Curso básico R studio - Clase 5

Stephanie Hereira Pacheco

Contents

tidyverse

Hasta ahora hemos estado manipulando las tablas o *dataframes* creando subconjuntos mediante la indexación y utlizando otras funciones del Rbase. Sin embargo, existe todo un universo llamado *tidyverse* que nos perimte hacer todo esto que vimos y más de manera más intuitiva.

Podemos cargar todos los paquetes del *tidyverse* a la vez al instalar y cargar el paquete **tidyverse**:

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
```

```
## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.4     v dplyr   1.0.7
## v tidyr   1.1.3     v stringr 1.4.0
## v readr   2.0.1     v forcats 0.5.1
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

Los paquetes que se activan son dplyr, purr, tidyr, stringr, tibble para el manejo de tablas; readr para importar y exportar datos y forcats para manejo de factores. Además de ggplot2 que es el paquete para graficar.

Datos *tidy*

Hemos estado trabajando con tablas o *dataframes*, sin embargo, el *tidyverse* presenta un nuevo tipo de forma de almacenamiento de datos. Decimos que una tabla de datos está en formato *tidy* si cada fila representa una observación y las columnas representan las diferentes variables disponibles para cada una de estas observaciones. El set de datos `us_rent_income` o como lo he denominado *rentas_us* es un ejemplo de un *data frame tidy*.

estado	variable	estimado
Alabama	ingreso	24476
Alabama	renta	747
Alaska	ingreso	32940
Alaska	renta	1200
Arizona	ingreso	27517
Arizona	renta	972
Arkansas	ingreso	23789
Arkansas	renta	709
California	ingreso	29454
California	renta	1358

Cada fila representa una medida con cada una de las otras dos columnas proveyendo una variable diferente relacionada con las otras dos: variable (si es el ingreso o la renta) y el estado.

Ahora bien, también podemos ver la misma información pero organizada de otra forma:

estado	ingreso	renta
Alabama	24476	747
Alaska	32940	1200
Arizona	27517	972
Arkansas	23789	709
California	29454	1358

Se provee la misma información, pero hay dos diferencias importantes en el formato: 1) cada fila incluye varias observaciones y 2) una de las variables, "variable", se almacena en el encabezado.
Para que los paquetes del *tidyverse* se utilicen de manera óptima, le tenemos que cambiar la forma a los datos para que estén en formato *tidy*, como las primera tabla. Esto podemos hacerlo fuera de R o también R tiene funciones para hacerlo, que veremos más adelante.

Tibbles

Los datos o tablas resultantes luego de aplicar funciones del *tidyverse* se conocen como **tibbles** y son prácticamente igual que los *dataframes* pero con unas ligeras diferencias. Veamos:

```
df<- data.frame(letras=c("a", "b", "c"),
                números=1:3)
tib<- as_tibble(df)
```

```
class(df)
```

```
## [1] "data.frame"
```

```
class(tib)
```

```
## [1] "tbl_df"     "tbl"        "data.frame"
```

```
print(df)
```

```
##   letras números
## 1      a       1
## 2      b       2
## 3      c       3
```

```
print(tib)
```

```
## # A tibble: 3 x 2
##   letras números
##   <chr>    <int>
## 1 a            1
## 2 b            2
## 3 c            3
```

Se ven ligeramente diferentes, como que la tibble te muestra información como tipo de dato de columna y las dimensiones, además de omitir los rownames por defecto. Algunas funciones pueden dar error si no es de un tipo u otro, por ejemplo si en vez de ser un *dataframe* es un *tibble* pero en escencia son lo mismo y se manejan igual. De aquí en adelante nos dirijiremos indistintamente sobre las dos, aunque ya sabemos la diferencia entre una y otra.

Manipulación de *data frames*

El paquete **dplyr** del *tidyverse* ofrece funciones que realizan algunas de las operaciones más comunes y que ya vimos el capítulo anterior con R base. Las funciones principales de *dplyr* son: *select*, *mutate*, *filter* y *summarise*. Pero antes, revisemos lo que es el *pipe*.

