
4   13         11         14
5   10         15         15
6   12          5         NA
```

Función filter_at()

- Filtrar columnas específicas evaluando una condición lógica:

```
filter_at(X=dataframe,columnas,condiciones)
```

- **columnas**: Que vectores deben de cumplir la condición a evaluar
- **condiciones**:
  - all_vars(): Todas las variables tienen que cumplir la condición lógica
  - any_vars(): Por lo menos una de las variables tiene que cumplir la condición lógica

Ejemplo función filter_at

```
> datos
  edad resultado1 resultado2
1   13         15         16
2   12         10          5
3   NA          9         11
4   10         NA         NA
5   13         11         14
6   10         15         15
7   12          5         NA
8   NA         NA         NA
```

```
> datos %>%  filter_at(vars(edad,
resultado1),all_vars(!is.na(.)))
  edad resultado1 resultado2
1   13         15         16
2   12         10          5
3   13         11         14
4   10         15         15
5   12          5         NA
```

```
> datos %>%  filter_at(vars(resultado1,
resultado2),all_vars(.>12))
  edad resultado1 resultado2
1   13         15         16
2   10         15         15
>
> datos %>%  filter_at(vars(resultado1,
resultado2),any_vars(.>12))
  edad resultado1 resultado2
1   13         15         16
2   13         11         14
3   10         15         15
>
> datos %>%  filter_at(vars(resultado1,
resultado2),any_vars(.<12))
  edad resultado1 resultado2
1   12         10          5
2   NA          9         11
3   13         11         14
4   12          5         NA
```

Función filter_if()

- Filtrar características específicas de las columnas, evaluando una condición lógica:

```
filter_if(X=dataframe,prop,condiciones)
```

- **prop**: Características que deben de cumplir las variables sobre las que queremos aplicar nuestras condiciones
- **condiciones**:
  - all_vars(): Todas las variables tienen que cumplir la condición lógica
  - any_vars(): Por lo menos una de las variables tiene que cumplir la condición lógica

Ejemplo función filter_if

```
> alumnos<-data.frame(nombres=c("Juan","Pedro","Paco","Ana",
NA,"Maria","Cecilia",NA),calificacion=c("MBS",NA,"B","B",
"MBS","MBS","D",NA),edad=c(13,12,NA,10,13,10,12,NA),resultado
=c(15,10,9,NA,11,15,5,NA))
```

```
> alumnos
  nombres calificacion edad resultado
1    Juan          MBS   13        15
2   Pedro         <NA>   12        10
3    Paco            B   NA         9
4     Ana            B   10        NA
5    <NA>          MBS   13        11
6   Maria          MBS   10        15
7 Cecilia            D   12         5
8    <NA>         <NA>   NA        NA
```

```
> alumnos %>% filter_if(is.numeric,all_vars(!is.na(.)))
  nombres calificacion edad resultado
1    Juan          MBS   13        15
2   Pedro         <NA>   12        10
3    <NA>          MBS   13        11
4   Maria          MBS   10        15
5 Cecilia            D   12         5
```

Ejemplo función filter_if

```
> alumnos %>% filter_if(is.numeric,all_vars(.>12))
  nombres calificacion edad resultado
1    Juan           MBS   13        15
> alumnos %>% filter_if(is.numeric,any_vars(.>12))
  nombres calificacion edad resultado
1    Juan           MBS   13        15
2    <NA>           MBS   13        11
3   Maria           MBS   10        15
> alumnos %>% filter_if(is.numeric,all_vars(.>12))
  nombres calificacion edad resultado
1    Juan           MBS   13        15
> alumnos %>% filter_if(is.numeric,all_vars(.<12))
[1] nombres      calificacion edad         resultado
<0 rows> (or 0-length row.names)
```

- Idem para la función summarize
  - summarise_all(): Afecta a todas las variables
  - summarise_at(): Afecta a todas las variables seleccionadas por medio de un vector chr o vars()
  - summarise_if(): afecta las variables seleccionadas con una función condicional.