Taller sobre el lenguaje R

Clase 3: Procesamiento, diagnóstico y estructuración de datos

Lic. Lucio José Pantazis

March 31, 2021

Procesamiento y manipulación de datos

dplyr & tidyr

dplyr y tidyr son dos paquetes especializados en la manipulación y procesamiento de datos, respectivamente.

```
library(dplyr)
library(tidyr)
```

Pipeline: "%>%"

- Ambos paquetes habilitan el uso del pipeline (%>%, atajo: Ctrl+Shift+M).

Pipeline: "%>%"

- Ambos paquetes habilitan el uso del pipeline (%>%, atajo: Ctrl+Shift+M).

- El pipeline pasa por default la variable a su izquierda como primer argumento a la función que está a su derecha.

Pipeline: "%>%"

- Ambos paquetes habilitan el uso del pipeline (%>%, atajo: Ctrl+Shift+M).

- El pipeline pasa por default la variable a su izquierda como primer argumento a la función que está a su derecha.

- Ejemplo:

```
head(mpg$trans)
```

```
## [1] "auto(l5)"   "manual(m5)" "manual(m6)" "auto(av)"   "auto(l5)"
## [6] "manual(m5)"
```

```
mpg$trans %>% head()
```

```
## [1] "auto(l5)"   "manual(m5)" "manual(m6)" "auto(av)"   "auto(l5)"
## [6] "manual(m5)"
```

Pipeline: "%>%"

- Ambos paquetes habilitan el uso del pipeline (%>%, atajo: Ctrl+Shift+M).

- El pipeline pasa por default la variable a su izquierda como primer argumento a la función que está a su derecha.

- Ejemplo:

```
head(mpg$trans)
```

```
## [1] "auto(l5)"   "manual(m5)" "manual(m6)" "auto(av)"   "auto(l5)"
## [6] "manual(m5)"
```

```
mpg$trans %>% head()
```

```
## [1] "auto(l5)"   "manual(m5)" "manual(m6)" "auto(av)"   "auto(l5)"
## [6] "manual(m5)"
```

- Se puede pasar el miembro izquierdo a otro argumento del derecho usando el punto:

```
N=4; N %>% head(mpg$trans,n=.)
```

```
## [1] "auto(l5)"   "manual(m5)" "manual(m6)" "auto(av)"
```

Uso del pipeline

El pipeline se usa para hacer transformaciones sucesivas a los datos de forma secuencial:

```
as.character(length(unique(mpg$trans)))
```

```
## [1] "10"
```

```
mpg$trans %>%
  unique() %>%
  length() %>%
  as.character()
```

```
## [1] "10"
```

stringr

stringr es un paquete que permite trabajar con los strings de una base. Usaremos las siguientes funciones:

stringr

stringr es un paquete que permite trabajar con los strings de una base. Usaremos las siguientes funciones:

- str_detect: Lógica, detecta si un string tiene o no un cierto patrón.

```
require(stringr)
Labs=names(mpg); Labs
```

```
##  [1] "manufacturer" "model"        "displ"        "year"         "cyl"
##  [6] "trans"        "drv"          "cty"          "hwy"          "fl"
## [11] "class"
```

```
str_detect(Labs,pattern = "a")
```

```
##  [1]  TRUE FALSE FALSE  TRUE FALSE  TRUE FALSE FALSE FALSE FALSE  TRUE
```

stringr

stringr es un paquete que permite trabajar con los strings de una base. Usaremos las siguientes funciones:

- str_replace_all: Devuelve un string reemplazando un cierto patrón por otro.

```
str_replace_all(Labs,pattern = "a",replacement = "A")
```

```
##  [1] "mAnufActurer" "model"        "displ"        "yeAr"         "cyl"
##  [6] "trAns"        "drv"          "cty"          "hwy"          "fl"
## [11] "clAss"
```

stringr

Hay que tener cuidado al usar stringr con las llamadas regular expressions (en general, signos de puntuación), ya que son expresiones usadas por R para interpretar acciones sobre los caracteres:

```
str=c("A.B.C","(que onda con esto Lucio?)")
str_detect(str,pattern = ".")
```

```
## [1] TRUE TRUE
```

```
str_replace_all(str,pattern = "(",replacement = "_")
```

```
## Error in stri_replace_all_regex(string, pattern, fix_replacement(replacer
```

stringr

Para evitar estos errores, los signos de puntuación tienen que ser antecedidos por una doble contrabarra:

```
str=c("A.B.C","(que onda con esto Lucio?)")
str_detect(str,pattern = "\\.")
```

```
## [1]  TRUE FALSE
```

```
str_replace_all(str,pattern = "\\(",replacement = "_")
```

```
## [1] "A.B.C"                    "_que onda con esto Lucio?)"
```

tidyr (Separación de variables)

- Notemos que en la base mpg, la variable trans podría ser separada en dos para que una contenga el dato de si la transmisión es automática o no (un factor de sólo dos niveles), y en otra variable que describa las especificaciones de la transmisión.

```
head(mpg)
```

```
## # A tibble: 6 x 11
##   manufacturer model displ  year   cyl trans      drv     cty   hwy fl
##   <chr>        <chr> <dbl> <int> <int> <chr>      <chr> <int> <int> <chr
## 1 audi         a4      1.8  1999     4 auto(l5)   f        18    29 p
## 2 audi         a4      1.8  1999     4 manual(m5) f        21    29 p
## 3 audi         a4      2    2008     4 manual(m6) f        20    31 p
## 4 audi         a4      2    2008     4 auto(av)   f        21    30 p
## 5 audi         a4      2.8  1999     6 auto(l5)   f        16    26 p
## 6 audi         a4      2.8  1999     6 manual(m5) f        18    26 p
```

tidyr (Separación de variables)

Para separar la columna en dos, tidyr provee el comando separate:

tidyr (Separación de variables)

Para separar la columna en dos, tidyr provee el comando separate:

- col es la columna que se quiere separar (En este caso, trans)
- into es el nombre que les dará a las columnas separadas (En este caso, trans y transType)
- sep es el patrón de caracteres que nos marca la separación (En este caso, el paréntesis "(", que debe ser descripto con doble contrabarra)

```
require(tidyr)
mpg %>%
  separate(col=trans,
           into=c("trans","transType"),
           sep="\\(") %>%
  head
```

```
## # A tibble: 6 x 12
##   manufacturer model displ  year   cyl trans transType drv     cty   hwy
##   <chr>        <chr> <dbl> <int> <int> <chr> <chr>     <chr> <int> <int>
## 1 audi         a4      1.8  1999     4 auto  l5)       f        18    29
## 2 audi         a4      1.8  1999     4 manu~ m5)       f        21    29
## 3 audi         a4      2    2008     4 manu~ m6)       f        20    31
## 4 audi         a4      2    2008     4 auto  av)       f        21    30
## 5 audi         a4      2.8  1999     6 auto  l5)       f        16    26
## 6 audi         a4      2.8  1999     6 manu~ m5)       f        18    26
## # ... with 1 more variable: class <chr>
```

tidyr (Separación de variables)

Notar que queda el paréntesis que cierra en transType, que se puede reemplazar por un string vacío con str_replace_all:

```
SepMPG=mpg %>%
  separate(col=trans,
           into=c("trans","transType"),
           sep="\\(") %>%
  head
SepMPG$transType=str_replace_all(SepMPG$transType,
                                 pattern = "\\)",replacement = "")
SepMPG
```

```
## # A tibble: 6 x 12
##   manufacturer model displ  year   cyl trans transType drv     cty   hwy
##   <chr>        <chr> <dbl> <int> <int> <chr> <chr>     <chr> <int> <int>
## 1 audi         a4      1.8  1999     4 auto  l5        f        18    29
## 2 audi         a4      1.8  1999     4 manu~ m5        f        21    29
## 3 audi         a4      2    2008     4 manu~ m6        f        20    31
## 4 audi         a4      2    2008     4 auto  av        f        21    30
## 5 audi         a4      2.8  1999     6 auto  l5        f        16    26
## 6 audi         a4      2.8  1999     6 manu~ m5        f        18    26
## # ... with 1 more variable: class <chr>
```

tidyr (Unificación de variables)

Para hacer el proceso inverso, tidyr ofrece la función unite. Por ejemplo, podríamos unir las columnas que describen al fabricante y al modelo del auto:

tidyr (Unificación de variables)

Para hacer el proceso inverso, tidyr ofrece la función unite. Por ejemplo, podríamos unir las columnas que describen al fabricante y al modelo del auto:

- col es la nueva columna (En este caso, la podemos llamar ManuMod)
- sep es lo que se usará para separar los valores dentro de la columna (En este caso, usaremos un guión bajo)
- Luego se agregan los nombres de las columnas a unir (en este caso, manufacturer y model)

```
mpg %>%
  unite(col="ManuMod",sep="_",manufacturer,model) %>%
  head
```

```
## # A tibble: 6 x 10
##   ManuMod displ  year   cyl trans      drv     cty   hwy fl    class
##   <chr>   <dbl> <int> <int> <chr>      <chr> <int> <int> <chr> <chr>
## 1 audi_a4   1.8  1999     4 auto(l5)   f        18    29 p     compact
## 2 audi_a4   1.8  1999     4 manual(m5) f        21    29 p     compact
## 3 audi_a4   2    2008     4 manual(m6) f        20    31 p     compact
## 4 audi_a4   2    2008     4 auto(av)   f        21    30 p     compact
## 5 audi_a4   2.8  1999     6 auto(l5)   f        16    26 p     compact
## 6 audi_a4   2.8  1999     6 manual(m5) f        18    26 p     compact
```

Formato wide

Cuando se tienen medidas repetidas sobre la misma variable, hay un formato posible que es el formato wide, en el que las mediciones de la variable se ubican en columnas paralelas.

Formato wide

Cuando se tienen medidas repetidas sobre la misma variable, hay un formato posible que es el formato wide, en el que las mediciones de la variable se ubican en columnas paralelas.

Por ejemplo, podemos ver la cantidad de goles a favor de la selección Argentina, la Española y la Mexicana (para no deprimirnos, omitimos la comparación con Alemania) en los últimos 4 mundiales del siguiente modo:

```
MundW=data.frame(Pais=c("Argentina","España","Mexico"),
            G2006=c(11,9,5),G2010=c(10,8,4),
            G2014=c(9,4,5),G2018=c(6,7,3))
MundW
```

```
##        Pais G2006 G2010 G2014 G2018
## 1 Argentina    11    10     9     6
## 2    España     9     8     4     7
## 3    Mexico     5     4     5     3
```

Formato wide

Este formato puede ser útil para realizar comparaciones entre distintos tiempos.

En este caso, se puede ver si aumentó o bajó la cantidad de goles de un equipo entre mundiales, por ejemplo.

```
MundW
```

```
##        Pais G2006 G2010 G2014 G2018
## 1 Argentina    11    10     9     6
## 2    España     9     8     4     7
## 3    Mexico     5     4     5     3
```

```
Dif2014_2006=MundW$G2014-MundW$G2006
names(Dif2014_2006)=MundW$Pais
Dif2014_2006
```

```
## Argentina    España    Mexico
##        -2        -5         0
```

Formato long

Sin embargo, notar que, salvo la primera, todas las columnas describen la misma variable (goles a favor), con otra variable (mundial) que describa a qué mundial pertenece. Entonces, podríamos tener este formato (llamado formato long):

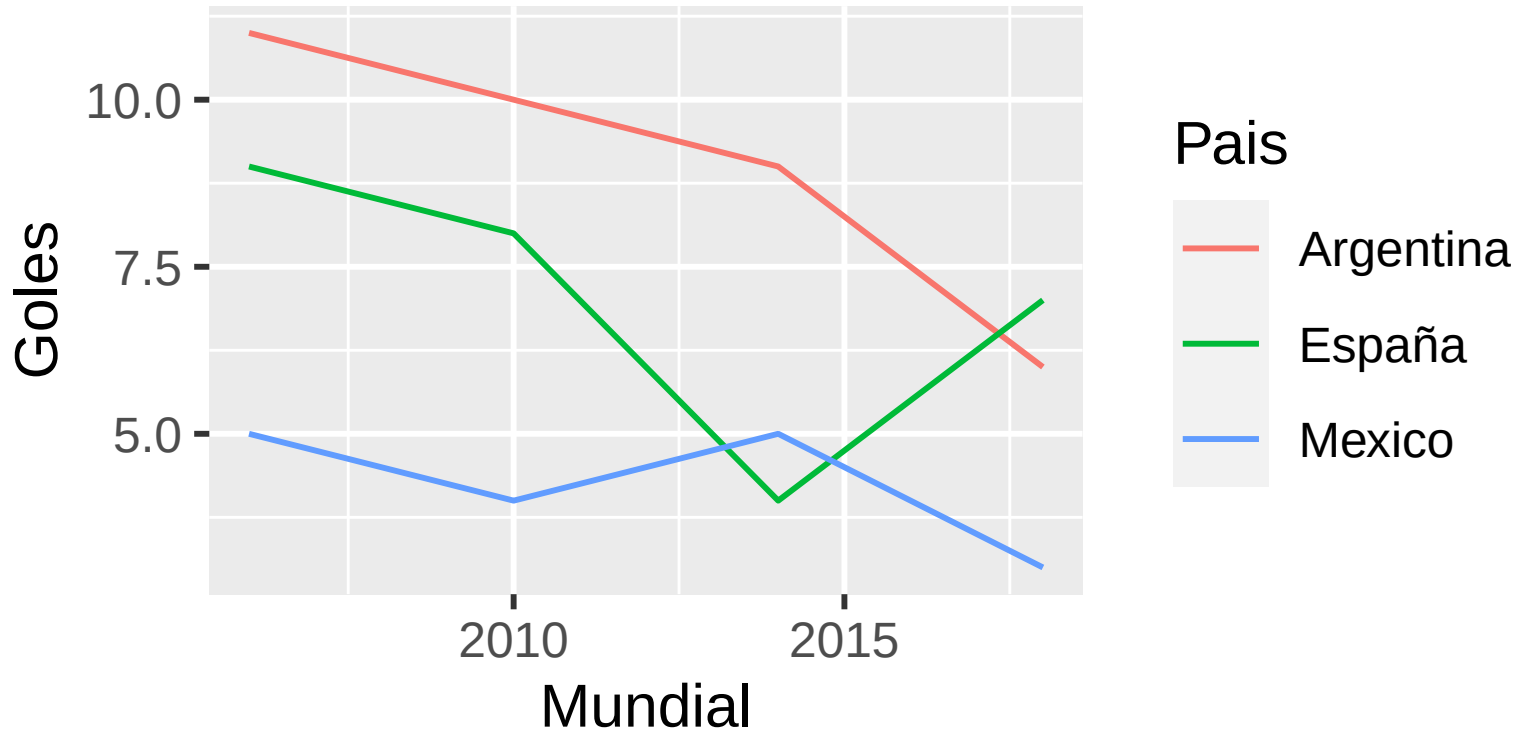
```
MundL=data.frame(Pais=rep(c("Argentina","España","Mexico"),4),
                 Mundial=rep(c(2006,2010,2014,2018),each=3),
                 Goles=c(11,9,5,10,8,4,9,4,5,6,7,3))
MundL
```

```
##          Pais Mundial Goles
## 1   Argentina    2006    11
## 2      España    2006     9
## 3      Mexico    2006     5
## 4   Argentina    2010    10
## 5      España    2010     8
## 6      Mexico    2010     4
## 7   Argentina    2014     9
## 8      España    2014     4
## 9      Mexico    2014     5
## 10  Argentina    2018     6
## 11     España    2018     7
## 12     Mexico    2018     3
```

Formato long

Esta notación es más útil para graficar las evoluciones en el tiempo ya que se puede especificar una única variable (en este caso, goles) para el eje y, manteniendo el orden de las medidas en el eje x.

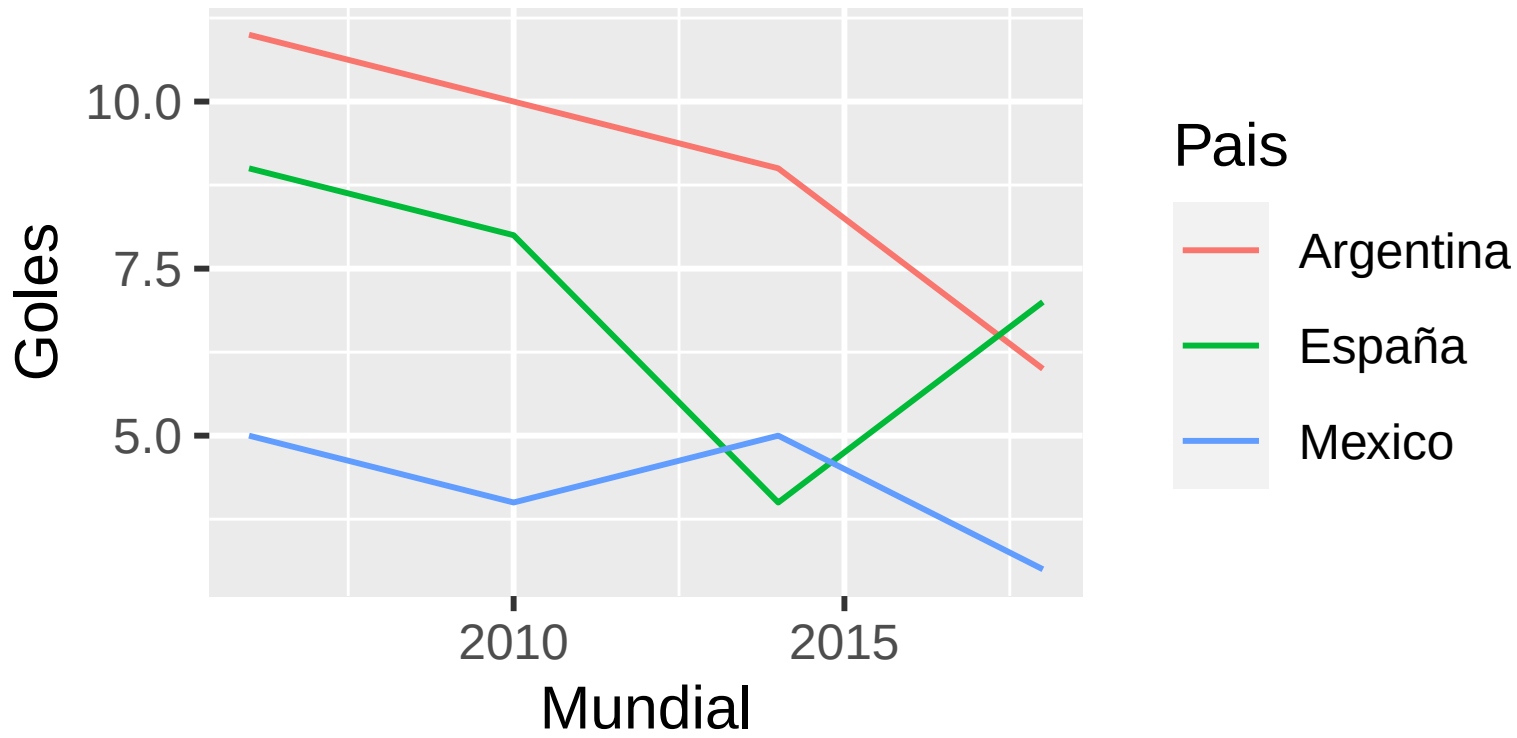
```
MundL %>%
  ggplot(aes(x=Mundial,y=Goles,color=Pais))+
  geom_line(aes(group=Pais))
```