El *pipe*: %>%

El *pipe* es la herramienta que nos permite darle *dplyr* las órdenes, comandos o funciones a realizar. Lo podemos poner con el atajo del teclado **"Ctrl + Shift + M (Windows/linux)"** y **"Cmd + Shift + M (Mac)"**.
Con **dplyr**, podemos realizar una serie de operaciones, por escoger una columna, crear una nueva, filtrar nuestras filas y demás . En Rbase tendríamos que hacer paso por paso, por ejemplo :

```
#asignando la data a la variable "mi_data"
mi_data<-ToothGrowth
head(mi_data)
```

```
##    len supp dose
## 1  4.2   VC  0.5
## 2 11.5   VC  0.5
## 3  7.3   VC  0.5
## 4  5.8   VC  0.5
## 5  6.4   VC  0.5
## 6 10.0   VC  0.5
```

```r
#escoger sólo las columnas len y dose
mi_data<- mi_data[c("len", "dose")]
head(mi_data)
```

```
##    len dose
## 1  4.2  0.5
## 2 11.5  0.5
## 3  7.3  0.5
## 4  5.8  0.5
## 5  6.4  0.5
## 6 10.0  0.5
```

```r
#hacer una nueva columna declarando la variable "dose" como un factor
mi_data$dose<- factor(mi_data$dose, levels = c(0.5,1.0, 2.0), labels =c("D0.5", "D1", "2"))
head(mi_data)
```

```
##    len dose
## 1  4.2 D0.5
## 2 11.5 D0.5
## 3  7.3 D0.5
## 4  5.8 D0.5
## 5  6.4 D0.5
## 6 10.0 D0.5
```

```r
#filtrar solo los que sean de dosis = 1
mi_data<- mi_data[mi_data$dose=="D1",]
head(mi_data)
```

```
##     len dose
## 11 16.5   D1
## 12 16.5   D1
## 13 15.2   D1
## 14 17.3   D1
## 15 22.5   D1
## 16 17.3   D1
```

En cambio en dplyr y con funciones que veremos a continuación, todos estos pasos podemos resumirlos a la siguiente línea de código:

```r
mi_data<- ToothGrowth %>% select(len, dose) %>% mutate(
  dose=case_when(dose==0.5~"D0.5",dose==1~"D1", dose==2~"D2")) %>% filter(dose=="D1")
head(mi_data)
```

```
##    len dose
## 1 16.5   D1
## 2 16.5   D1
## 3 15.2   D1
## 4 17.3   D1
## 5 22.5   D1
## 6 17.3   D1
```

Como vimos para realizar la secuencia de estos pasos y unir estas funciones en una sólo línea de código hicimos uso del *pipe* %>%. En general, el *pipe* envía el resultado que se encuentra en el lado izquierdo del *pipe* para ser el primer argumento de la función en el lado derecho del *pipe*. Aquí vemos un ejemplo sencillo:

```
16 %>% sqrt()
```

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

Podemos continuar canalizando (*piping* en inglés) valores a lo largo de:

```
16 %>% sqrt() %>% log2()
```

```
## [1] 2
```

La declaración anterior es equivalente a:

```
log2(sqrt(16))
```

```
## [1] 2
```

Ahora veremos todas las funciones que nos permitirán manipular nuestras tablas en el mismo orden que las vimos en Rbase (el capítulo pasado).

1. Seleccionando columnas

dplyr tiena una función muy intuitiva para seleccionar las columnas que queremos en un *dataframe* y es **select()**

```
head(select(.data = ToothGrowth, len, dose))
```

```
##    len dose
## 1  4.2  0.5
## 2 11.5  0.5
## 3  7.3  0.5
## 4  5.8  0.5
## 5  6.4  0.5
## 6 10.0  0.5
```

Usandola con el *pipe* sería algo así:

```
#seleccionando columnas que queremos
ToothGrowth %>% select(len, dose) %>% head()
```

```
##    len dose
## 1  4.2  0.5
## 2 11.5  0.5
## 3  7.3  0.5
## 4  5.8  0.5
## 5  6.4  0.5
## 6 10.0  0.5
```

```
ToothGrowth %>% select(len:dose) %>% glimpse()
```

```
## Rows: 60
## Columns: 3
## $ len  <dbl> 4.2, 11.5, 7.3, 5.8, 6.4, 10.0, 11.2, 11.2, 5.2, 7.0, 16.5, 16.5,~
## $ supp <fct> VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, V~
## $ dose <dbl> 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 1.0, 1.0, 1.0, ~
```

```
#Seleccionando columnas que no queremos
ToothGrowth %>% select(-len) %>% glimpse()
```

```
## Rows: 60
## Columns: 2
## $ supp <fct> VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, V~
## $ dose <dbl> 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 1.0, 1.0, 1.0, ~
```

La función **glimpse** nos permite ver los datos como la función *str* pero uniendolo al pipe.

También podemos seleccionar columnas con criterios, por ejemplo:

```
ToothGrowth %>% select(starts_with("d")) %>% glimpse()
```

```
## Rows: 60
## Columns: 1
## $ dose <dbl> 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 1.0, 1.0, 1.0, ~
```

```
ToothGrowth %>% select(contains("ose")) %>% glimpse()
```

```
## Rows: 60
## Columns: 1
## $ dose <dbl> 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 1.0, 1.0, 1.0, ~
```

```
ToothGrowth %>% select(ends_with("ose")) %>% glimpse()
```

```
## Rows: 60
## Columns: 1
## $ dose <dbl> 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 1.0, 1.0, 1.0, ~
```

```
ToothGrowth %>% select(matches("o.+e")) %>% glimpse()
```

```
## Rows: 60
## Columns: 1
## $ dose <dbl> 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 1.0, 1.0, 1.0, ~
```