Formato long

Comentario: Para evoluciones a lo largo del tiempo es importante aplicar la variable estética group, que marca a partir de los valores de qué variable se van a agrupar los datos. Miremos lo que pasa si las líneas no se agrupan por país:

```
MundL %>%
  ggplot(aes(x=Mundial,y=Goles,color=Pais))+
  geom_line()
```



tidyr (Pasar de formato long a formato wide)

Para pasar de formato wide a long tidyr ofrece el comando spread:

tidyr (Pasar de formato long a formato wide)

Para pasar de formato wide a long tidyr ofrece el comando spread:

- key: Nombre de la variable que permite identificar las columnas del formato wide (En este caso, Mundial)
- value: Nombre de la variable unificadora (En este caso, Goles)

```
MundL %>%
  spread(key="Mundial",value="Goles")
```

```
##        Pais 2006 2010 2014 2018
## 1 Argentina   11   10    9    6
## 2    España    9    8    4    7
## 3    Mexico    5    4    5    3
```

tidyr (Pasar de formato wide a formato long)

Para pasar de formato wide a long tidyr ofrece el comando gather:

tidyr (Pasar de formato wide a formato long)

Para pasar de formato wide a long tidyr ofrece el comando gather:

- key: Nombre de la variable que permite identificar las columnas del formato wide (En este caso, Mundial)
- value: Nombre de la variable unificadora (En este caso, Goles)
- las columnas a "apilar" en el formato long

```
MundW %>%
  gather(key="Mundial",value="Goles",G2006,G2010,G2014,G2018)
```

```
##          Pais Mundial Goles
## 1   Argentina   G2006    11
## 2      España   G2006     9
## 3      Mexico   G2006     5
## 4   Argentina   G2010    10
## 5      España   G2010     8
## 6      Mexico   G2010     4
## 7   Argentina   G2014     9
## 8      España   G2014     4
## 9      Mexico   G2014     5
## 10  Argentina   G2018     6
## 11     España   G2018     7
## 12     Mexico   G2018     3
```

tidyr (Pasar de formato wide a formato long)

Otra forma de seleccionar las columnas a apilar es aplicando un - a las variables que NO se "apilan", (en este caso, Pais)

```
MundW %>%
  gather(key="Mundial",value="Goles",-Pais)
```

```
##          Pais Mundial Goles
## 1   Argentina   G2006    11
## 2      España   G2006     9
## 3      Mexico   G2006     5
## 4   Argentina   G2010    10
## 5      España   G2010     8
## 6      Mexico   G2010     4
## 7   Argentina   G2014     9
## 8      España   G2014     4
## 9      Mexico   G2014     5
## 10  Argentina   G2018     6
## 11     España   G2018     7
## 12     Mexico   G2018     3
```

dplyr (Seleccionando variables)

El paquete dplyr se encarga más de transformaciones de las variables y de las bases.

dplyr (Seleccionando variables)

El paquete dplyr se encarga más de transformaciones de las variables y de las bases.

Para reducir la cantidad de columnas de la base se usa el comando select. Por ejemplo, podemos seleccionar de la base mpg las columnas manufacturer,hwy, displ, cyl y drv:

```
require(dplyr);head(mpg,n=3)
```

```
## # A tibble: 3 x 11
##   manufacturer model displ  year   cyl trans      drv     cty   hwy fl
##   <chr>        <chr> <dbl> <int> <int> <chr>      <chr> <int> <int> <chr
## 1 audi         a4      1.8  1999     4 auto(l5)   f        18    29 p
## 2 audi         a4      1.8  1999     4 manual(m5) f        21    29 p
## 3 audi         a4      2    2008     4 manual(m6) f        20    31 p
```

```
SelMPG=mpg %>%
  select(manufacturer,hwy,displ,cyl,drv)
SelMPG %>%
  head(n=3)
```

```
## # A tibble: 3 x 5
##   manufacturer   hwy displ   cyl drv
##   <chr>        <int> <dbl> <int> <chr>
## 1 audi            29   1.8     4 f
## 2 audi            29   1.8     4 f
## 3 audi            31   2       4 f
```

dplyr (Seleccionando variables)

Otra forma es aplicando un menos a lo que no se quiere seleccionar:

```
head(mpg,n=3)
```

```
## # A tibble: 3 x 11
##   manufacturer model displ  year   cyl trans      drv     cty   hwy fl
##   <chr>        <chr> <dbl> <int> <int> <chr>      <chr> <int> <int> <chr
## 1 audi         a4      1.8  1999     4 auto(l5)   f        18    29 p
## 2 audi         a4      1.8  1999     4 manual(m5) f        21    29 p
## 3 audi         a4      2    2008     4 manual(m6) f        20    31 p
```

```
mpg %>%
  select(-fl,-manufacturer,-cyl) %>%
  head(n=3)
```

```
## # A tibble: 3 x 8
##   model displ  year trans      drv     cty   hwy class
##   <chr> <dbl> <int> <chr>      <chr> <int> <int> <chr>
## 1 a4      1.8  1999 auto(l5)   f        18    29 compact
## 2 a4      1.8  1999 manual(m5) f        21    29 compact
## 3 a4      2    2008 manual(m6) f        20    31 compact
```

dplyr (Seleccionando filas)

Para elegir filas que cumplan una cierta condición, se usa el comando filter:

```
SelMPG %>%
  filter(cyl==5)
```

```
## # A tibble: 4 x 5
##   manufacturer   hwy displ   cyl drv
##   <chr>        <int> <dbl> <int> <chr>
## 1 volkswagen      29   2.5     5 f
## 2 volkswagen      29   2.5     5 f
## 3 volkswagen      28   2.5     5 f
## 4 volkswagen      29   2.5     5 f
```

dplyr (Seleccionando filas)

Para elegir filas que cumplan una cierta condición, se usa el comando filter:

```
SelMPG %>%
  filter(hwy>25) %>%
  head(n=3)
```

```
## # A tibble: 3 x 5
##   manufacturer   hwy displ   cyl drv
##   <chr>        <int> <dbl> <int> <chr>
## 1 audi            29   1.8     4 f
## 2 audi            29   1.8     4 f
## 3 audi            31   2       4 f
```

dplyr (Seleccionando filas)

Para elegir filas que cumplan una cierta condición, se usa el comando filter:

```
SelMPG %>%
  filter(str_detect(manufacturer,pattern = "chev")) %>%
  head(n=3)
```

```
## # A tibble: 3 x 5
##   manufacturer   hwy displ   cyl drv
##   <chr>        <int> <dbl> <int> <chr>
## 1 chevrolet       20   5.3     8 r
## 2 chevrolet       15   5.3     8 r
## 3 chevrolet       20   5.3     8 r
```

dplyr (Transformando variables)

El paquete dplyr se encarga más de transformaciones de las variables y de las bases.

Para transformar una variable de la base, se usa el comando mutate. Por ejemplo, aquí vamos a agregarle 1 a la variable cyl:

```
head(SelMPG,n=3)
```

```
## # A tibble: 3 x 5
##   manufacturer   hwy displ   cyl drv
##   <chr>        <int> <dbl> <int> <chr>
## 1 audi            29   1.8     4 f
## 2 audi            29   1.8     4 f
## 3 audi            31   2       4 f
```

```
SelMPG %>%
  mutate(cyl=cyl+1) %>%
  head(n=3)
```

```
## # A tibble: 3 x 5
##   manufacturer   hwy displ   cyl drv
##   <chr>        <int> <dbl> <dbl> <chr>
## 1 audi            29   1.8     5 f
## 2 audi            29   1.8     5 f
## 3 audi            31   2       5 f
```

dplyr (Transformando variables)

En el caso en que la variable no sea de la base, agrega una columna con el nombre:

```
head(SelMPG,n=3)
```

```
## # A tibble: 3 x 5
##   manufacturer   hwy displ   cyl drv
##   <chr>        <int> <dbl> <int> <chr>
## 1 audi            29   1.8     4 f
## 2 audi            29   1.8     4 f
## 3 audi            31   2       4 f
```

```
SelMPG %>%
  mutate(dif=hwy/cyl) %>%
  head(n=3)
```

```
## # A tibble: 3 x 6
##   manufacturer   hwy displ   cyl drv     dif
##   <chr>        <int> <dbl> <int> <chr> <dbl>
## 1 audi            29   1.8     4 f      7.25
## 2 audi            29   1.8     4 f      7.25
## 3 audi            31   2       4 f      7.75
```

dplyr (Resumiendo variables)

Para calcular alguna función sobre las variables se puede usar el comando summarise. Aquí calculamos las medias de las variables hwy y displ, y tambien podemos contar cuantos audis hay en la base:

```
mpg %>%
  summarise(mHWY=mean(hwy),
            mDISPL=mean(displ),
            nAudi=sum(manufacturer=="audi"))
```

```
## # A tibble: 1 x 3
##    mHWY mDISPL nAudi
##   <dbl>  <dbl> <int>
## 1  23.4   3.47    18
```

dplyr (Resumiendo variables)

La gran ventaja del dplyr es que permite primero agrupar los datos y luego resumir las variables por grupo. Por ejemplo, si quisiéramos ver cuántas observaciones hay, cuál es la media de hwy y displ por cada grupo de drv:

```
ResMPG=mpg %>%
  group_by(drv) %>%
  summarise(n=n(),
            mHWY=mean(hwy),
            mDISPL=mean(displ))
ResMPG
```

```
## # A tibble: 3 x 4
##   drv       n  mHWY mDISPL
##   <chr> <int> <dbl>  <dbl>
## 1 4       103  19.2   4.00
## 2 f       106  28.2   2.56
## 3 r        25  21     5.18
```

dplyr (Resumiendo variables)

Se puede agrupar por varios factores:

```
mpg %>%
  group_by(drv,year) %>%
  summarise(n=n(),
            mHWY=mean(hwy),
            mDISPL=mean(displ))
```

```
## # A tibble: 6 x 5
## # Groups:   drv [3]
##   drv    year     n  mHWY mDISPL
##   <chr> <int> <int> <dbl>  <dbl>
## 1 4      1999    49  18.8   3.88
## 2 4      2008    54  19.5   4.11
## 3 f      1999    57  27.9   2.44
## 4 f      2008    49  28.4   2.70
## 5 r      1999    11  20.6   4.97
## 6 r      2008    14  21.3   5.34
```

dplyr (Ordenando bases)

La función arrange ordena la base según ciertas variables:

```
head(SelMPG,n=3)
```

```
## # A tibble: 3 x 5
##   manufacturer   hwy displ   cyl drv
##   <chr>        <int> <dbl> <int> <chr>
## 1 audi            29   1.8     4 f
## 2 audi            29   1.8     4 f
## 3 audi            31   2       4 f
```

```
SelMPG %>%
  arrange(hwy) %>%
  head(n=3)
```

```
## # A tibble: 3 x 5
##   manufacturer   hwy displ   cyl drv
##   <chr>        <int> <dbl> <int> <chr>
## 1 dodge           12   4.7     8 4
## 2 dodge           12   4.7     8 4
## 3 dodge           12   4.7     8 4
```

dplyr (Ordenando bases)

Se puede ordenar por más de una variable:

```
head(SelMPG,n=3)
```

```
## # A tibble: 3 x 5
##   manufacturer   hwy displ   cyl drv
##   <chr>        <int> <dbl> <int> <chr>
## 1 audi            29   1.8     4 f
## 2 audi            29   1.8     4 f
## 3 audi            31   2       4 f
```

```
SelMPG %>%
  arrange(displ,hwy) %>%
  head(n=3)
```

```
## # A tibble: 3 x 5
##   manufacturer   hwy displ   cyl drv
##   <chr>        <int> <dbl> <int> <chr>
## 1 honda           29   1.6     4 f
## 2 honda           32   1.6     4 f
## 3 honda           32   1.6     4 f
```

dplyr (Operaciones entre bases)

Considerando dos subconjuntos de la base, los que tienen rendimiento en autopista mayor a 25 y los chevrolets de la base, podemos realizar las siguientes operaciones:

- intersect: devuelve las observaciones comunes a ambas bases

```
Fil25=SelMPG %>% filter(hwy>25)
FilChev=SelMPG %>% filter(str_detect(manufacturer,pattern = "chev"))
intersect(Fil25,FilChev) %>%
  head()
```

```
## # A tibble: 6 x 5
##   manufacturer   hwy displ   cyl drv
##   <chr>        <int> <dbl> <int> <chr>
## 1 chevrolet       26   5.7     8 r
## 2 chevrolet       26   6.2     8 r
## 3 chevrolet       27   2.4     4 f
## 4 chevrolet       30   2.4     4 f
## 5 chevrolet       26   3.1     6 f
## 6 chevrolet       29   3.5     6 f
```

dplyr (Operaciones entre bases)

Considerando dos subconjuntos de la base, los que tienen rendimiento en autopista mayor a 25 y los chevrolets de la base, podemos realizar las siguientes operaciones:

- setdiff: devuelve las observaciones de la primer base sin los que son comunes a la segunda

```
Fil25=SelMPG %>% filter(hwy>25)
FilChev=SelMPG %>% filter(str_detect(manufacturer,pattern = "chev"))
setdiff(Fil25,FilChev) %>%
  head()
```

```
## # A tibble: 6 x 5
##   manufacturer   hwy displ   cyl drv
##   <chr>        <int> <dbl> <int> <chr>
## 1 audi            29   1.8     4 f
## 2 audi            31   2       4 f
## 3 audi            30   2       4 f
## 4 audi            26   2.8     6 f
## 5 audi            27   3.1     6 f
## 6 audi            26   1.8     4 4
```

dplyr (Operaciones entre bases)

Considerando dos subconjuntos de la base, los que tienen rendimiento en autopista mayor a 25 y los chevrolets de la base, podemos realizar las siguientes operaciones:

- setdiff: devuelve las observaciones de la primer base sin los que son comunes a la segunda (el orden importa)

```
Fil25=SelMPG %>% filter(hwy>25)
FilChev=SelMPG %>% filter(str_detect(manufacturer,pattern = "chev"))
setdiff(FilChev,Fil25) %>%
  head()
```

```
## # A tibble: 6 x 5
##   manufacturer   hwy displ   cyl drv
##   <chr>        <int> <dbl> <int> <chr>
## 1 chevrolet       20   5.3     8 r
## 2 chevrolet       15   5.3     8 r
## 3 chevrolet       17   5.7     8 r
## 4 chevrolet       17   6       8 r
## 5 chevrolet       23   5.7     8 r
## 6 chevrolet       25   6.2     8 r
```

dplyr (Operaciones entre bases)

Considerando dos subconjuntos de la base, los que tienen rendimiento en autopista mayor a 25 y los chevrolets de la base, podemos realizar las siguientes operaciones:

- union: devuelve la unión entre las dos bases, sin repetir las observaciones comunes

```
Fil25=SelMPG %>% filter(hwy>25)
FilChev=SelMPG %>% filter(str_detect(manufacturer,pattern = "chev"))
union(FilChev,Fil25) %>%
  head()
```

```
## # A tibble: 6 x 5
##   manufacturer   hwy displ   cyl drv
##   <chr>        <int> <dbl> <int> <chr>
## 1 chevrolet       20   5.3     8 r
## 2 chevrolet       15   5.3     8 r
## 3 chevrolet       17   5.7     8 r
## 4 chevrolet       17   6       8 r
## 5 chevrolet       26   5.7     8 r
## 6 chevrolet       23   5.7     8 r
```

dplyr (Operaciones entre bases)

Considerando dos subconjuntos de la base, los que tienen rendimiento en autopista mayor a 25 y los chevrolets de la base, podemos realizar las siguientes operaciones:

- union: devuelve la unión entre las dos bases, sin repetir las observaciones comunes

```
Fil25=SelMPG %>% filter(hwy>25)
FilChev=SelMPG %>% filter(str_detect(manufacturer,pattern = "chev"))
union(FilChev,Fil25) %>%
  tail()
```

```
## # A tibble: 6 x 5
##   manufacturer   hwy displ   cyl drv
##   <chr>        <int> <dbl> <int> <chr>
## 1 volkswagen      41   1.9     4 f
## 2 volkswagen      28   2.5     5 f
## 3 volkswagen      29   1.8     4 f
## 4 volkswagen      28   2       4 f
## 5 volkswagen      26   2.8     6 f
## 6 volkswagen      26   3.6     6 f
```

dplyr (Operaciones entre bases)

Estas operaciones tienen menos sentido cuando se tratan de subconjuntos de la misma base, ya que se pueden manejar con el comando filter:

```
SelMPG %>%
  filter(hwy>25 & str_detect(manufacturer,pattern = "chev"))%>%
  head(n=3)
```

```
## # A tibble: 3 x 5
##   manufacturer   hwy displ   cyl drv
##   <chr>        <int> <dbl> <int> <chr>
## 1 chevrolet       26   5.7     8 r
## 2 chevrolet       26   6.2     8 r
## 3 chevrolet       27   2.4     4 f
```

```
intersect(Fil25,FilChev) %>%
  head(n=3)
```

```
## # A tibble: 3 x 5
##   manufacturer   hwy displ   cyl drv
##   <chr>        <int> <dbl> <int> <chr>
## 1 chevrolet       26   3.1     6 f
## 2 chevrolet       26   3.6     6 f
## 3 chevrolet       26   5.7     8 r
```

dplyr (Operaciones entre bases)

Estas operaciones tienen menos sentido cuando se tratan de subconjuntos de la misma base, ya que se pueden manejar con el comando filter:

```
SelMPG %>%
  filter(hwy>25 & !str_detect(manufacturer,pattern = "chev"))%>%
  head(n=3)
```

```
## # A tibble: 3 x 5
##   manufacturer   hwy displ   cyl drv
##   <chr>        <int> <dbl> <int> <chr>
## 1 audi            29   1.8     4 f
## 2 audi            29   1.8     4 f
## 3 audi            31   2       4 f
```

```
setdiff(Fil25,FilChev) %>%
  head(n=3)
```

```
## # A tibble: 3 x 5
##   manufacturer   hwy displ   cyl drv
##   <chr>        <int> <dbl> <int> <chr>
## 1 audi            26   1.8     4 4
## 2 audi            26   2.8     6 f
## 3 audi            27   2       4 4
```

dplyr (Operaciones entre bases)

Estas operaciones tienen menos sentido cuando se tratan de subconjuntos de la misma base, ya que se pueden manejar con el comando filter:

```
SelMPG %>%
  filter(!hwy>25 & str_detect(manufacturer,pattern = "chev")) %>%
  head(n=3)
```

```
## # A tibble: 3 x 5
##   manufacturer   hwy displ   cyl drv
##   <chr>        <int> <dbl> <int> <chr>
## 1 chevrolet       20   5.3     8 r
## 2 chevrolet       15   5.3     8 r
## 3 chevrolet       20   5.3     8 r
```

```
setdiff(FilChev,Fil25) %>%
  head(n=3)
```

```
## # A tibble: 3 x 5
##   manufacturer   hwy displ   cyl drv
##   <chr>        <int> <dbl> <int> <chr>
## 1 chevrolet       14   5.3     8 4
## 2 chevrolet       15   5.3     8 r
## 3 chevrolet       15   5.7     8 4
```

dplyr (Operaciones entre bases)

Estas operaciones tienen menos sentido cuando se tratan de subconjuntos de la misma base, ya que se pueden manejar con el comando filter:

```
SelMPG %>%
  filter(hwy>25 | str_detect(manufacturer,pattern = "chev")) %>%
  head(n=3)
```

```
## # A tibble: 3 x 5
##   manufacturer   hwy displ   cyl drv
##   <chr>        <int> <dbl> <int> <chr>
## 1 audi            29   1.8     4 f
## 2 audi            29   1.8     4 f
## 3 audi            31   2       4 f
```

```
union(Fil25,FilChev) %>%
  head(n=3)
```

```
## # A tibble: 3 x 5
##   manufacturer   hwy displ   cyl drv
##   <chr>        <int> <dbl> <int> <chr>
## 1 audi            26   1.8     4 4
## 2 audi            26   2.8     6 f
## 3 audi            27   2       4 4
```

dplyr (Operaciones entre bases)

Estas operaciones tienen menos sentido cuando se tratan de subconjuntos de la misma base, ya que se pueden manejar con el comando filter:

```
SelMPG %>%
  filter(hwy>25 | str_detect(manufacturer,pattern = "chev")) %>%
  tail(n=3)
```

```
## # A tibble: 3 x 5
##   manufacturer   hwy displ   cyl drv
##   <chr>        <int> <dbl> <int> <chr>
## 1 volkswagen      26   2.8     6 f
## 2 volkswagen      26   2.8     6 f
## 3 volkswagen      26   3.6     6 f
```

```
union(FilChev,Fil25) %>%
  tail(n=3)
```

```
## # A tibble: 3 x 5
##   manufacturer   hwy displ   cyl drv
##   <chr>        <int> <dbl> <int> <chr>
## 1 volkswagen      29   2.5     5 f
## 2 volkswagen      41   1.9     4 f
## 3 volkswagen      44   1.9     4 f
```

dplyr (Operaciones entre bases)

Sin embargo, estas operaciones son de mucha utilidad cuando se comparan dos bases distintas:

```
BaseA=data.frame(x=1:4,y=rep(c("A","B"),2));BaseA
```

```
##   x y
## 1 1 A
## 2 2 B
## 3 3 A
## 4 4 B
```

```
BaseB=data.frame(x=1:4,y=rep(c("A","B"),each=2));BaseB
```

```
##   x y
## 1 1 A
## 2 2 A
## 3 3 B
## 4 4 B
```

dplyr (Operaciones entre bases)

Sin embargo, estas operaciones son de mucha utilidad cuando se comparan dos bases distintas:

```
intersect(BaseA,BaseB)
```

```
##   x y
## 1 1 A
## 2 4 B
```

```
union(BaseA,BaseB)
```

```
##   x y
## 1 1 A
## 2 2 B
## 3 3 A
## 4 4 B
## 5 2 A
## 6 3 B
```

dplyr (Operaciones entre bases)

Sin embargo, estas operaciones son de mucha utilidad cuando se comparan dos bases distintas:

```
setdiff(BaseA,BaseB)
```

```
##   x y
## 1 2 B
## 2 3 A
```

```
setdiff(BaseB,BaseA)
```

```
##   x y
## 1 2 A
## 2 3 B
```

Procesamiento secuencial de datos

Combinando varios de estos comandos, se puede procesar línea por línea los datos:

```
mpg %>%
  filter(hwy>25) %>%
  head()
```

```
## # A tibble: 6 x 11
##   manufacturer model displ  year   cyl trans      drv     cty   hwy fl
##   <chr>        <chr> <dbl> <int> <int> <chr>      <chr> <int> <int> <chr
## 1 audi         a4      1.8  1999     4 auto(l5)   f        18    29 p
## 2 audi         a4      1.8  1999     4 manual(m5) f        21    29 p
## 3 audi         a4      2    2008     4 manual(m6) f        20    31 p
## 4 audi         a4      2    2008     4 auto(av)   f        21    30 p
## 5 audi         a4      2.8  1999     6 auto(l5)   f        16    26 p
## 6 audi         a4      2.8  1999     6 manual(m5) f        18    26 p
```

Procesamiento secuencial de datos

Combinando varios de estos comandos, se puede procesar línea por línea los datos:

```
mpg %>%
  filter(hwy>25) %>%
  select(hwy,displ,drv) %>%
  head()
```

```
## # A tibble: 6 x 3
##     hwy displ drv
##   <int> <dbl> <chr>
## 1    29   1.8 f
## 2    29   1.8 f
## 3    31   2   f
## 4    30   2   f
## 5    26   2.8 f
## 6    26   2.8 f
```

Procesamiento secuencial de datos

Combinando varios de estos comandos, se puede procesar línea por línea los datos:

```
mpg %>%
  filter(hwy>25) %>%
  select(hwy,displ,drv) %>%
  group_by(drv) %>%
  head()
```

```
## # A tibble: 6 x 3
## # Groups:   drv [1]
##     hwy displ drv
##   <int> <dbl> <chr>
## 1    29   1.8 f
## 2    29   1.8 f
## 3    31   2   f
## 4    30   2   f
## 5    26   2.8 f
## 6    26   2.8 f
```

Procesamiento secuencial de datos

Combinando varios de estos comandos, se puede procesar línea por línea los datos:

```
mpg %>%
  filter(hwy>25) %>%
  select(hwy,displ,drv) %>%
  group_by(drv) %>%
  summarise(mHWY=mean(hwy))
```

```
## # A tibble: 3 x 2
##   drv    mHWY
##   <chr> <dbl>
## 1 4      26.5
## 2 f      29.3
## 3 r      26
```

Procesamiento secuencial de datos

Combinando varios de estos comandos, se puede procesar línea por línea los datos:

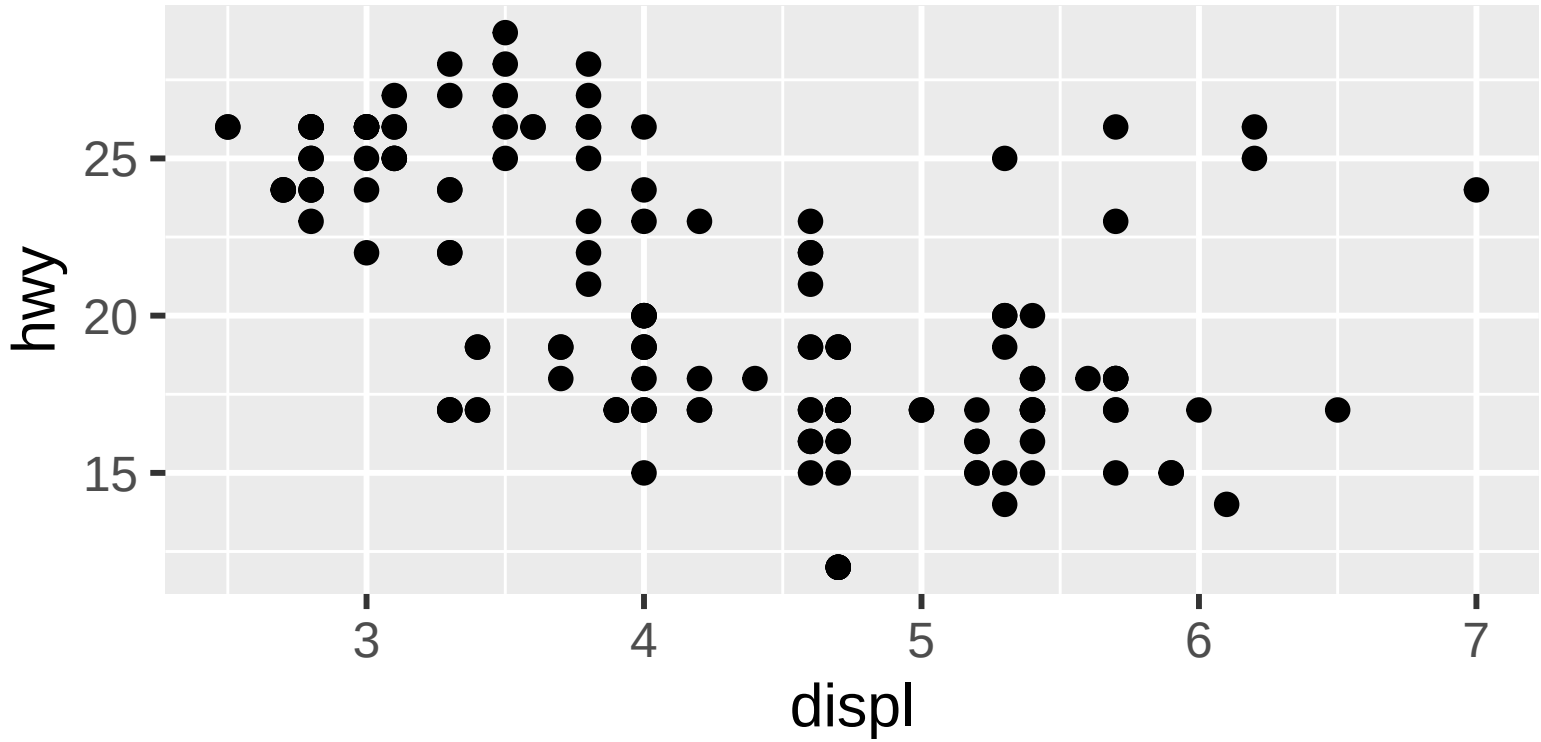
```
mpg %>%
  filter(hwy>25) %>%
  select(hwy,displ,drv) %>%
  group_by(drv) %>%
  summarise(mHWY=mean(hwy)) %>%
  arrange(mHWY)
```

```
## # A tibble: 3 x 2
##   drv    mHWY
##   <chr> <dbl>
## 1 r      26
## 2 4      26.5
## 3 f      29.3
```

Complementación con ggplot

Por empezar, el uso del pipeline ayuda a primero procesar la base, y luego a graficar sus variables. Por ejemplo, vamos a graficar los puntos con coordenada x displ y coordenada y hwy, de las observaciones con más de 5 cilindros:

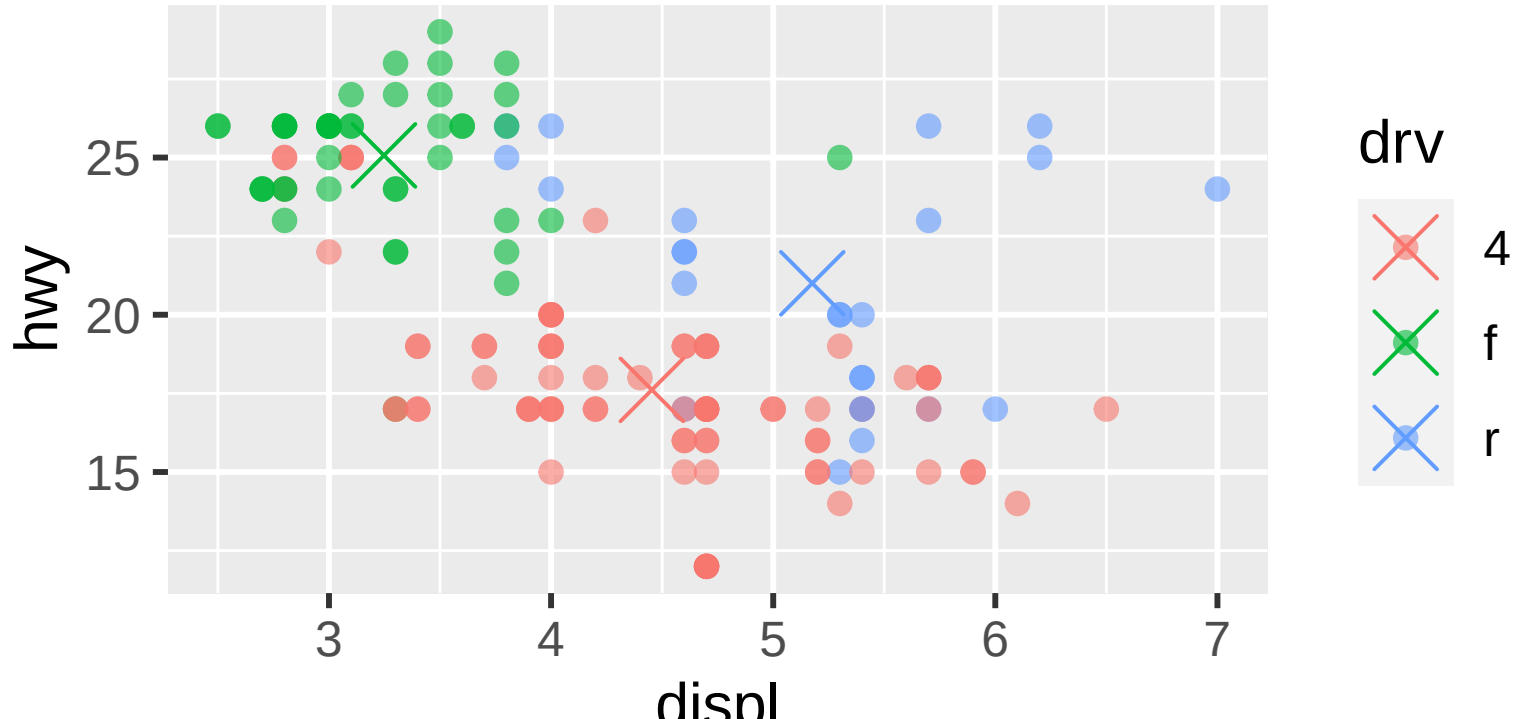
```
mpg %>%
  filter(cyl>5) %>%
  ggplot(aes(x=displ,y=hwy))+
  geom_point()
```



Complementación con ggplot

Además, le podemos agregar las medias de displ y hwy por cada nivel de drv:

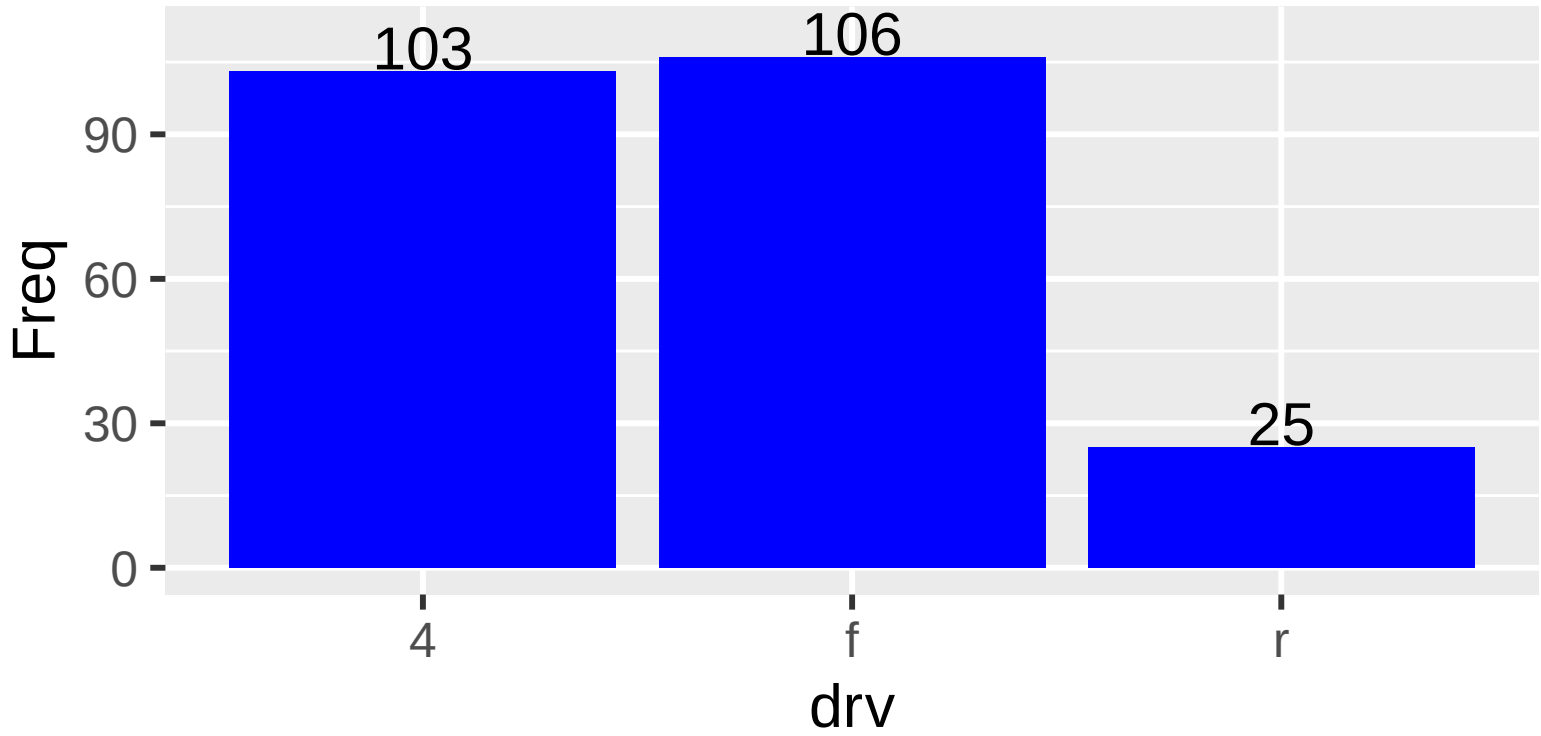
```
mpg %>%
  filter(cyl>5) %>%
  ggplot(aes(x=displ,y=hwy,color=drv))+
  geom_point(alpha=0.6)+
  mpg %>%
  filter(cyl>5) %>%
  group_by(drv) %>%
  summarise(mDISPL=mean(displ),mHWY=mean(hwy)) %>%
  geom_point(data=.,mapping=aes(x=mDISPL,y=mHWY,color=drv),shape=4,size=5)
```



Complementación con ggplot

Otra ventaja es poder tener una ubicación en los gráficos de barras para agregar texto:

```
mpg %>%
  group_by(drv) %>%
  summarise(Freq=n()) %>%
  ggplot(aes(x=drv))+
  geom_bar(aes(y=Freq),fill="blue",stat = "identity")+
  geom_text(aes(y=Freq,label=Freq),nudge_y = 5)
```