Seleccionando usando un vector, notemos también que en el orden que ponemos el vector así va a apareciendo reordenando las columnas en nuestra tabla:

```
colum <- c("supp", "len")
ToothGrowth  %>%  select(!!colum) %>% glimpse()
```

```
## Rows: 60
## Columns: 2
## $ supp <fct> VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, V~
## $ len  <dbl> 4.2, 11.5, 7.3, 5.8, 6.4, 10.0, 11.2, 11.2, 5.2, 7.0, 16.5, 16.5,~
```

Cada función del *tidyverse* tiene tres variantes que son *at*, *if* y *all* que al combinarlos con nuestras funciones principales como *select* nos permiten hacer muchas cosas más.
Por ejemplo, si usamos *if* sería bajo un criterio como un tipo de dato :

```
#selección positiva
ToothGrowth %>% select_if(is.numeric) %>% glimpse()
```

```
## Rows: 60
## Columns: 2
## $ len  <dbl> 4.2, 11.5, 7.3, 5.8, 6.4, 10.0, 11.2, 11.2, 5.2, 7.0, 16.5, 16.5,~
## $ dose <dbl> 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 1.0, 1.0, 1.0, ~
```

```
#selección negativa
ToothGrowth %>% select_if(~!is.numeric(.)) %>% glimpse()
```

```
## Rows: 60
## Columns: 1
## $ supp <fct> VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, V~
```

Con *all* podemos reformatear los nombres de nuestras columnas:

```
ToothGrowth %>% select_all(toupper) %>% glimpse()
```

```
## Rows: 60
## Columns: 3
## $ LEN  <dbl> 4.2, 11.5, 7.3, 5.8, 6.4, 10.0, 11.2, 11.2, 5.2, 7.0, 16.5, 16.5,~
## $ SUPP <fct> VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, V~
## $ DOSE <dbl> 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 1.0, 1.0, 1.0, ~
```

Y con *at* también podemos escoger columnas basadas en criterios, por ejemplo:

```
#npositiva
ToothGrowth%>% select_at(vars(contains("ose"))) %>% glimpse()
```

```
## Rows: 60
## Columns: 1
## $ dose <dbl> 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 1.0, 1.0, 1.0, ~
```

```
#negativa
ToothGrowth%>% select_at(vars(-contains("ose"))) %>% glimpse()
```

```
## Rows: 60
## Columns: 2
## $ len  <dbl> 4.2, 11.5, 7.3, 5.8, 6.4, 10.0, 11.2, 11.2, 5.2, 7.0, 16.5, 16.5,~
## $ supp <fct> VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, V~
```

```
ToothGrowth%>% select_at(vars(!contains("ose"))) %>% glimpse()
```

```
## Rows: 60
## Columns: 2
## $ len  <dbl> 4.2, 11.5, 7.3, 5.8, 6.4, 10.0, 11.2, 11.2, 5.2, 7.0, 16.5, 16.5,~
## $ supp <fct> VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, V~
```

2. Filtrando filas

Para filtrar nuestra data a nivel de filas usamos la función *filter*:

```
head(filter(ToothGrowth, supp=="OJ") )
```

```
##    len supp dose
## 1 15.2   OJ  0.5
## 2 21.5   OJ  0.5
## 3 17.6   OJ  0.5
## 4  9.7   OJ  0.5
## 5 14.5   OJ  0.5
## 6 10.0   OJ  0.5
```

o con el pipe:

```
ToothGrowth %>% filter(supp=="OJ") %>% glimpse()
```

```
## Rows: 30
## Columns: 3
## $ len  <dbl> 15.2, 21.5, 17.6, 9.7, 14.5, 10.0, 8.2, 9.4, 16.5, 9.7, 19.7, 23.~
## $ supp <fct> OJ, OJ, OJ, OJ, OJ, OJ, OJ, OJ, OJ, OJ, OJ, OJ, OJ, OJ, OJ, OJ, O~
## $ dose <dbl> 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 1.0, 1.0, 1.0, ~
```

```
ToothGrowth %>% filter(len>30) %>% glimpse()
```

```
## Rows: 3
## Columns: 3
## $ len  <dbl> 33.9, 32.5, 30.9
## $ supp <fct> VC, VC, OJ
## $ dose <dbl> 2, 2, 2
```

```
ToothGrowth %>% filter(supp=="OJ", len>30) %>% glimpse()
```

```
## Rows: 1
## Columns: 3
## $ len  <dbl> 30.9
## $ supp <fct> OJ
## $ dose <dbl> 2
```