Datos temporales

Base storms

Consideremos la base storms, correspondiente al paquete dplyr, contiene datos sobre los distintos huracanes en el Atlántico norte.

```
head(storms)
```

```
## # A tibble: 6 x 13
##   name   year month   day  hour   lat  long status category  wind pressur
##   <chr> <dbl> <dbl> <int> <dbl> <dbl> <dbl> <chr>  <ord>    <int>    <int
## 1 Amy    1975     6    27     0  27.5 -79   tropi~ -1          25      101
## 2 Amy    1975     6    27     6  28.5 -79   tropi~ -1          25      101
## 3 Amy    1975     6    27    12  29.5 -79   tropi~ -1          25      101
## 4 Amy    1975     6    27    18  30.5 -79   tropi~ -1          25      101
## 5 Amy    1975     6    28     0  31.5 -78.8 tropi~ -1          25      101
## 6 Amy    1975     6    28     6  32.4 -78.7 tropi~ -1          25      101
## # ... with 2 more variables: ts_diameter <dbl>, hu_diameter <dbl>
```

Base storms

Consideremos la base storms, correspondiente al paquete dplyr, contiene datos sobre los distintos huracanes en el Atlántico norte.

```
str(storms)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    10010 obs. of  13 variables
##  $ name       : chr  "Amy" "Amy" "Amy" "Amy" ...
##  $ year       : num  1975 1975 1975 1975 1975 ...
##  $ month      : num  6 6 6 6 6 6 6 6 6 6 ...
##  $ day        : int  27 27 27 27 28 28 28 28 29 29 ...
##  $ hour       : num  0 6 12 18 0 6 12 18 0 6 ...
##  $ lat        : num  27.5 28.5 29.5 30.5 31.5 32.4 33.3 34 34.4 34 ...
##  $ long       : num  -79 -79 -79 -79 -78.8 -78.7 -78 -77 -75.8 -74.8 ...
##  $ status     : chr  "tropical depression" "tropical depression" "tropica
##  $ category   : Ord.factor w/ 7 levels "-1"<"0"<"1"<"2"<..: 1 1 1 1 1 1 1
##  $ wind       : int  25 25 25 25 25 25 25 30 35 40 ...
##  $ pressure   : int  1013 1013 1013 1013 1012 1012 1011 1006 1004 1002 .
##  $ ts_diameter: num  NA NA NA NA NA NA NA NA NA NA ...
##  $ hu_diameter: num  NA NA NA NA NA NA NA NA NA NA ...
```

Base storms

Notemos que la fecha está dividida en tres variables: año, mes y día. Podemos unir las variables usando unite:

```
storms %>% select(name,year,month,day,hour,wind,pressure) %>% head(n=3)
```

```
## # A tibble: 3 x 7
##   name   year month   day  hour  wind pressure
##   <chr> <dbl> <dbl> <int> <dbl> <int>    <int>
## 1 Amy    1975     6    27     0    25     1013
## 2 Amy    1975     6    27     6    25     1013
## 3 Amy    1975     6    27    12    25     1013
```

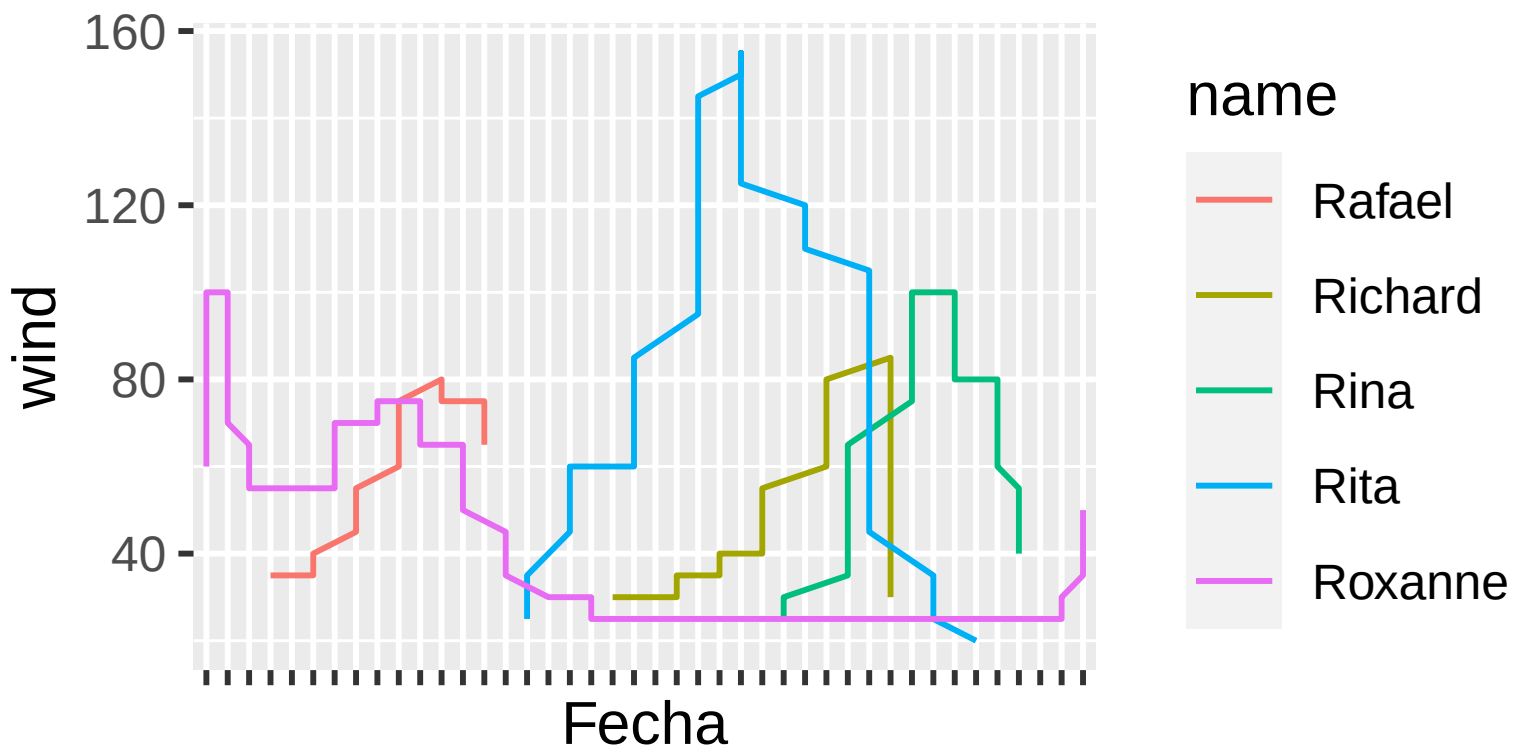
```
ModStorms=storms %>%
  unite(col="Fecha",day,month,year,sep="-") %>%
  select(name,Fecha,hour,wind,pressure)
ModStorms %>%
  head(n=3)
```

```
## # A tibble: 3 x 5
##   name  Fecha      hour  wind pressure
##   <chr> <chr>     <dbl> <int>    <int>
## 1 Amy   27-6-1975     0    25     1013
## 2 Amy   27-6-1975     6    25     1013
## 3 Amy   27-6-1975    12    25     1013
```

Base storms

Con esta variable Fecha, podemos intentar ver cómo evoluciona la intensidad del viento para cada tormenta, cuyo nombre comienza con "R" (para que sea visible):

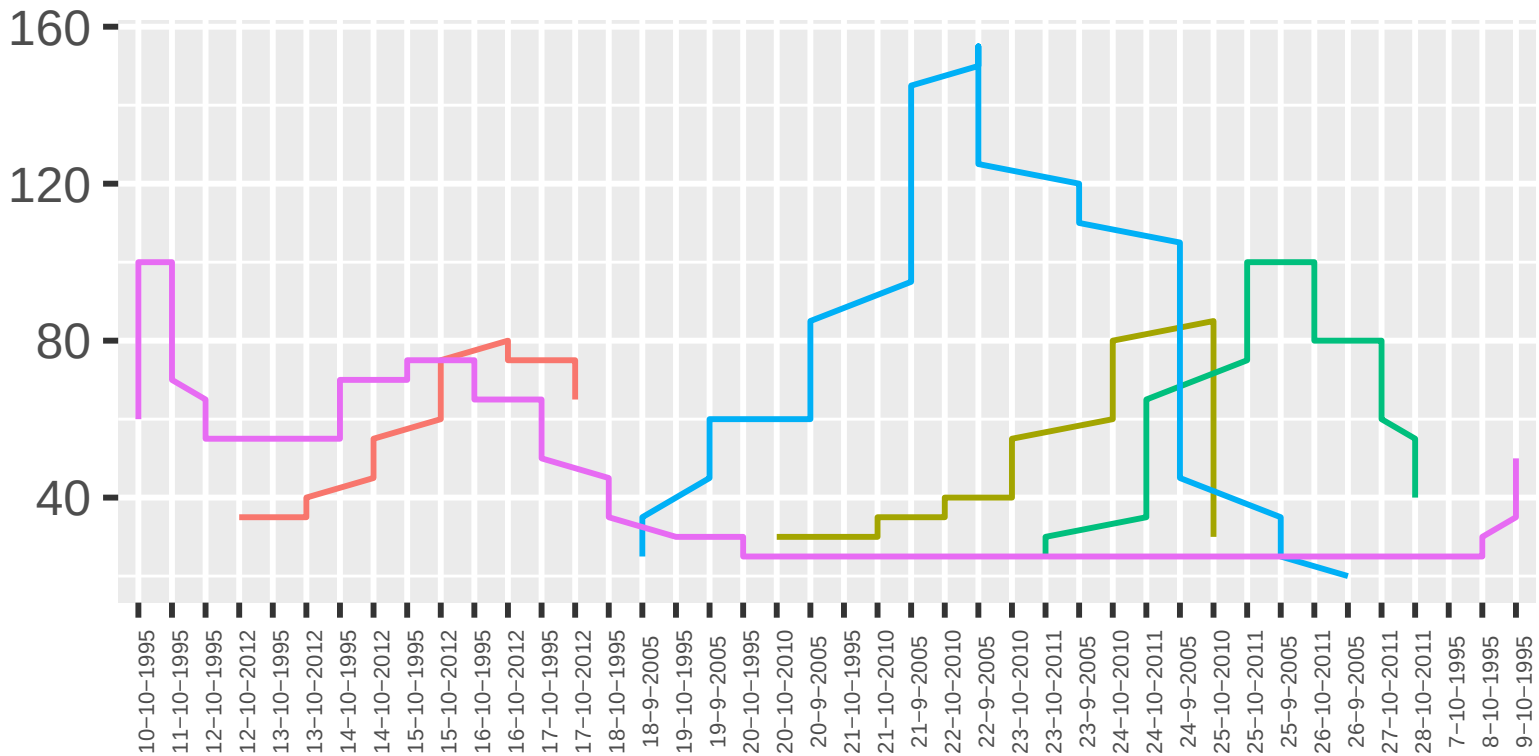
```
ModStorms %>%
  filter(str_detect(name,pattern="R")) %>%
  ggplot(aes(x=Fecha,y=wind,color=name))+
  geom_line(aes(group=name))+
  theme(axis.text.x = element_blank())
```



Base storms

Sin embargo, notar que si le agregamos texto al eje x, vemos que ggplot interpreta la variable fecha como un factor (están todas las marcas del eje igualmente distanciadas). Además, no están correctamente ordenados:

```
ModStorms %>%
  filter(str_detect(name,pattern="R")) %>%
  ggplot(aes(x=Fecha,y=wind,color=name))+
  geom_line(aes(group=name))+
  theme(axis.text.x = element_text(angle=90,size=4),
        axis.title = element_blank(),legend.position = "none")
```



Tipo de dato "Date"

Esto se debe a que interpreta la columna Fecha como caracteres, para que adopte el formato correcto (por lo menos, debería estar ordenado). Para eso está el tipo de dato "Date", podemos coercionar los datos a ser del tipo Fecha:

```
head(ModStorms$Fecha,n=5)
```

```
## [1] "27-6-1975" "27-6-1975" "27-6-1975" "27-6-1975" "28-6-1975"
```

```
NewFecha=as.Date(head(ModStorms$Fecha,n=5))
NewFecha
```

```
## [1] "27-06-19" "27-06-19" "27-06-19" "27-06-19" "28-06-19"
```

```
class(NewFecha)
```

```
## [1] "Date"
```

Tipo de dato "Date"

Esto se debe a que interpreta la columna Fecha como caracteres, para que adopte el formato correcto (por lo menos, debería estar ordenado). Para eso está el tipo de dato "Date", podemos coercionar los datos a ser del tipo Fecha:

```
head(ModStorms$Fecha,n=5)
```

```
## [1] "27-6-1975" "27-6-1975" "27-6-1975" "27-6-1975" "28-6-1975"
```

```
NewFecha=as.Date(head(ModStorms$Fecha,n=5))
NewFecha
```

```
## [1] "27-06-19" "27-06-19" "27-06-19" "27-06-19" "28-06-19"
```

```
class(NewFecha)
```

```
## [1] "Date"
```

Notemos que no lo transformó bien. Eso es porque no interpreta correctamente cuáles son los días, meses y año.

Paquete lubridate

Estos errores se pueden arreglar agregando argumentos a as.Date. Sin embargo, el mejor paquete para trabajar con fechas es el paquete lubridate. Tiene funciones que transforman cualquier formato de fecha en el adecuado, cuando se especifica el orden de los días "d", los meses "m" y los años "y":

```
require(lubridate)
dmy("22-06-1986")
```

```
## [1] "1986-06-22"
```

```
mdy("Jun 22 1986")
```

```
## [1] "1986-06-22"
```

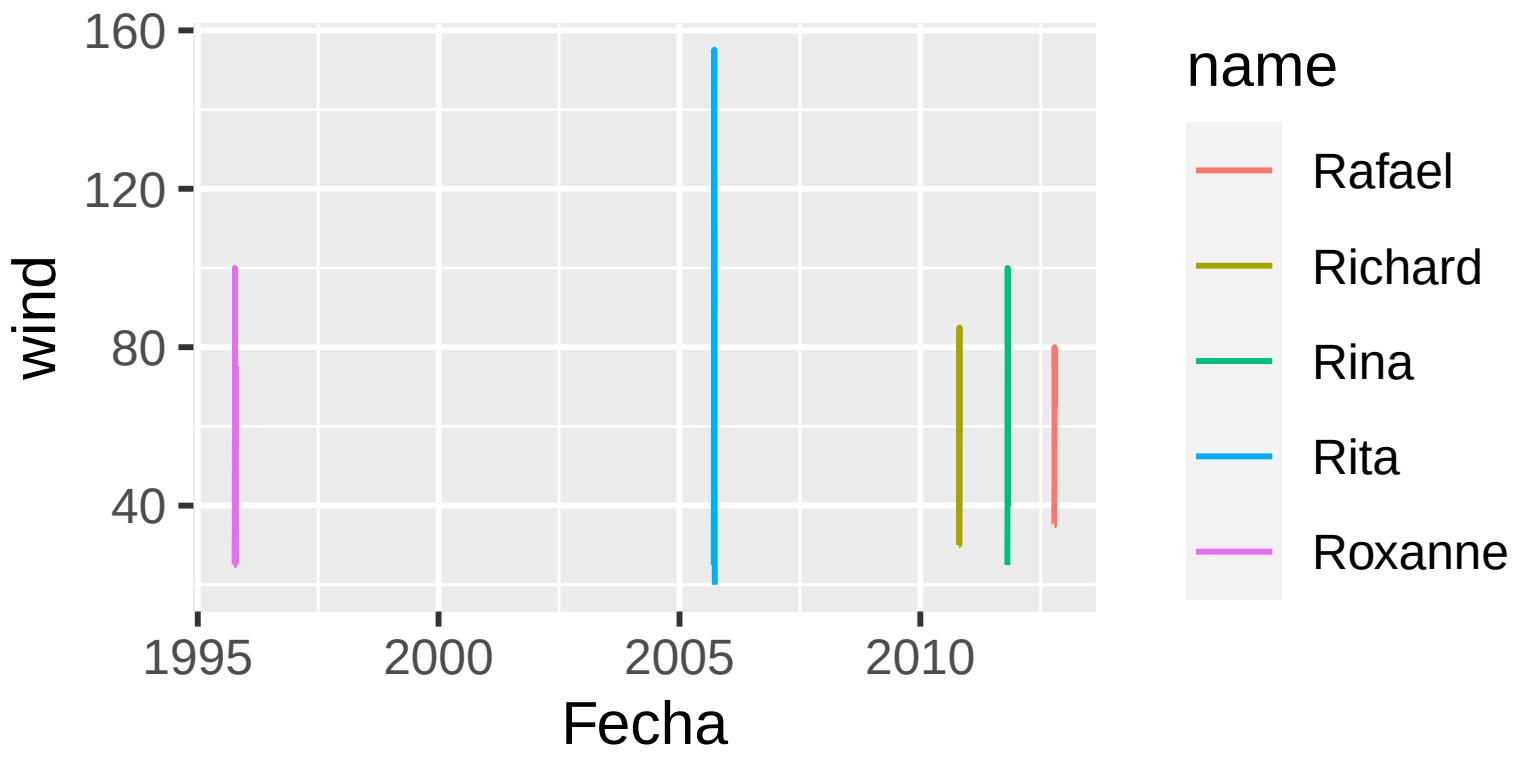
```
ymd("86/6/22")
```

```
## [1] "1986-06-22"
```

Transformación de Fecha

Por lo tanto, vamos a modificar los datos para que sean fechas y volver a correr el gráfico:

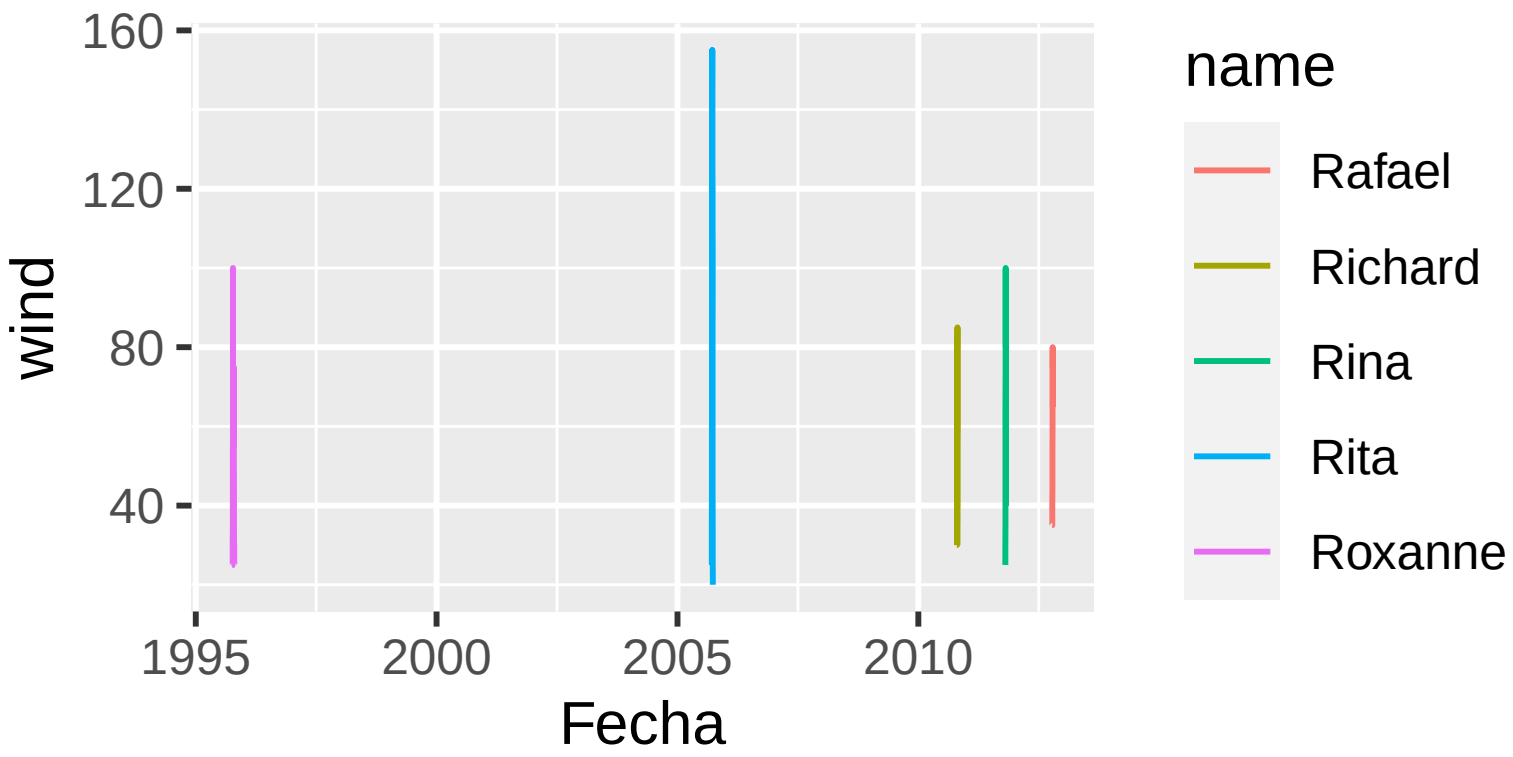
```
ModStorms %>%
  filter(str_detect(name,pattern="R")) %>%
  mutate(Fecha=dmy(Fecha)) %>%
  ggplot(aes(x=Fecha,y=wind,color=name))+
  geom_line(aes(group=name))
```



Transformación de Fecha

Por lo tanto, vamos a modificar los datos para que sean fechas y volver a correr el gráfico:

```
ModStorms %>%
  filter(str_detect(name,pattern="R")) %>%
  mutate(Fecha=dmy(Fecha)) %>%
  ggplot(aes(x=Fecha,y=wind,color=name))+
  geom_line(aes(group=name))
```

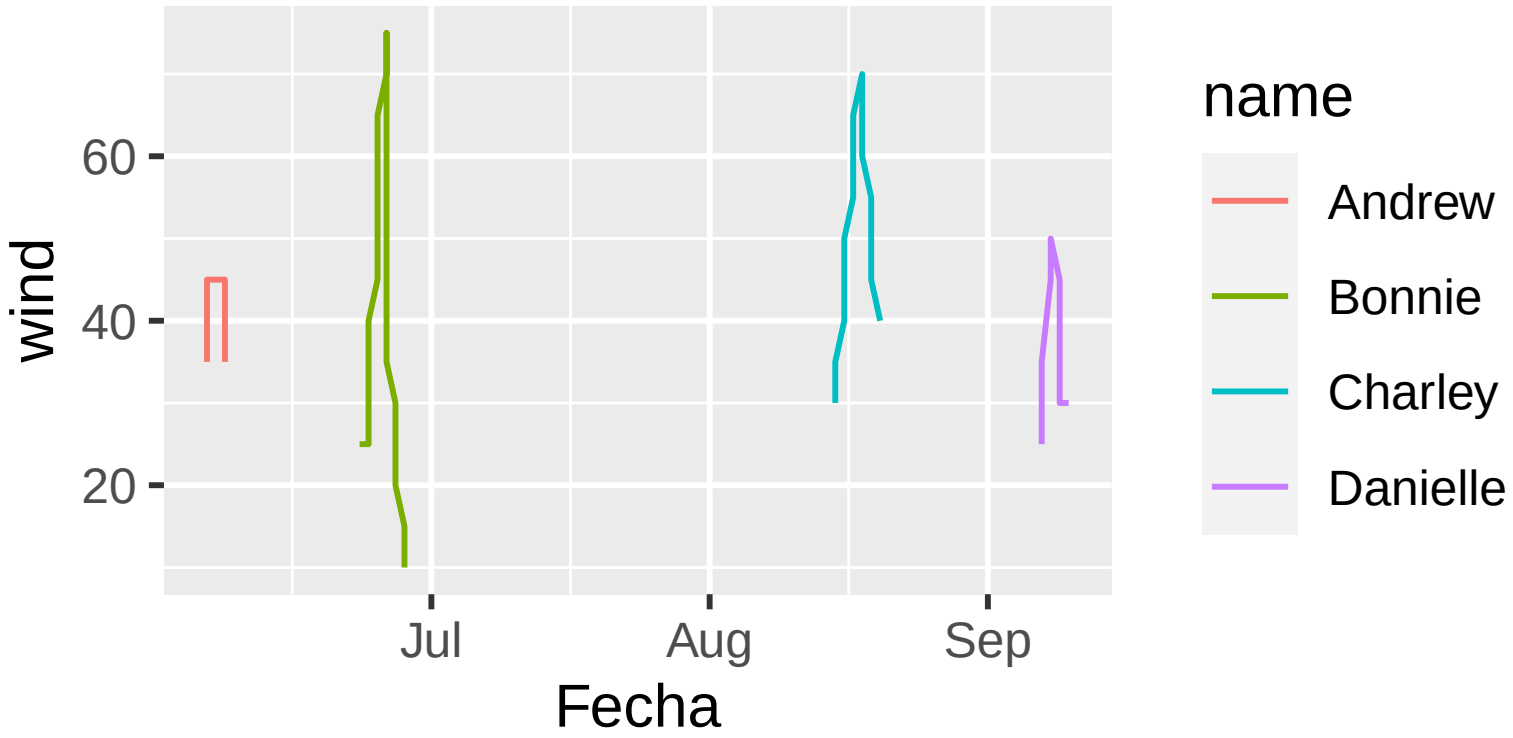


Notar que los gráficos quedan ordenados, aunque no se aprecia mucho la comparación

Transformación de Fecha

Vamos a cambiar el enfoque, vamos a analizar las tormentas del año 1986:

```
ModStorms %>%
  filter(str_detect(Fecha,pattern="1986")) %>%
  mutate(Fecha=dmy(Fecha)) %>%
  ggplot(aes(x=Fecha,y=wind,color=name))+
  geom_line(aes(group=name))
```

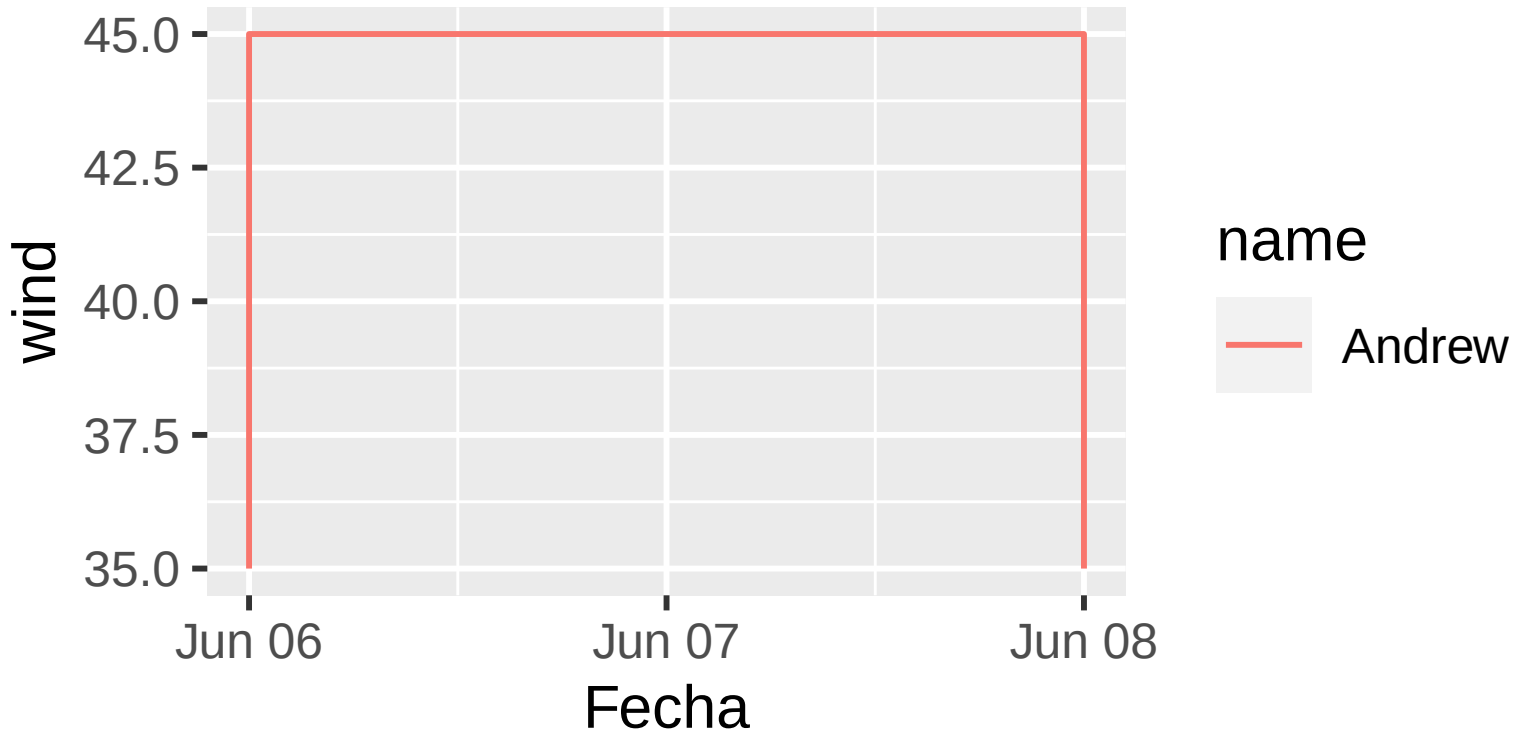


Notar que la tormenta "Andrew" tiene una forma de líneas rectas, no es muy común en evoluciones temporales, ya que siempre hay un incremento temporal.

Inclusión de las horas

Esto se debe a que no consideramos las horas. Miremos específicamente la tormenta "Andrew":

```
ModStorms %>%
  filter(str_detect(Fecha,pattern="1986")) %>%
  filter(name=="Andrew") %>%
  mutate(Fecha=dmy(Fecha)) %>%
  ggplot(aes(x=Fecha,y=wind,color=name))+
  geom_line(aes(group=name))
```



Inclusión de las horas

Vamos a volver a unir las columnas Fecha con la variable hour y llamarla FechaHora:

```
head(ModStorms,n=3)
```

```
## # A tibble: 3 x 5
##   name  Fecha      hour  wind pressure
##   <chr> <chr>     <dbl> <int>    <int>
## 1 Amy   27-6-1975     0    25     1013
## 2 Amy   27-6-1975     6    25     1013
## 3 Amy   27-6-1975    12    25     1013
```

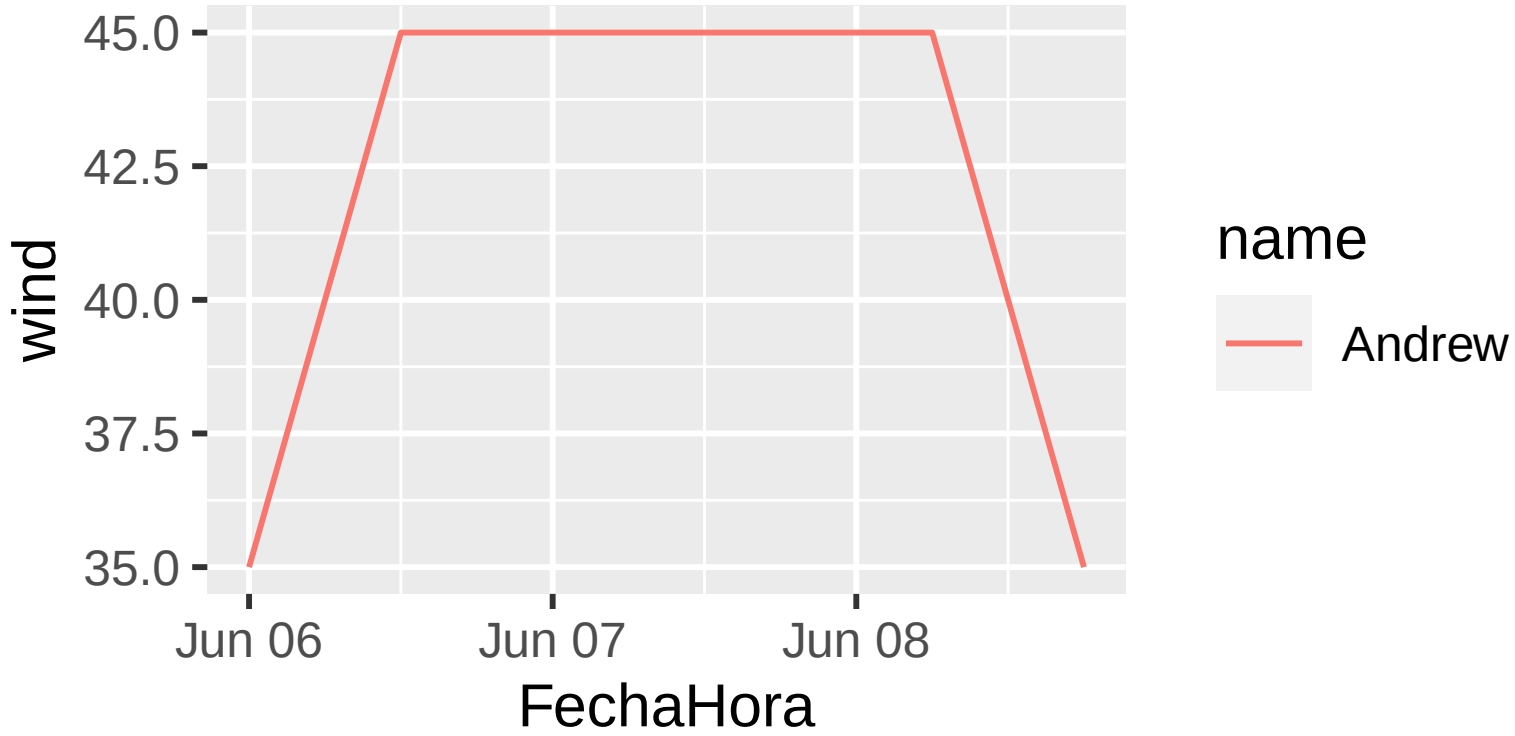
```
ModStorms %>%
  unite(col = "FechaHora",Fecha,hour,sep=" ") %>%
  head(n=3)
```

```
## # A tibble: 3 x 4
##   name  FechaHora      wind pressure
##   <chr> <chr>         <int>    <int>
## 1 Amy   27-6-1975 0      25     1013
## 2 Amy   27-6-1975 6      25     1013
## 3 Amy   27-6-1975 12     25     1013
```

Inclusión de las horas

De este modo, vamos a transformar la FechaHora usando el comando de lubridate llamado dmy_h:

```
ModStorms %>%
  filter(str_detect(Fecha,pattern="1986")) %>%
  unite(col = "FechaHora",Fecha,hour,sep=" ") %>%
  filter(name=="Andrew") %>%
  mutate(FechaHora=dmy_h(FechaHora)) %>%
  ggplot(aes(x=FechaHora,y=wind,color=name))+
  geom_line(aes(group=name))
```



Minutos y segundos

Por último, si a esta variable le agregamos minutos y segundos, podemos agregar minutos y segundos a FechaHora:

```
head(ModStorms,n=3)
```

```
## # A tibble: 3 x 4
##   name   FechaHora    wind pressure
##   <chr>  <chr>       <int>    <int>
## 1 Andrew 6-6-1986 0     35     1002
## 2 Andrew 6-6-1986 6     40     1003
## 3 Andrew 6-6-1986 12    45     1005
```

```
ModStorms=ModStorms %>%
  mutate(FechaHMS=paste0(FechaHora,":0:0"))
ModStorms %>%
  head(n=3)
```

```
## # A tibble: 3 x 5
##   name   FechaHora    wind pressure FechaHMS
##   <chr>  <chr>       <int>    <int> <chr>
## 1 Andrew 6-6-1986 0     35     1002 6-6-1986 0:0:0
## 2 Andrew 6-6-1986 6     40     1003 6-6-1986 6:0:0
## 3 Andrew 6-6-1986 12    45     1005 6-6-1986 12:0:0
```

Minutos y segundos

Ahora podemos usar el comando dmy_hms (va a dar el mismo resultado):

```
ModStorms %>%
  mutate(FechaHMS=dmy_hms(FechaHMS)) %>%
  head(n=3)
```

```
## # A tibble: 3 x 5
##   name   FechaHora    wind pressure FechaHMS
##   <chr>  <chr>       <int>    <int> <dttm>
## 1 Andrew 6-6-1986 0     35     1002 1986-06-06 00:00:00
## 2 Andrew 6-6-1986 6     40     1003 1986-06-06 06:00:00
## 3 Andrew 6-6-1986 12    45     1005 1986-06-06 12:00:00
```

Datos faltantes

Base storms

Las últimas 2 variables de la base storm miden el diámetro de las áreas afectadas por vientos de tormenta tropical y huracanados (de más de 32 y 64 nudos, respectivamente). Como algunas tormentas no cumplen esta condición, la base tiene datos faltantes:

```
storms %>%  select(name,ts_diameter,hu_diameter) %>% head(n=4)
```

```
## # A tibble: 4 x 3
##   name  ts_diameter hu_diameter
##   <chr>       <dbl>       <dbl>
## 1 Amy            NA          NA
## 2 Amy            NA          NA
## 3 Amy            NA          NA
## 4 Amy            NA          NA
```

Base storms

Las últimas 2 variables de la base storm miden el diámetro de las áreas afectadas por vientos de tormenta tropical y huracanados (de más de 32 y 64 nudos, respectivamente). Como algunas tormentas no cumplen esta condición, la base tiene datos faltantes:

```
storms %>%  select(name,ts_diameter,hu_diameter) %>% head(n=4)
```

```
## # A tibble: 4 x 3
##   name  ts_diameter hu_diameter
##   <chr>       <dbl>       <dbl>
## 1 Amy            NA          NA
## 2 Amy            NA          NA
## 3 Amy            NA          NA
## 4 Amy            NA          NA
```

De hecho, podemos ver que son las únicas columnas que tienen datos faltantes:

```
colSums(is.na(storms))
```

```
##        name        year       month         day        hour         lat
##           0           0           0           0           0           0
##        long      status    category        wind    pressure ts_diameter
##           0           0           0           0           0        6528
## hu_diameter
##        6528
```

Filtrado de datos faltantes

Se pueden filtrar los datos faltantes de cada columna apelando al comando filter:

```
SubTS=storms %>%
  select(name,ts_diameter,hu_diameter,wind,pressure,year,month,day)%>%
  filter(!is.na(ts_diameter))
SubTS %>%
  head()
```

```
## # A tibble: 6 x 8
##   name  ts_diameter hu_diameter  wind pressure  year month   day
##   <chr>       <dbl>       <dbl> <int>    <int> <dbl> <dbl> <int>
## 1 Alex          0             0    25     1010  2004     7    31
## 2 Alex          0             0    25     1009  2004     8     1
## 3 Alex          0             0    25     1009  2004     8     1
## 4 Alex          0             0    30     1009  2004     8     1
## 5 Alex         57.5           0    35     1009  2004     8     1
## 6 Alex         57.5           0    35     1007  2004     8     2
```

Filtrado de datos faltantes

Se pueden filtrar los datos faltantes de cada columna apelando al comando filter:

```
SubTS=storms %>%
  select(name,ts_diameter,hu_diameter,wind,pressure,year,month,day)%>%
  filter(!is.na(ts_diameter))
SubTS %>%
  head()
```

```
## # A tibble: 6 x 8
##   name  ts_diameter hu_diameter  wind pressure  year month   day
##   <chr>       <dbl>       <dbl> <int>    <int> <dbl> <dbl> <int>
## 1 Alex          0             0    25     1010  2004     7    31
## 2 Alex          0             0    25     1009  2004     8     1
## 3 Alex          0             0    25     1009  2004     8     1
## 4 Alex          0             0    30     1009  2004     8     1
## 5 Alex         57.5           0    35     1009  2004     8     1
## 6 Alex         57.5           0    35     1007  2004     8     2
```

Notemos que por definición, ts_diameter deja de ser cero cuando wind pasa el valor de 32.

Filtrado de datos faltantes

Se pueden filtrar los datos faltantes de cada columna apelando al comando filter:

```
SubHU=storms %>%
  select(name,ts_diameter,hu_diameter,wind,pressure,year,month,day)%>%
  filter(!is.na(hu_diameter))
SubHU %>%
  head()
```

```
## # A tibble: 6 x 8
##   name  ts_diameter hu_diameter  wind pressure  year month   day
##   <chr>       <dbl>       <dbl> <int>    <int> <dbl> <dbl> <int>
## 1 Alex          0             0    25     1010  2004     7    31
## 2 Alex          0             0    25     1009  2004     8     1
## 3 Alex          0             0    25     1009  2004     8     1
## 4 Alex          0             0    30     1009  2004     8     1
## 5 Alex         57.5           0    35     1009  2004     8     1
## 6 Alex         57.5           0    35     1007  2004     8     2
```

Filtrado de datos faltantes

Se pueden filtrar los datos faltantes de cada columna apelando al comando filter:

```
SubHU=storms %>%
  select(name,ts_diameter,hu_diameter,wind,pressure,year,month,day)%>%
  filter(!is.na(hu_diameter))
SubHU %>%
  head()
```

```
## # A tibble: 6 x 8
##   name  ts_diameter hu_diameter  wind pressure  year month   day
##   <chr>       <dbl>       <dbl> <int>    <int> <dbl> <dbl> <int>
## 1 Alex          0             0    25     1010  2004     7    31
## 2 Alex          0             0    25     1009  2004     8     1
## 3 Alex          0             0    25     1009  2004     8     1
## 4 Alex          0             0    30     1009  2004     8     1
## 5 Alex         57.5           0    35     1009  2004     8     1
## 6 Alex         57.5           0    35     1007  2004     8     2
```

En este caso, como wind nunca pasa el valor 64, no hay valores distintos de cero en hu_diameter.

Filtrado de datos faltantes

Podemos usar nuevamente el comando filter para remover las filas con ts_diameter==0 o hu_diameter==0, para determinar los peores momentos de las tormentas:

```
SubTS=SubTS %>%
  filter(ts_diameter>0)
head(SubTS,n=3)
```

```
## # A tibble: 3 x 8
##   name  ts_diameter hu_diameter  wind pressure  year month   day
##   <chr>       <dbl>       <dbl> <int>    <int> <dbl> <dbl> <int>
## 1 Alex         57.5           0    35     1009  2004     8     1
## 2 Alex         57.5           0    35     1007  2004     8     2
## 3 Alex        173.            0    40     1005  2004     8     2
```

```
SubHU=SubHU %>%
  filter(hu_diameter>0)
head(SubHU,n=3)
```

```
## # A tibble: 3 x 8
##   name  ts_diameter hu_diameter  wind pressure  year month   day
##   <chr>       <dbl>       <dbl> <int>    <int> <dbl> <dbl> <int>
## 1 Alex         150.         46.0    70      983  2004     8     3
## 2 Alex         150.         46.0    85      974  2004     8     3
## 3 Alex         190.         57.5    85      972  2004     8     3
```

Filtrado de datos faltantes

Notar que por definición, ts_diameter es mayor que hu_diameter. Podríamos determinar qué proporción de las zonas afectadas fueron por viento huracanados realizando el cociente:

```
SubHU %>%
  select(-year,-month,-day) %>%
  mutate(propHU=hu_diameter/ts_diameter) %>%
  head()
```

```
## # A tibble: 6 x 6
##   name  ts_diameter hu_diameter  wind pressure propHU
##   <chr>       <dbl>       <dbl> <int>    <int>  <dbl>
## 1 Alex         150.        46.0    70      983  0.308
## 2 Alex         150.        46.0    85      974  0.308
## 3 Alex         190.        57.5    85      972  0.303
## 4 Alex         178.        63.3    80      974  0.355
## 5 Alex         224.        74.8    80      973  0.333
## 6 Alex         224.        74.8    85      973  0.333
```

Utilización de archivos externos

Casos COVID

Algunos datos sobre los testeos por COVID están disponibles en el archivo "casos_covid19.csv". Sin embargo, para facilitar el trabajo se tomaron las primeras 1000 observaciones en "sub_casos_covid19.csv". El archivo tiene la siguiente forma:

```
numero_de_caso,fecha_apertura_snvs,fecha_toma_muestra,fecha_clasificacion,provincia,barrio,comuna,genero,edad,clasificacion,fecha_fallecimiento,fallecido,fecha_alta,tipc
3665724,22OCT2020:00:00:00.000000,20OCT2020:00:00:00.000000,22OCT2020:00:00:00.000000,Buenos Aires,,,masculino,44,confirmado,,NA,,Comunitario
3665737,22OCT2020:00:00:00.000000,20OCT2020:00:00:00.000000,20OCT2020:00:00:00.000000,Buenos Aires,,,masculino,40,confirmado,,NA,,Comunitario
3665781,22OCT2020:00:00:00.000000,19OCT2020:00:00:00.000000,20OCT2020:00:00:00.000000,Buenos Aires,,,masculino,42,confirmado,,NA,,Comunitario
3665785,22OCT2020:00:00:00.000000,20OCT2020:00:00:00.000000,21OCT2020:00:00:00.000000,Buenos Aires,,,masculino,25,confirmado,,NA,,Comunitario
3665844,22OCT2020:00:00:00.000000,21OCT2020:00:00:00.000000,21OCT2020:00:00:00.000000,Buenos Aires,,,masculino,34,confirmado,,NA,,Comunitario
3665865,22OCT2020:00:00:00.000000,20OCT2020:00:00:00.000000,21OCT2020:00:00:00.000000,Buenos Aires,,,femenino,50,confirmado,,NA,,Comunitario
3665910,22OCT2020:00:00:00.000000,19OCT2020:00:00:00.000000,21OCT2020:00:00:00.000000,Buenos Aires,,,femenino,18,confirmado,,NA,,Comunitario
3665922,22OCT2020:00:00:00.000000,20OCT2020:00:00:00.000000,21OCT2020:00:00:00.000000,Buenos Aires,,,femenino,30,confirmado,,NA,,Comunitario
3665926,22OCT2020:00:00:00.000000,20OCT2020:00:00:00.000000,21OCT2020:00:00:00.000000,Buenos Aires,,,femenino,46,confirmado,,NA,,Comunitario
3665943,22OCT2020:00:00:00.000000,20OCT2020:00:00:00.000000,22OCT2020:00:00:00.000000,Buenos Aires,,,masculino,49,confirmado,,NA,,Comunitario
3665979,22OCT2020:00:00:00.000000,21OCT2020:00:00:00.000000,21OCT2020:00:00:00.000000,CABA,,,masculino,21,confirmado,,NA,,En Investigación
3665991,22OCT2020:00:00:00.000000,22OCT2020:00:00:00.000000,22OCT2020:00:00:00.000000,CABA,ALMAGRO,5,masculino,68,confirmado,,NA,,Comunitario
3666014,22OCT2020:00:00:00.000000,20OCT2020:00:00:00.000000,21OCT2020:00:00:00.000000,CABA,VILLA LUGANO,8,masculino,49,confirmado,,NA,,Comunitario
3666087,22OCT2020:00:00:00.000000,20OCT2020:00:00:00.000000,21OCT2020:00:00:00.000000,Buenos Aires,,,masculino,44,confirmado,,NA,,Comunitario
3666170,22OCT2020:00:00:00.000000,20OCT2020:00:00:00.000000,22OCT2020:00:00:00.000000,Buenos Aires,,,femenino,45,confirmado,,NA,,Comunitario
3666239,22OCT2020:00:00:00.000000,20OCT2020:00:00:00.000000,21OCT2020:00:00:00.000000,Buenos Aires,,,femenino,41,confirmado,,NA,,Comunitario
3666283,22OCT2020:00:00:00.000000,20OCT2020:00:00:00.000000,20OCT2020:00:00:00.000000,Córdoba,,,masculino,29,confirmado,,NA,,Comunitario
3666311,22OCT2020:00:00:00.000000,21OCT2020:00:00:00.000000,21OCT2020:00:00:00.000000,Buenos Aires,,,masculino,57,confirmado,,NA,,Comunitario
3666355,22OCT2020:00:00:00.000000,20OCT2020:00:00:00.000000,21OCT2020:00:00:00.000000,Buenos Aires,,,masculino,19,confirmado,,NA,,Comunitario
3666423,22OCT2020:00:00:00.000000,21OCT2020:00:00:00.000000,21OCT2020:00:00:00.000000,CABA,BALVANERA,3,femenino,29,confirmado,,NA,,Comunitario
3666477,22OCT2020:00:00:00.000000,20OCT2020:00:00:00.000000,21OCT2020:00:00:00.000000,Buenos Aires,,,femenino,25,confirmado,,NA,,Comunitario
3666486,22OCT2020:00:00:00.000000,20OCT2020:00:00:00.000000,21OCT2020:00:00:00.000000,Buenos Aires,,,masculino,30,confirmado,,NA,,Comunitario
3666600,22OCT2020:00:00:00.000000,20OCT2020:00:00:00.000000,21OCT2020:00:00:00.000000,CABA,BELGRANO,13,femenino,86,confirmado,,NA,,Comunitario
3666742,22OCT2020:00:00:00.000000,20OCT2020:00:00:00.000000,21OCT2020:00:00:00.000000,CABA,PARQUE PATRICIOS,4,femenino,10,confirmado,,NA,,Comunitario
3666770,22OCT2020:00:00:00.000000,20OCT2020:00:00:00.000000,20OCT2020:00:00:00.000000,Buenos Aires,,,masculino,56,confirmado,,NA,,Comunitario
3693033,22OCT2020:00:00:00.000000,22OCT2020:00:00:00.000000,23OCT2020:00:00:00.000000,Buenos Aires,,,femenino,33,confirmado,,NA,,Trabajador de la Salud
3693185,22OCT2020:00:00:00.000000,27OCT2020:00:00:00.000000,27OCT2020:00:00:00.000000,CABA,VILLA DEL PARQUE,11,masculino,12,confirmado,,NA,,Comunitario
3693425,22OCT2020:00:00:00.000000,20OCT2020:00:00:00.000000,22OCT2020:00:00:00.000000,Buenos Aires,,,femenino,31,confirmado,,NA,,Comunitario
3693446,22OCT2020:00:00:00.000000,21OCT2020:00:00:00.000000,21OCT2020:00:00:00.000000,CABA,BELGRANO,13,masculino,11,confirmado,,NA,,Comunitario
3693472,22OCT2020:00:00:00.000000,22OCT2020:00:00:00.000000,23OCT2020:00:00:00.000000,Buenos Aires,,,masculino,64,confirmado,,NA,20OCT2020:00:00:00.000000,Contacto
3693588,22OCT2020:00:00:00.000000,07NOV2020:00:00:00.000000,07NOV2020:00:00:00.000000,Buenos Aires,,,femenino,35,confirmado,,NA,,Comunitario
3693640,22OCT2020:00:00:00.000000,22OCT2020:00:00:00.000000,23OCT2020:00:00:00.000000,CABA,,,femenino,46,confirmado,,NA,,Trabajador de la Salud
3693752,22OCT2020:00:00:00.000000,20NOV2020:00:00:00.000000,20NOV2020:00:00:00.000000,Buenos Aires,,,masculino,43,confirmado,,NA,,Comunitario
3693773,22OCT2020:00:00:00.000000,11NOV2020:00:00:00.000000,12NOV2020:00:00:00.000000,Buenos Aires,,,masculino,41,confirmado,,NA,,Comunitario
```

Las observaciones están separadas por comas, ya que csv viene de "comma separated values".

Directorio de trabajo

- Cuando queremos cargar el archivo con el comando read.table, nos salta un error:

```
DatCov=read.table("sub_casos_covid19.csv")
```

```
## Error in file(file, "rt"): cannot open the connection
```

Directorio de trabajo

- Cuando queremos cargar el archivo con el comando read.table, nos salta un error:

```
DatCov=read.table("sub_casos_covid19.csv")
```

```
## Error in file(file, "rt"): cannot open the connection
```

- Esto se debe a que el directorio de trabajo (wd: working directory) no contiene el archivo, para obtener el directorio de trabajo, se usa el comando getwd():

```
getwd()
```

```
## [1] "/home/lucio/Documents/ITBA/Taller R/Presentaciones/Tercera Clase"
```

Directorio de trabajo

- setwd cambia el directorio de trabajo:

```
setwd("/home/lucio/Documents/ITBA/Taller R/Presentaciones/Tercera Clase/Dato
```

```
DatCov=read.table("sub_casos_covid19.csv")
head(DatCov)
```

```
##
## 1 numero_de_caso,fecha_apertura_snvs,fecha_toma_muestra,fecha_clasificac
## 2                                      3665724,22OCT2020:00:00:00.000000
## 3                                      3665737,22OCT2020:00:00:00.000000
## 4                                      3665781,22OCT2020:00:00:00.000000
## 5                                      3665785,22OCT2020:00:00:00.000000
## 6                                      3665844,22OCT2020:00:00:00.000000
```

Directorio de trabajo

- setwd cambia el directorio de trabajo:

```
setwd("/home/lucio/Documents/ITBA/Taller R/Presentaciones/Tercera Clase/Dato
```

```
DatCov=read.table("sub_casos_covid19.csv")
head(DatCov)
```

```
##
## 1 numero_de_caso,fecha_apertura_snvs,fecha_toma_muestra,fecha_clasificac
## 2                                      3665724,22OCT2020:00:00:00.000000
## 3                                      3665737,22OCT2020:00:00:00.000000
## 4                                      3665781,22OCT2020:00:00:00.000000
## 5                                      3665785,22OCT2020:00:00:00.000000
## 6                                      3665844,22OCT2020:00:00:00.000000
```

Notar que quedó un espanto, esto se debe a que no se consideró correctamente la separación entre columnas. Esto se puede ver calculando la cantidad de columnas:

```
ncol(DatCov)
```

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

Leyendo archivos

Vamos a añadir un parámetro que permita separar por comas cada columna:

```
DatCov=read.table("sub_casos_covid19.csv",sep=",")
head(DatCov[,1:9])
```

```
##                V1                             V2                             V3
## 1 numero_de_caso           fecha_apertura_snvs             fecha_toma_muestra
## 2         3665724 22OCT2020:00:00:00.000000 20OCT2020:00:00:00.000000
## 3         3665737 22OCT2020:00:00:00.000000 20OCT2020:00:00:00.000000
## 4         3665781 22OCT2020:00:00:00.000000 19OCT2020:00:00:00.000000
## 5         3665785 22OCT2020:00:00:00.000000 20OCT2020:00:00:00.000000
## 6         3665844 22OCT2020:00:00:00.000000 21OCT2020:00:00:00.000000
##                             V4           V5     V6     V7           V8   V9
## 1        fecha_clasificacion     provincia barrio comuna     genero edad
## 2 22OCT2020:00:00:00.000000 Buenos Aires                masculino   44
## 3 20OCT2020:00:00:00.000000 Buenos Aires                masculino   40
## 4 20OCT2020:00:00:00.000000 Buenos Aires                masculino   42
## 5 21OCT2020:00:00:00.000000 Buenos Aires                masculino   25
## 6 21OCT2020:00:00:00.000000 Buenos Aires                masculino   34
```

Leyendo archivos

Notar que vuelve a haber algo raro, fijémosnos la estructura de la base:

```
str(DatCov[,1:9])
```

```
## 'data.frame':    999 obs. of  9 variables:
##  $ V1: Factor w/ 999 levels "3255754","3270806",..: 999 83 84 85 86 87 88
##  $ V2: Factor w/ 27 levels "01NOV2020:00:00:00.000000",..: 27 17 17 17 17
##  $ V3: Factor w/ 58 levels "01DEC2020:00:00:00.000000",..: 58 37 37 35 37
##  $ V4: Factor w/ 59 levels "01DEC2020:00:00:00.000000",..: 59 41 39 39 40
##  $ V5: Factor w/ 10 levels "Buenos Aires",..: 7 1 1 1 1 1 1 1 1 1 ...
##  $ V6: Factor w/ 48 levels "","AGRONOMIA",..: 6 1 1 1 1 1 1 1 1 1 ...
##  $ V7: Factor w/ 17 levels "","1","10","11",..: 17 1 1 1 1 1 1 1 1 1 ...
##  $ V8: Factor w/ 3 levels "femenino","genero",..: 2 3 3 3 3 3 1 1 1 1 ...
##  $ V9: Factor w/ 91 levels "1","10","11",..: 91 40 36 38 19 29 47 11 25 4
```

Leyendo archivos

Notar que vuelve a haber algo raro, fijémosnos la estructura de la base:

```
str(DatCov[,1:9])
```

```
## 'data.frame':    999 obs. of  9 variables:
##  $ V1: Factor w/ 999 levels "3255754","3270806",..: 999 83 84 85 86 87 88
##  $ V2: Factor w/ 27 levels "01NOV2020:00:00:00.000000",..: 27 17 17 17 17
##  $ V3: Factor w/ 58 levels "01DEC2020:00:00:00.000000",..: 58 37 37 35 37
##  $ V4: Factor w/ 59 levels "01DEC2020:00:00:00.000000",..: 59 41 39 39 40
##  $ V5: Factor w/ 10 levels "Buenos Aires",..: 7 1 1 1 1 1 1 1 1 1 ...
##  $ V6: Factor w/ 48 levels "","AGRONOMIA",..: 6 1 1 1 1 1 1 1 1 1 ...
##  $ V7: Factor w/ 17 levels "","1","10","11",..: 17 1 1 1 1 1 1 1 1 1 ...
##  $ V8: Factor w/ 3 levels "femenino","genero",..: 2 3 3 3 3 3 1 1 1 1 ...
##  $ V9: Factor w/ 91 levels "1","10","11",..: 91 40 36 38 19 29 47 11 25 4
```

Son todos factores, y además, las variables se llaman V1, V2, etc.