Filtrando basado en un vector:

```
len_quiero<-c(26.4, 27.3 ,29.4, 23.0)
ToothGrowth %>% filter(len %in% len_quiero) %>% glimpse()
```

```
## Rows: 8
## Columns: 3
## $ len  <dbl> 26.4, 26.4, 27.3, 26.4, 26.4, 27.3, 29.4, 23.0
## $ supp <fct> VC, OJ, OJ, OJ, OJ, OJ, OJ, OJ
## $ dose <dbl> 2, 1, 1, 2, 2, 2, 2, 2
```

```
ToothGrowth %>% filter(!len %in% len_quiero) %>% glimpse()
```

```
## Rows: 52
## Columns: 3
## $ len  <dbl> 4.2, 11.5, 7.3, 5.8, 6.4, 10.0, 11.2, 11.2, 5.2, 7.0, 16.5, 16.5,~
## $ supp <fct> VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, V~
## $ dose <dbl> 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 1.0, 1.0, 1.0, ~
```

Filtrando NA's (en caso de que este dataset tuviera NA's):

```
ToothGrowth %>% filter(!is.na(len))
```

En el caso de *if* , *all* y *at* , nos permiten filtrar con condiciones y a través de varias columnas:

```
ToothGrowth %>% filter_if(is.numeric, all_vars(between(.,2,20)))
```

```
##    len supp dose
## 1 18.5   VC    2
```

```
ToothGrowth %>% filter_all(any_vars(. > 30))
```

```
## Warning in Ops.factor(supp, 30): '>' not meaningful for factors
```

```
##    len supp dose
## 1 33.9   VC    2
## 2 32.5   VC    2
## 3 30.9   OJ    2
```

```
ToothGrowth %>% filter_at(vars(len, dose), all_vars(.>1)) %>% head()
```

```
##    len supp dose
## 1 23.6   VC    2
## 2 18.5   VC    2
## 3 33.9   VC    2
## 4 25.5   VC    2
## 5 26.4   VC    2
## 6 32.5   VC    2
```

3. Creando una nueva columna

Para crear una nueva columna en *dplyr* usamos la función *mutate()*:

```
ToothGrowth %>% mutate(lenlog = log(len)) %>% head()
```

```
##    len supp dose   lenlog
## 1  4.2   VC  0.5 1.435085
## 2 11.5   VC  0.5 2.442347
## 3  7.3   VC  0.5 1.987874
## 4  5.8   VC  0.5 1.757858
## 5  6.4   VC  0.5 1.856298
## 6 10.0   VC  0.5 2.302585
```

Al igual que las funciones pasadas también podemos utilizar *if*, *all* y *at*:

```
ToothGrowth %>% mutate_if(is.numeric, round) %>% head()
```

```
##   len supp dose
## 1   4   VC    0
## 2  12   VC    0
## 3   7   VC    0
## 4   6   VC    0
## 5   6   VC    0
## 6  10   VC    0
```

```
ToothGrowth %>% mutate_all(tolower) %>% head()
```

```
##    len supp dose
## 1  4.2   vc  0.5
## 2 11.5   vc  0.5
## 3  7.3   vc  0.5
## 4  5.8   vc  0.5
## 5  6.4   vc  0.5
## 6   10   vc  0.5
```

La acción de mutar o la función que se pone después del argumento (como round y tolower), muchas veces se pone sin paréntesis pero otras las requiere. Vemos también que usando el *mutate_all* cambia todas las

columnas (lo que quiere decir que las numéricas las convierte en character). Ahora si quisieramos usarla al revés:

```
ToothGrowth %>% mutate_all(round) %>% head()
```

Error: Problem with `mutate()` column `supp`

En estos casos es mejor usar if, como vimos en el ejemplo anterior, *all* aplica mejor si tenemos una data con el mismo tipo de datos (números, caracterres, factores). Ahora bien, *at* nos permite hacer cambios a columnas específicas:

```
ToothGrowth %>%  mutate_at(vars(contains("ose")), ~(.*100)) %>% head()
```

```
##    len supp dose
## 1  4.2   VC   50
## 2 11.5   VC   50
## 3  7.3   VC   50
## 4  5.8   VC   50
## 5  6.4   VC   50
## 6 10.0   VC   50
```

```
ToothGrowth %>%  mutate_at("dose", ~(.*100)) %>% head()
```

```
##    len supp dose
## 1  4.2   VC   50
## 2 11.5   VC   50
## 3  7.3   VC   50
## 4  5.8   VC   50
## 5  6.4   VC   50
## 6 10.0   VC   50
```

Datos discretos

Existen varias herramientas que nos sirven para trabajar con datos discretos, por ejemplo si queremos cambiar los datos de una columna y modificarlos:

```
ToothGrowth %>%  mutate(supp2 = recode_factor(supp,
                        "OJ" = "Jugo",
                        "VC" = "Ascórbico",
                        .default = "other",
                        .ordered = TRUE)) %>% tail()
```

```
##     len supp dose supp2
## 55 24.8   OJ    2  Jugo
## 56 30.9   OJ    2  Jugo
## 57 26.4   OJ    2  Jugo
## 58 27.3   OJ    2  Jugo
## 59 29.4   OJ    2  Jugo
## 60 23.0   OJ    2  Jugo
```

Otra cosa que podemos hacer es crear una nueva columna con valores discretos usando valores numéricos, por ejemplo:

```
ToothGrowth%>%  mutate(dose2 = ifelse(dose > 1, "alto", "bajo")) %>% head()
```

```
##    len supp dose dose2
## 1  4.2   VC  0.5  bajo
## 2 11.5   VC  0.5  bajo
## 3  7.3   VC  0.5  bajo
## 4  5.8   VC  0.5  bajo
## 5  6.4   VC  0.5  bajo
## 6 10.0   VC  0.5  bajo
```

Y si queremos renombrar los datos en una columna, entonces:

```
ToothGrowth%>%mutate(dose = case_when(
    dose == 0.5 ~ "D_0.5",
    dose == 1 ~ "D_1",
    dose == 2 ~ "D_2")) %>%  mutate(
      dose = factor(dose, levels = c("D_2", "D_1", "D_0.5"))) %>% head()
```

```
##    len supp  dose
## 1  4.2   VC D_0.5
## 2 11.5   VC D_0.5
## 3  7.3   VC D_0.5
## 4  5.8   VC D_0.5
## 5  6.4   VC D_0.5
## 6 10.0   VC D_0.5
```

Para separar o unir datos de una columna con data discreta (caracter), podemos usar las funciones *unite()* y *separate()*, por ejemplo:

```
ToothGrowth %>% unite("interaccion", supp:dose, sep = "_") %>% head()
```

```
##    len interaccion
## 1  4.2      VC_0.5
## 2 11.5      VC_0.5
## 3  7.3      VC_0.5
## 4  5.8      VC_0.5
## 5  6.4      VC_0.5
## 6 10.0      VC_0.5
```

Para ejemplifcar *separate* usaremos el ejemplo anterior:

```
ToothGrowth%>%mutate(dose = case_when(
    dose == 0.5 ~ "D_0.5",
    dose == 1 ~ "D_1",
    dose == 2 ~ "D_2")) %>% separate(dose, c("D", "dose"),
                                     sep = "_") %>% head()
```

```
##    len supp D dose
## 1  4.2   VC D  0.5
## 2 11.5   VC D  0.5
## 3  7.3   VC D  0.5
## 4  5.8   VC D  0.5
## 5  6.4   VC D  0.5
```

```
## 6 10.0   VC D  0.5
```

Resumiendo los datos

Hay varias funciones en *tidyverse* que nos permiten hacer un resumen de nuestros datos, como por ejemplo la función *count()*:

```
ToothGrowth %>% count(supp, sort=TRUE)
```

```
##   supp  n
## 1   OJ 30
## 2   VC 30
```

```
ToothGrowth %>% count(dose, supp, sort=TRUE)
```

```
##   dose supp  n
## 1  0.5   OJ 10
## 2  0.5   VC 10
## 3  1.0   OJ 10
## 4  1.0   VC 10
## 5  2.0   OJ 10
## 6  2.0   VC 10
```

Otra forma es usar *group_by()* que nos permite agrupar nuestros datos bajo alguna condición y luego aplicar una función a estos:

```
ToothGrowth %>% group_by(supp) %>%count()
```

```
## # A tibble: 2 x 2
## # Groups:   supp [2]
##   supp      n
##   <fct> <int>
## 1 OJ       30
## 2 VC       30
```

Con este *group_by()* podemos aplicar cualquier cantidad de funciones, por ejemplo para en vez que me de el conteo me de el promedio, mediana, cuenta, suma, etc; para esto, debemos ocupar una nueva función muy útil llamada *summarise()*:

```
ToothGrowth %>% group_by(supp) %>% summarise(promedio = mean(len), suma = sum(len),
                                             n = n(), mediana =median(len))
```

```
## # A tibble: 2 x 5
##   supp  promedio  suma     n mediana
##   <fct>    <dbl> <dbl> <int>   <dbl>
## 1 OJ        20.7  620.    30    22.7
## 2 VC        17.0  509.    30    16.5
```

Esta función también viene en todas las presentaciones, es decir, *at*, *if* y *all*, Ejemplos:

```
ToothGrowth %>%  group_by(supp) %>% summarise_if(is.numeric, mean)
```

```
## # A tibble: 2 x 3
##   supp    len  dose
##   <fct> <dbl> <dbl>
## 1 OJ     20.7  1.17
## 2 VC     17.0  1.17
```

```
ToothGrowth%>%  group_by(supp) %>% summarise_at(vars(contains("ose")), mean)
```

```
## # A tibble: 2 x 2
##   supp   dose
##   <fct> <dbl>
## 1 OJ     1.17
## 2 VC     1.17
```

```
ToothGrowth %>%  group_by(supp) %>%  summarise_all(mean)
```

```
## # A tibble: 2 x 3
##   supp    len  dose
##   <fct> <dbl> <dbl>
## 1 OJ     20.7  1.17
## 2 VC     17.0  1.17
```