Leyendo archivos

Agregamos dos argumentos: header=T y stringsAsFactors=F. Esto da la base que queríamos:

```
DatCov=read.table("sub_casos_covid19.csv",header = T,sep=",",stringsAsFactor
head(DatCov[,1:9])
```

```
##   numero_de_caso       fecha_apertura_snvs        fecha_toma_muestra
## 1        3665724 22OCT2020:00:00:00.000000 20OCT2020:00:00:00.000000
## 2        3665737 22OCT2020:00:00:00.000000 20OCT2020:00:00:00.000000
## 3        3665781 22OCT2020:00:00:00.000000 19OCT2020:00:00:00.000000
## 4        3665785 22OCT2020:00:00:00.000000 20OCT2020:00:00:00.000000
## 5        3665844 22OCT2020:00:00:00.000000 21OCT2020:00:00:00.000000
## 6        3665865 22OCT2020:00:00:00.000000 20OCT2020:00:00:00.000000
##         fecha_clasificacion    provincia barrio comuna    genero edad
## 1 22OCT2020:00:00:00.000000 Buenos Aires             NA masculino   44
## 2 20OCT2020:00:00:00.000000 Buenos Aires             NA masculino   40
## 3 20OCT2020:00:00:00.000000 Buenos Aires             NA masculino   42
## 4 21OCT2020:00:00:00.000000 Buenos Aires             NA masculino   25
## 5 21OCT2020:00:00:00.000000 Buenos Aires             NA masculino   34
## 6 21OCT2020:00:00:00.000000 Buenos Aires             NA  femenino   50
```

Leyendo archivos

Agregamos dos argumentos: header=T y stringsAsFactors=F. Esto da la base que queríamos:

```
str(DatCov[,1:9])
```

```
## 'data.frame':    998 obs. of  9 variables:
##  $ numero_de_caso     : int  3665724 3665737 3665781 3665785 3665844 3665
##  $ fecha_apertura_snvs: chr  "22OCT2020:00:00:00.000000" "22OCT2020:00:0
##  $ fecha_toma_muestra : chr  "20OCT2020:00:00:00.000000" "20OCT2020:00:0
##  $ fecha_clasificacion: chr  "22OCT2020:00:00:00.000000" "20OCT2020:00:0
##  $ provincia          : chr  "Buenos Aires" "Buenos Aires" "Buenos Aires"
##  $ barrio             : chr  "" "" "" "" ...
##  $ comuna             : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ genero             : chr  "masculino" "masculino" "masculino" "mascul
##  $ edad               : int  44 40 42 25 34 50 18 30 46 49 ...
```

Leyendo archivos

De todos modos, el comando read.csv ya tiene estos valores incorporados como default (salvo stringsAsFactors):

```
DatCov=read.csv("sub_casos_covid19.csv",stringsAsFactors = F)
head(DatCov[,1:9])
```

```
##   numero_de_caso       fecha_apertura_snvs        fecha_toma_muestra
## 1        3665724 22OCT2020:00:00:00.000000 20OCT2020:00:00:00.000000
## 2        3665737 22OCT2020:00:00:00.000000 20OCT2020:00:00:00.000000
## 3        3665781 22OCT2020:00:00:00.000000 19OCT2020:00:00:00.000000
## 4        3665785 22OCT2020:00:00:00.000000 20OCT2020:00:00:00.000000
## 5        3665844 22OCT2020:00:00:00.000000 21OCT2020:00:00:00.000000
## 6        3665865 22OCT2020:00:00:00.000000 20OCT2020:00:00:00.000000
##         fecha_clasificacion    provincia barrio comuna    genero edad
## 1 22OCT2020:00:00:00.000000 Buenos Aires             NA masculino   44
## 2 20OCT2020:00:00:00.000000 Buenos Aires             NA masculino   40
## 3 20OCT2020:00:00:00.000000 Buenos Aires             NA masculino   42
## 4 21OCT2020:00:00:00.000000 Buenos Aires             NA masculino   25
## 5 21OCT2020:00:00:00.000000 Buenos Aires             NA masculino   34
## 6 21OCT2020:00:00:00.000000 Buenos Aires             NA  femenino   50
```

Leyendo scripts

Supongamos que tenemos la siguiente función definida en otro script llamado "Script_Auxiliar.R". Esta función cambia (mutate) las variables que tienen el string "fecha" en su nombre, aplicándoles dmy_hms:

```
ModFechas=function(Base){
  ModBase=Base %>%
    mutate_at(vars(contains("fecha")),dmy_hms)
  return(ModBase)
}
```

Leyendo scripts

Si queremos llamar la función sin correr el script externo, nos tira un error:

```
ModDatCov=ModFechas(DatCov)
```

```
## Error in ModFechas(DatCov): could not find function "ModFechas"
```

Leyendo scripts

Si queremos llamar la función sin correr el script externo, nos tira un error:

```
ModDatCov=ModFechas(DatCov)
```

```
## Error in ModFechas(DatCov): could not find function "ModFechas"
```

Para leer un script externo, se usa el comando source:

```
source("Script_Auxiliar.R")
ModDatCov=ModFechas(DatCov)
head(ModDatCov[,1:9],n=3)
```

```
##   numero_de_caso fecha_apertura_snvs fecha_toma_muestra fecha_clasificac
## 1        3665724          2020-10-22          2020-10-20          2020-10-
## 2        3665737          2020-10-22          2020-10-20          2020-10-
## 3        3665781          2020-10-22          2020-10-19          2020-10-
##      provincia barrio comuna    genero edad
## 1 Buenos Aires             NA masculino   44
## 2 Buenos Aires             NA masculino   40
## 3 Buenos Aires             NA masculino   42
```

Leyendo scripts

Por otro lado, notemos que la base cambió sus variables fechas a tipo POSIXct, que es el tipo de datos con fechas, horas, minutos y segundos.

```
str(ModDatCov)
```

```
## 'data.frame':    998 obs. of  14 variables:
##  $ numero_de_caso     : int  3665724 3665737 3665781 3665785 3665844 3665
##  $ fecha_apertura_snvs: POSIXct, format: "2020-10-22" "2020-10-22" ...
##  $ fecha_toma_muestra : POSIXct, format: "2020-10-20" "2020-10-20" ...
##  $ fecha_clasificacion: POSIXct, format: "2020-10-22" "2020-10-20" ...
##  $ provincia          : chr  "Buenos Aires" "Buenos Aires" "Buenos Aires"
##  $ barrio             : chr  "" "" "" "" ...
##  $ comuna             : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ genero             : chr  "masculino" "masculino" "masculino" "mascul
##  $ edad               : int  44 40 42 25 34 50 18 30 46 49 ...
##  $ clasificacion      : chr  "confirmado" "confirmado" "confirmado" "con
##  $ fecha_fallecimiento: POSIXct, format: NA NA ...
##  $ fallecido          : logi  NA NA NA NA NA NA ...
##  $ fecha_alta         : POSIXct, format: NA NA ...
##  $ tipo_contagio      : chr  "Comunitario" "Comunitario" "Comunitario" "C
```

Escribiendo tablas

Podemos escribir una tabla con la cantidad de cada tipo de contagio usando los comandos del dplyr:

```
ResCov=DatCov %>%
  group_by(tipo_contagio) %>%
  summarise(Freq=n())
ResCov
```

```
## # A tibble: 5 x 2
##   tipo_contagio           Freq
##   <chr>                  <int>
## 1 Comunitario              803
## 2 Contacto                  96
## 3 En Investigación          60
## 4 Trabajador de la Salud    38
## 5 <NA>                       1
```

Escribiendo tablas

Por ejemplo, para escribirlo en formato de tablas de LaTeX, las separaciones son con "&" y el final de línea (eol: end of line) es con doble contrabarra, dos veces, o sea 4 contrabarras, ya que cada doble contrabarra, se interpreta en caracteres como una sola:

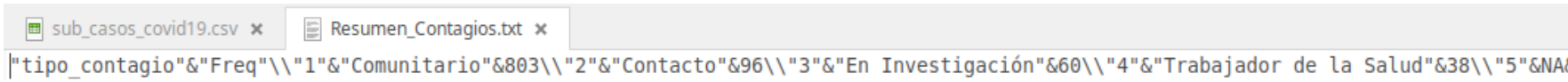
```
write.table(ResCov,file="Resumen_Contagios.txt",
            sep="&",eol="\\\\")
```

Escribiendo tablas

Por ejemplo, para escribirlo en formato de tablas de LaTeX, las separaciones son con "&" y el final de línea (eol: end of line) es con doble contrabarra, dos veces, o sea 4 contrabarras, ya que cada doble contrabarra, se interpreta en caracteres como una sola:

```
write.table(ResCov,file="Resumen_Contagios.txt",
            sep="&",eol="\\\\")
```

El resultado:



Escribiendo tablas

Notar que las variables aparecen con comillas, que no termina de cambiar las filas en el output, y que tienen un número inicial que indica cada fila. Eso se resuelve con quote=F, agregando "contrabarra+n" al final de eol, y con row.names=F, respectivamente:

```
write.table(ResCov,file="Resumen_Contagios.txt",
            sep="&",eol="\\\\\n",quote=F,row.names = F)
```

Escribiendo tablas

Notar que las variables aparecen con comillas, que no termina de cambiar las filas en el output, y que tienen un número inicial que indica cada fila. Eso se resuelve con quote=F, agregando "contrabarra+n" al final de eol, y con row.names=F, respectivamente:

```
write.table(ResCov,file="Resumen_Contagios.txt",
            sep="&",eol="\\\\\n",quote=F,row.names = F)
```

El resultado:

```
tipo_contagio&Freq\\
Comunitario&803\\
Contacto&96\\
En Investigación&60\\
Trabajador de la Salud&38\\
NA&1\\
```

Starwars

Base starwars

dplyr tiene una base con características de los personajes de starwars. Ñoñada si las hay, pero vamos a usarla.

```
head(starwars[,1:6])
```

```
## # A tibble: 6 x 6
##   name           height  mass hair_color  skin_color  eye_color
##   <chr>           <int> <dbl> <chr>       <chr>       <chr>
## 1 Luke Skywalker    172    77 blond       fair        blue
## 2 C-3PO             167    75 <NA>        gold        yellow
## 3 R2-D2              96    32 <NA>        white, blue red
## 4 Darth Vader       202   136 none        white       yellow
## 5 Leia Organa       150    49 brown       light       brown
## 6 Owen Lars         178   120 brown, grey light       blue
```

Base starwars

Las últimas dos columnas son listas, que describen las películas en las que estuvieron y los vehículos que manejaron. Son listas dado que no siempre son la misma cantidad de elementos por observación.

```
starwars$name[4:5]
```

```
## [1] "Darth Vader" "Leia Organa"
```

```
starwars$films[4:5]
```

```
## [[1]]
## [1] "The Empire Strikes Back" "Revenge of the Sith"
## [3] "Return of the Jedi"      "A New Hope"
##
## [[2]]
## [1] "The Empire Strikes Back" "Revenge of the Sith"
## [3] "Return of the Jedi"      "A New Hope"
## [5] "The Force Awakens"
```

Base starwars

Las últimas dos columnas son listas, que describen las películas en las que estuvieron y los vehículos que manejaron. Son listas dado que no siempre son la misma cantidad de elementos por observación.

```
starwars$name[c(1,5)]
```

```
## [1] "Luke Skywalker" "Leia Organa"
```

```
starwars$vehicles[c(1,5)]
```

```
## [[1]]
## [1] "Snowspeeder"           "Imperial Speeder Bike"
##
## [[2]]
## [1] "Imperial Speeder Bike"
```

Base starwars

Para que la base no tenga estas listas dispares, vamos a modificar la base agregandole la cantidad de películas en las que estuvo cada personaje (NFilms) y la cantidad de vehículos que manejó (NVehi):

```
ModStarWars=starwars %>%
  select(-films,-vehicles) %>%
  mutate(NFilms=sapply(starwars$films,length),
         NVehi=sapply(starwars$vehicles,length))
ModStarWars %>%
  select(name,height,mass,NFilms,NVehi) %>%
  head()
```

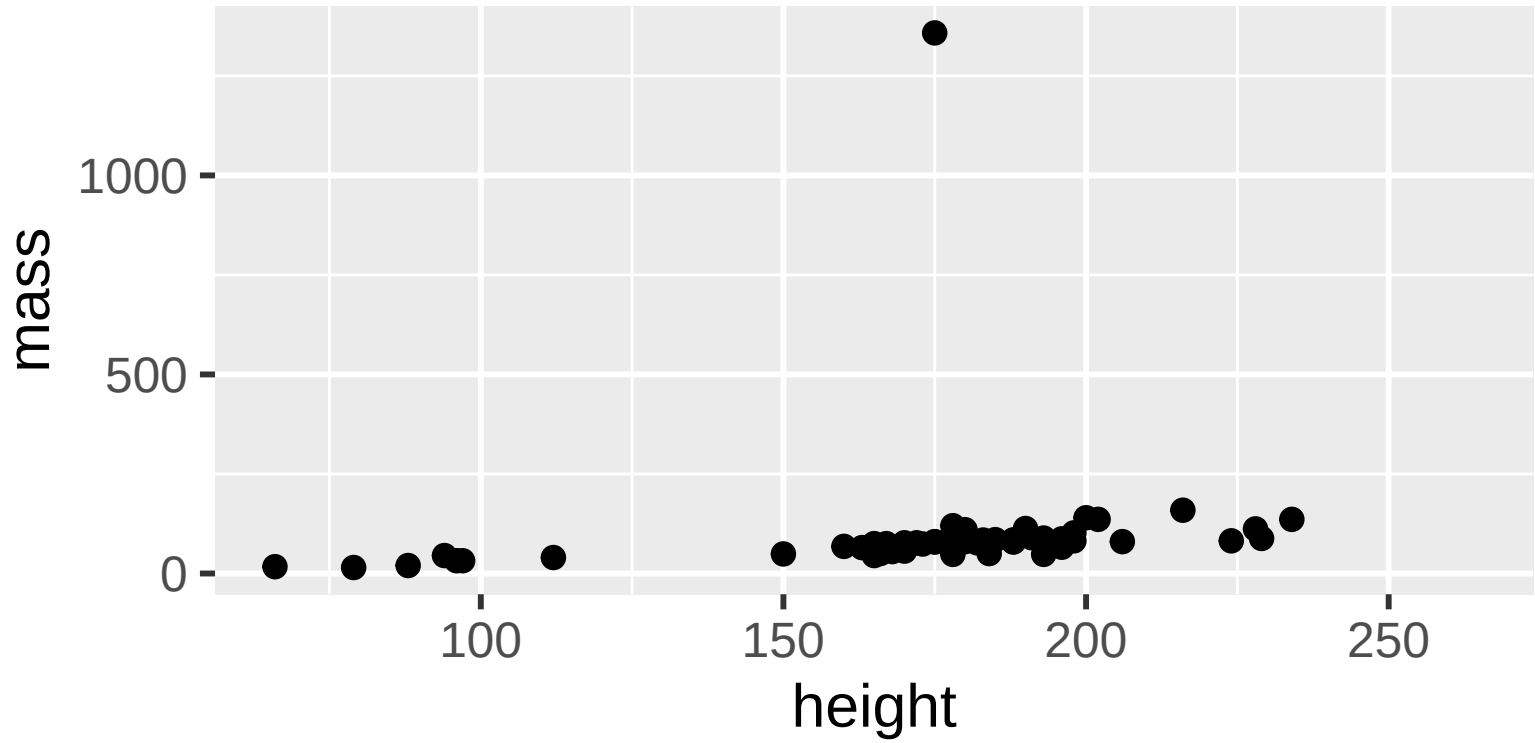
```
## # A tibble: 6 x 5
##   name           height  mass NFilms NVehi
##   <chr>           <int> <dbl>  <int> <int>
## 1 Luke Skywalker    172    77      5     2
## 2 C-3PO             167    75      6     0
## 3 R2-D2              96    32      7     0
## 4 Darth Vader       202   136      4     0
## 5 Leia Organa       150    49      5     1
## 6 Owen Lars         178   120      3     0
```

Base starwars

Una cosa que podríamos ver es cómo se relacionan la masa con la altura de los personajes:

```
ModStarWars %>%
  ggplot(aes(x=height,y=mass))+
  geom_point()
```

```
## Warning: Removed 28 rows containing missing values (geom_point).
```