Renombrando columnas Con *tidyverse*

Existen diferentes formas de renombrar las columnas una es con la función *select()*:

```
ToothGrowth %>% select(dosis=dose, longitud=len, suplemento=supp) %>% glimpse()
```

```
## Rows: 60
## Columns: 3
## $ dosis      <dbl> 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 1.0, 1.0,~
## $ longitud   <dbl> 4.2, 11.5, 7.3, 5.8, 6.4, 10.0, 11.2, 11.2, 5.2, 7.0, 16.5,~
## $ suplemento <fct> VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC,~
```

También existe la función rename en todas sus versiones, ejemplos:

```
ToothGrowth %>% rename(dosis=dose) %>% glimpse()
```

```
## Rows: 60
## Columns: 3
## $ len   <dbl> 4.2, 11.5, 7.3, 5.8, 6.4, 10.0, 11.2, 11.2, 5.2, 7.0, 16.5, 16.5~
## $ supp  <fct> VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, ~
## $ dosis <dbl> 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 1.0, 1.0, 1.0,~
```

```
ToothGrowth %>% rename_if(is.numeric, ~paste0("Num_", .)) %>% glimpse()
```

```
## Rows: 60
## Columns: 3
## $ Num_len  <dbl> 4.2, 11.5, 7.3, 5.8, 6.4, 10.0, 11.2, 11.2, 5.2, 7.0, 16.5, 1~
## $ supp     <fct> VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, V~
## $ Num_dose <dbl> 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 1.0, 1.0, 1~
```

```
ToothGrowth%>% rename_at(vars(contains("ose")), ~paste0("Num_", .)) %>% glimpse()
```

```
## Rows: 60
## Columns: 3
## $ len      <dbl> 4.2, 11.5, 7.3, 5.8, 6.4, 10.0, 11.2, 11.2, 5.2, 7.0, 16.5, 1~
## $ supp     <fct> VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, V~
## $ Num_dose <dbl> 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 1.0, 1.0, 1~
```

```
ToothGrowth%>% rename_all(toupper) %>% glimpse()
```

```
## Rows: 60
## Columns: 3
## $ LEN  <dbl> 4.2, 11.5, 7.3, 5.8, 6.4, 10.0, 11.2, 11.2, 5.2, 7.0, 16.5, 16.5,~
## $ SUPP <fct> VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, VC, V~
## $ DOSE <dbl> 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 1.0, 1.0, 1.0, ~
```

Ordenando tablas por un criterio o columna

Conocemos las funciones `order` y `sort`, pero para ordenar tablas enteras, la función `arrange` de **dplyr** es útil. Por ejemplo:

```
ToothGrowth %>% arrange(len) %>% head()
```

```
##   len supp dose
## 1 4.2   VC  0.5
## 2 5.2   VC  0.5
## 3 5.8   VC  0.5
## 4 6.4   VC  0.5
## 5 7.0   VC  0.5
## 6 7.3   VC  0.5
```

```
ToothGrowth %>% arrange(-len) %>% head()
```

```
##    len supp dose
## 1 33.9   VC    2
## 2 32.5   VC    2
## 3 30.9   OJ    2
## 4 29.5   VC    2
## 5 29.4   OJ    2
## 6 27.3   OJ    1
```

También podemos ordenar por varios criterios:

```
ToothGrowth%>%  arrange(dose, supp) %>%  head()
```

```
##    len supp dose
## 1 15.2   OJ  0.5
## 2 21.5   OJ  0.5
## 3 17.6   OJ  0.5
## 4  9.7   OJ  0.5
## 5 14.5   OJ  0.5
## 6 10.0   OJ  0.5
```

Uniendo tablas

Para la unión de tablas usaremos una familia de funciones denominadas **joins**.

Las diferentes presentaciones de esta función nos permite juntar tablas:
- inner_join() : incluye todas las filas en x y y (es decir la inteserccion o las que comparten).

- left_join(): incluye todas las filas en x.
- right_join(): incuye todas las filas en y.
- full_join(): incluye todas las filas en x o y (este incluye todos, incluso las que no comparten)

```
band_members; band_instruments
```

```
## # A tibble: 3 x 2
##   name  band
##   <chr> <chr>
## 1 Mick  Stones
## 2 John  Beatles
## 3 Paul  Beatles
```

```
## # A tibble: 3 x 2
##   name  plays
##   <chr> <chr>
## 1 John  guitar
## 2 Paul  bass
## 3 Keith guitar
```

```
band_members %>% full_join(band_instruments)
```

```
## Joining, by = "name"
```

```
## # A tibble: 4 x 3
##   name  band    plays
##   <chr> <chr>   <chr>
## 1 Mick  Stones  <NA>
## 2 John  Beatles guitar
## 3 Paul  Beatles bass
## 4 Keith <NA>    guitar
```

```
band_members %>% inner_join(band_instruments, by = "name")
```

```
## # A tibble: 2 x 3
##   name  band    plays
##   <chr> <chr>   <chr>
## 1 John  Beatles guitar
## 2 Paul  Beatles bass
```

```
band_members %>% left_join(band_instruments)
```

```
## Joining, by = "name"
```

```
## # A tibble: 3 x 3
##   name  band    plays
##   <chr> <chr>   <chr>
## 1 Mick  Stones  <NA>
## 2 John  Beatles guitar
## 3 Paul  Beatles bass
```

```
band_members %>% right_join(band_instruments)
```

```
## Joining, by = "name"
```

```
## # A tibble: 3 x 3
##   name  band    plays
##   <chr> <chr>   <chr>
## 1 John  Beatles guitar
## 2 Paul  Beatles bass
## 3 Keith <NA>    guitar
```

Funciones adicionales del *tydiverse*

Anteriormente, usamos la función `head` para evitar que la página se llene con todo el set de datos. Si queremos ver una mayor proporción, podemos usar la función `top_n`. Esta función toma un *data frame* como primer argumento, el número de filas para mostrar en el segundo y la variable para filtrar en el tercero. Aquí hay un ejemplo de cómo ver las 5 filas superiores:

```
ToothGrowth %>% top_n(5, len)
```

```
##    len supp dose
## 1 33.9   VC    2
## 2 32.5   VC    2
## 3 29.5   VC    2
## 4 30.9   OJ    2
## 5 29.4   OJ    2
```

También hay otras funciones que son útiles en los diferentes análisis donde a veces ocupamos los rownames o a veces no, estas funciones nos permiten hacer una columna que sea rownames y viceversa, por ejemplo:

```
ToothGrowth %>% rownames_to_column(var = "row") %>% head()
```

```
##   row  len supp dose
## 1   1  4.2   VC  0.5
## 2   2 11.5   VC  0.5
## 3   3  7.3   VC  0.5
## 4   4  5.8   VC  0.5
## 5   5  6.4   VC  0.5
## 6   6 10.0   VC  0.5
```

```
ToothGrowth %>% rownames_to_column(
  var = "row") %>% column_to_rownames(var = "row") %>% head()
```

```
##    len supp dose
## 1  4.2   VC  0.5
## 2 11.5   VC  0.5
## 3  7.3   VC  0.5
## 4  5.8   VC  0.5
## 5  6.4   VC  0.5
## 6 10.0   VC  0.5
```

Si queremos convertir las tablas como al principio del capítulo, es decir, convertir una tabla que no está en formato *tidy* a una que sí esté y viceversa podemos usar las funciones **pivot_longer()** y **pivot_wider()** (antes denominadas **gather()** y **spread()**, pero ya están reemplazadas aunque aún no eliminadas).

```
data(iris)
data_iris<- iris %>% select(Sepal.Length, Sepal.Width) %>% rownames_to_column(var = "ids")
head(data_iris)
```

```
##   ids Sepal.Length Sepal.Width
## 1   1          5.1         3.5
## 2   2          4.9         3.0
## 3   3          4.7         3.2
## 4   4          4.6         3.1
## 5   5          5.0         3.6
## 6   6          5.4         3.9
```

Vamos a cambiarla a formato tidy:

```
tidy_iris<-pivot_longer(names_to = "variable",
             values_to = "longitud",
             data = data_iris,
             cols = Sepal.Length:Sepal.Width)
head(tidy_iris)
```

```
## # A tibble: 6 x 3
##   ids   variable     longitud
##   <chr> <chr>           <dbl>
## 1 1     Sepal.Length      5.1
## 2 1     Sepal.Width       3.5
## 3 2     Sepal.Length      4.9
## 4 2     Sepal.Width       3
## 5 3     Sepal.Length      4.7
## 6 3     Sepal.Width       3.2
```

Y si queremos regresar a como lo teníamos:

```
notidy_iris<- tidy_iris  %>% pivot_wider(names_from = variable,
                          values_from = longitud)
head(notidy_iris)
```

```
## # A tibble: 6 x 3
##   ids   Sepal.Length Sepal.Width
##   <chr>        <dbl>       <dbl>
## 1 1              5.1         3.5
## 2 2              4.9         3
## 3 3              4.7         3.2
## 4 4              4.6         3.1
## 5 5              5           3.6
## 6 6              5.4         3.9
```

Ejemplo Aplicado

```
primera<- data.frame(Nombre= c("Astrid", "Lea", "Sarina", "Remon", "Letizia",
                               "Babice", "Jonas", "Wendy", "Nivedithia", "Gioia"),
                     Sexo= c("F", "F", "F", "M", "F", "F", "M", "F", "F", "F"),
                     Edad= c(30,25,25,29,22,22,35,19,32,21))
```

```
segunda<- data.frame(Nombre= c("Astrid", "Lea", "Sarina", "Remon", "Letizia",
                               "Babice", "Jonas", "Wendy", "Nivedithia", "Gioia"),
                     Superhéroe= c("Batman", "Superman", "Batman", "Spiderman", "Batman",
                                   "Antman", "Batman", "Superman", "Maggot", "Superman"),
                     Tatuajes= c(11,15,12,5,65,3,9,13,900,0))
knitr::kable(primera); knitr::kable(segunda)
```

Nombre	Sexo	Edad
Astrid	F	30
Lea	F	25
Sarina	F	25
Remon	M	29
Letizia	F	22
Babice	F	22
Jonas	M	35
Wendy	F	19
Nivedithia	F	32
Gioia	F	21

Nombre	Superhéroe	Tatuajes
Astrid	Batman	11
Lea	Superman	15
Sarina	Batman	12
Remon	Spiderman	5
Letizia	Batman	65
Babice	Antman	3
Jonas	Batman	9
Wendy	Superman	13
Nivedithia	Maggot	900
Gioia	Superman	0

Para hacer:

1. Combina las dos tablas en una sola y completa las siguientes asignaciones.
2. ¿Cuál es la edad media de las mujeres y hombres por separado?
3. ¿Cuál fue el número más alto de tatuajes en un hombre?
4. ¿Cuál es el porcentaje de personas debajo de 32 años que son mujeres?
5. Agrega una nueva columna a a la data llamada `tatuajes.por.año` que muestre cuántos tatuajes por año se ha hecho cada persona por cada año en su vida.
6. ¿Cuál persona tiene el mayor número de tatuajes por año?
7. ¿Cuáles son los nombres de las mujeres a las que su superheroe favorito es superman?
8. ¿Cuál es la mediana del número de tatuajes de cad apersona que está por encima de los 20 años y que su personaje favorito es Batman?

Resolviendo

1. Combina las dos tablas en una sola y completa las siguientes asignaciones.

```
encuestas<-primera %>% full_join(segunda)
```

```
## Joining, by = "Nombre"
```

2. ¿Cuál es la edad media de las mujeres y hombres por separado?

```
encuestas %>% group_by(Sexo) %>% summarise_at(c("Edad"), mean)
```

```
## # A tibble: 2 x 2
##   Sexo   Edad
##   <chr> <dbl>
## 1 F      24.5
## 2 M      32
```

3. ¿Cuál fue el número más alto de tatuajes en un hombre?

```
encuestas %>% filter(Sexo=="M") %>% filter(Tatuajes == max(Tatuajes))
```

```
##   Nombre Sexo Edad Superhéroe Tatuajes
## 1  Jonas    M   35     Batman        9
```

4. ¿Cuál es el porcentaje de mujeres debajo de 32 años?

```
fem<- encuestas %>% filter(Sexo=="F")
fem_32<- encuestas %>% filter( Sexo=="F", Edad<32)
(nrow(fem_32)/nrow(fem))*100
```

```
## [1] 87.5
```

5. Agrega una nueva columna a a la data llamada `tatuajes.por.año` que muestre cuántos tatuajes por año se ha hecho cada persona por cada año en su vida.

```
encuestas<- encuestas %>% mutate("tatuajesporaño"=Tatuajes/Edad)
encuestas$tatuajesporaño
```

```
##  [1]  0.3666667  0.6000000  0.4800000  0.1724138  2.9545455  0.1363636
##  [7]  0.2571429  0.6842105 28.1250000  0.0000000
```

6. ¿Cuál persona tiene el mayor número de tatuajes por año?

```
encuestas  %>% filter(tatuajesporaño == max(tatuajesporaño))
```

```
##       Nombre Sexo Edad Superhéroe Tatuajes tatuajesporaño
## 1 Nivedithia    F   32     Maggot      900         28.125
```

7. ¿Cuáles son los nombres de las mujeres a las que su superheroe favorito es superman?

```
encuestas %>% filter(Sexo=="F", Superhéroe=="Superman") %>% select(Nombre)
```

```
##   Nombre
## 1    Lea
## 2  Wendy
## 3  Gioia
```

8. ¿Cuál es la mediana del número de tatuajes de cada persona que está por encima de los 20 años y que su personaje favorito es batman?

```
encuestas %>% filter(Edad>20, Superhéroe =="Batman") %>% summarise(mediana=median(Tatuajes))
```

```
##   mediana
## 1    11.5
```