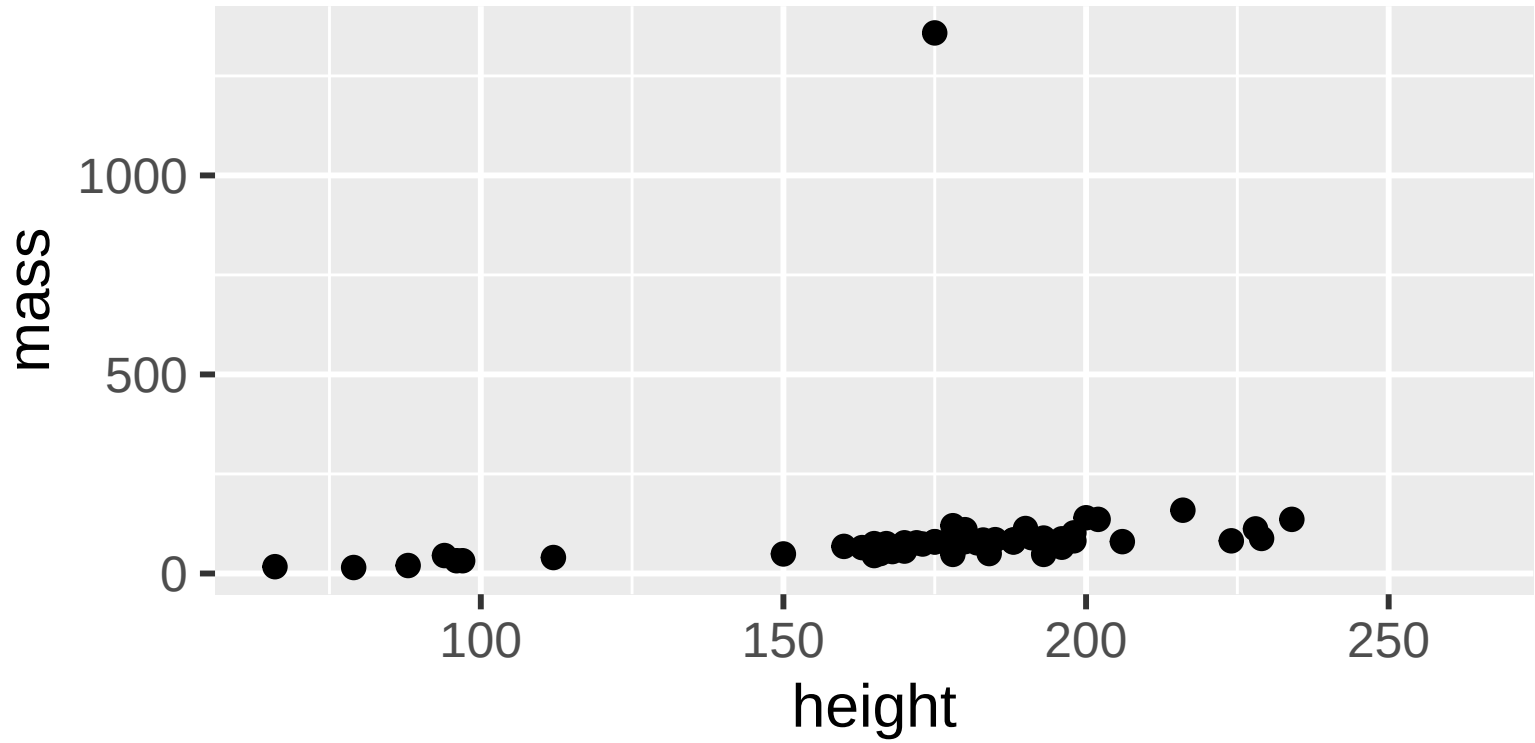


Base starwars

Una cosa que podríamos ver es cómo se relacionan la masa con la altura de los personajes:

```
ModStarWars %>%
  ggplot(aes(x=height,y=mass))+
  geom_point()
```

```
## Warning: Removed 28 rows containing missing values (geom_point).
```



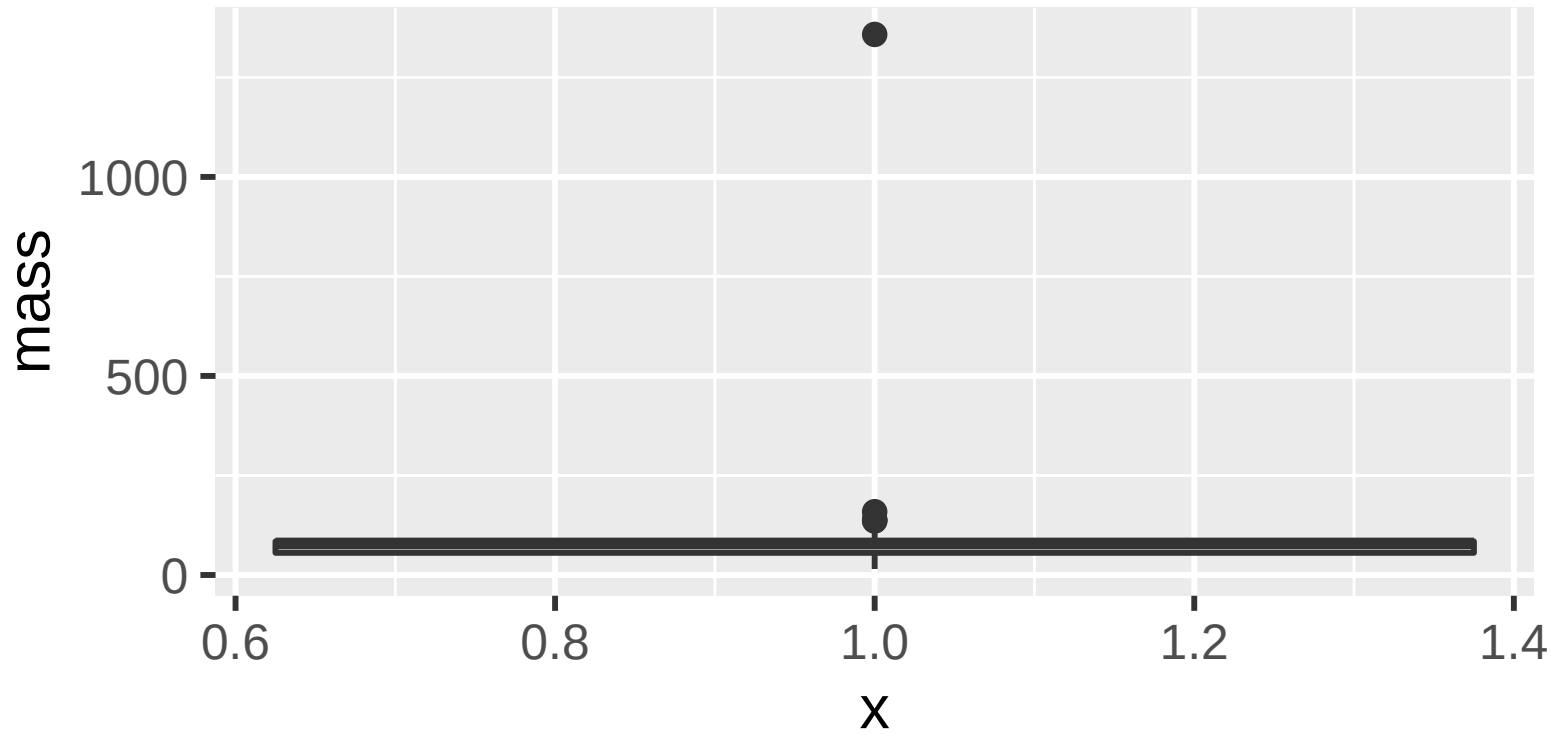
Notemos que hay un punto muuuuuuy alejado del resto

Base starwars

Más aún, claramente es un outlier y lo muestra el boxplot:

```
ModStarWars %>%
  ggplot(aes(x=1,y=mass))+
  geom_boxplot()
```

```
## Warning: Removed 28 rows containing non-finite values (stat_boxplot).
```



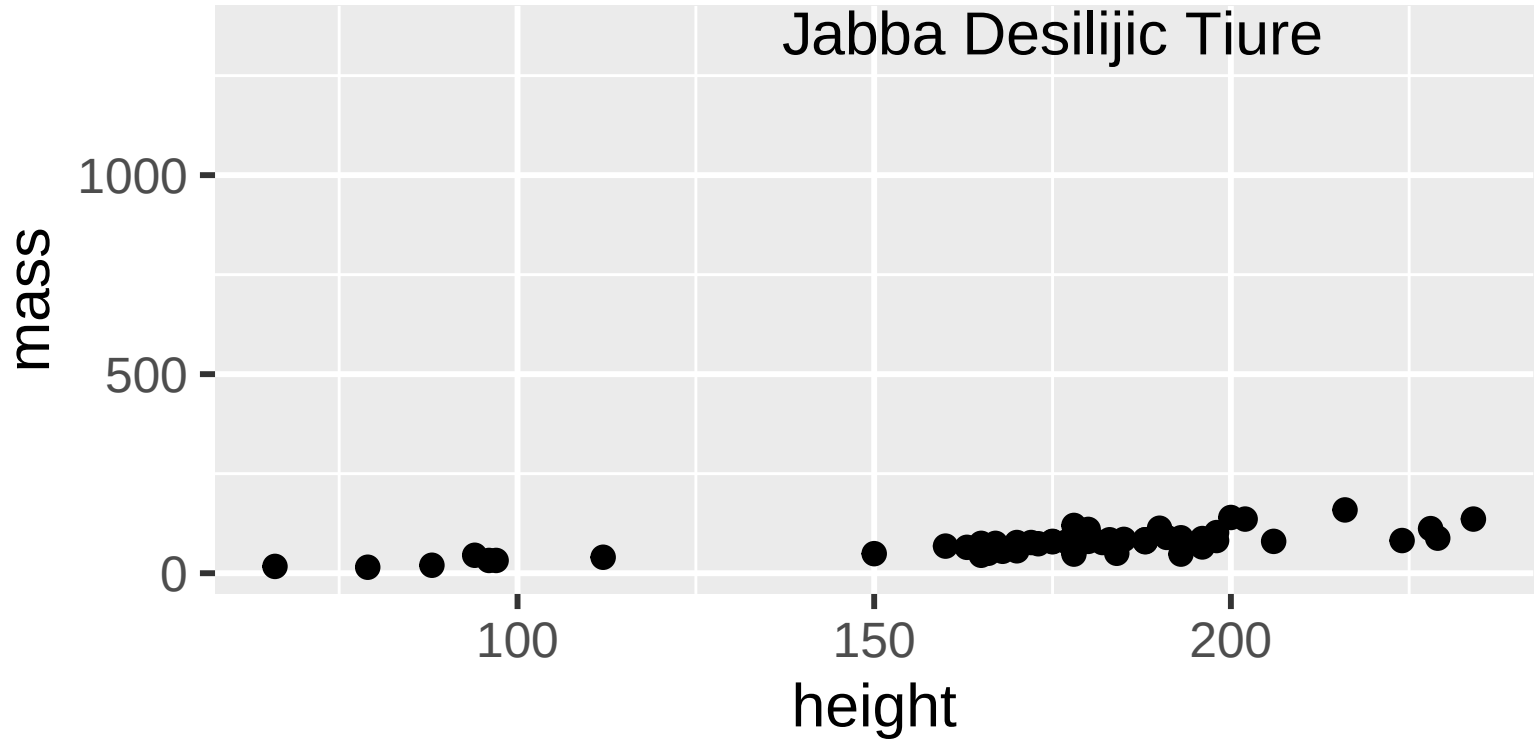
Base starwars

Ahí es cuando uno recuerda a Jabba the Hutt:

Base starwars

De hecho:

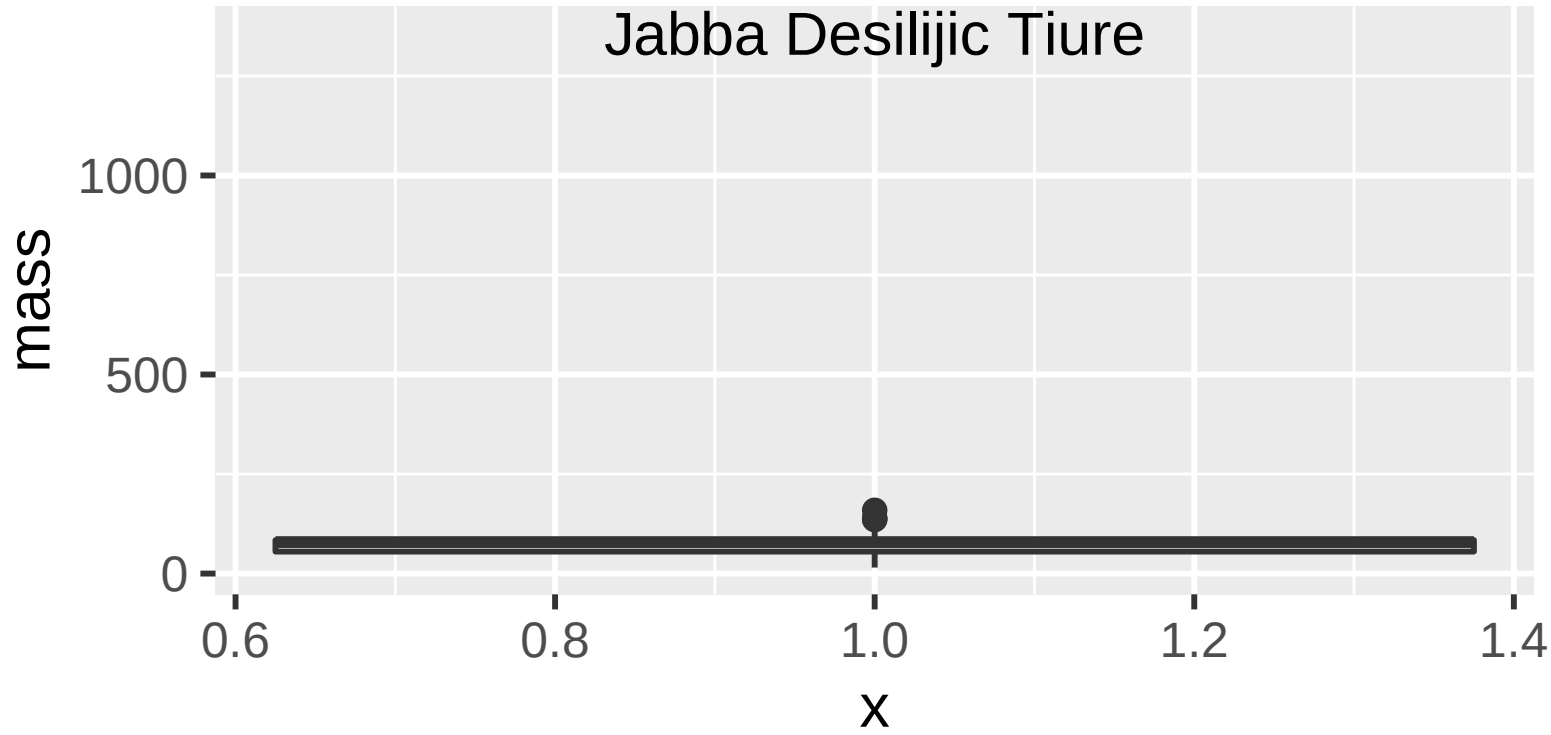
```
ModStarWars %>%
  filter(mass<1000) %>%
  ggplot(aes(x=height,y=mass))+
  geom_point()+
  ModStarWars %>%
  filter(mass>1000) %>%
  geom_text(data=.,mapping=aes(x=height,y=mass,label=name))
```



Base starwars

De hecho:

```
ModStarWars %>%
  filter(mass<1000) %>%
  ggplot(aes(x=1,y=mass))+
  geom_boxplot()+
  ModStarWars %>%
  filter(mass>1000) %>%
  geom_text(data=.,mapping=aes(x=1,y=mass,label=name))
```



Base starwars

Ya que estamos, vamos a averiguar que especie y de qué planeta es Jabba:

```
ModStarWars %>%
  filter(str_detect(name,"Jabba")) %>%
  select(name,height,mass,homeworld,species)
```

```
## # A tibble: 1 x 5
##   name                  height  mass homeworld species
##   <chr>                  <int> <dbl> <chr>     <chr>
## 1 Jabba Desilijic Tiure    175  1358 Nal Hutta Hutt
```

Base starwars

Ya que estamos, vamos a averiguar que especie y de qué planeta es Jabba:

```
ModStarWars %>%
  filter(str_detect(name,"Jabba")) %>%
  select(name,height,mass,homeworld,species)
```

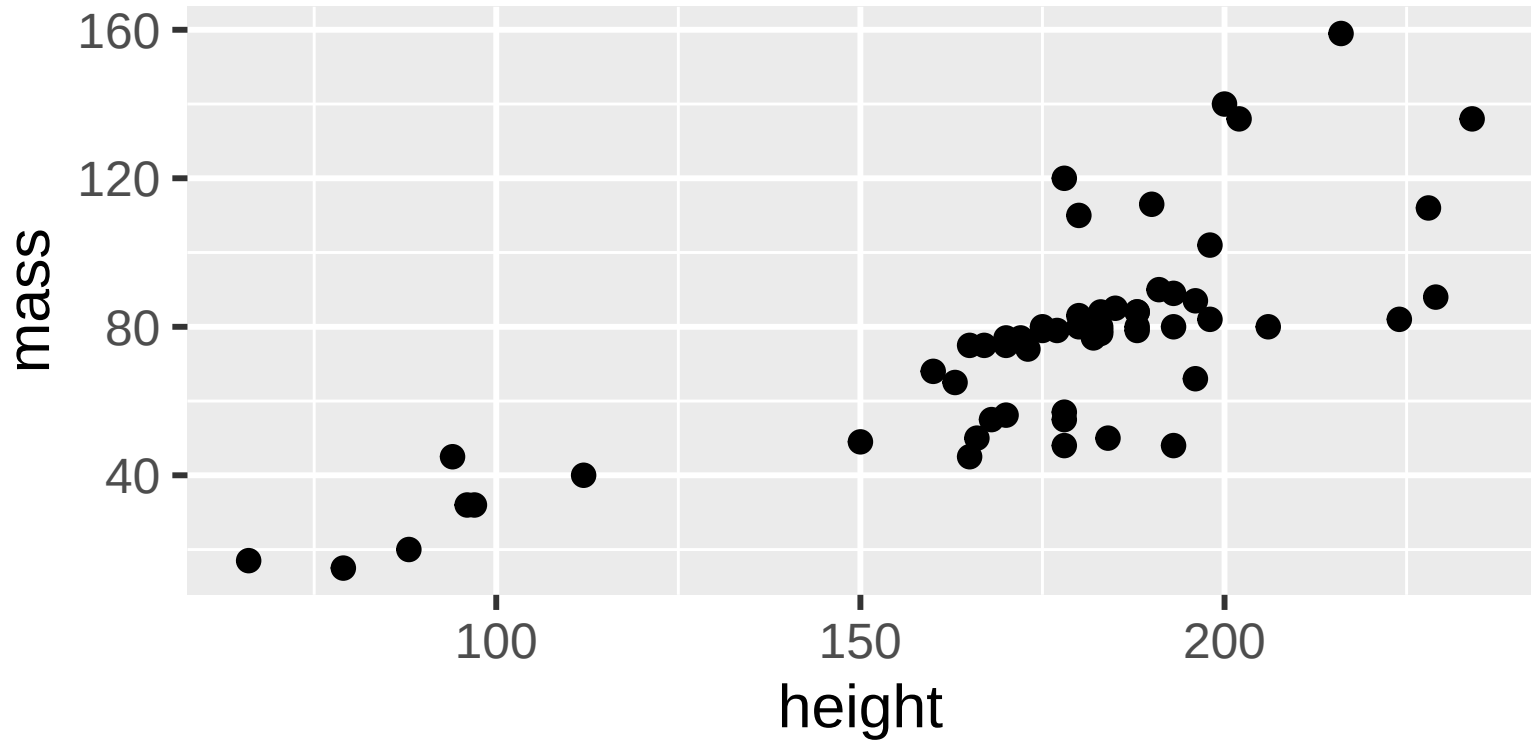
```
## # A tibble: 1 x 5
##   name                  height  mass homeworld species
##   <chr>                  <int> <dbl> <chr>     <chr>
## 1 Jabba Desilijic Tiure    175  1358 Nal Hutta Hutt
```

Shockeado, Jabba the Hutt es un Hutt, de un planeta que se llama Hutta.

Base starwars

Vamos a excluir a Jabba del análisis y ver cómo quedan los datos de altura y masa:

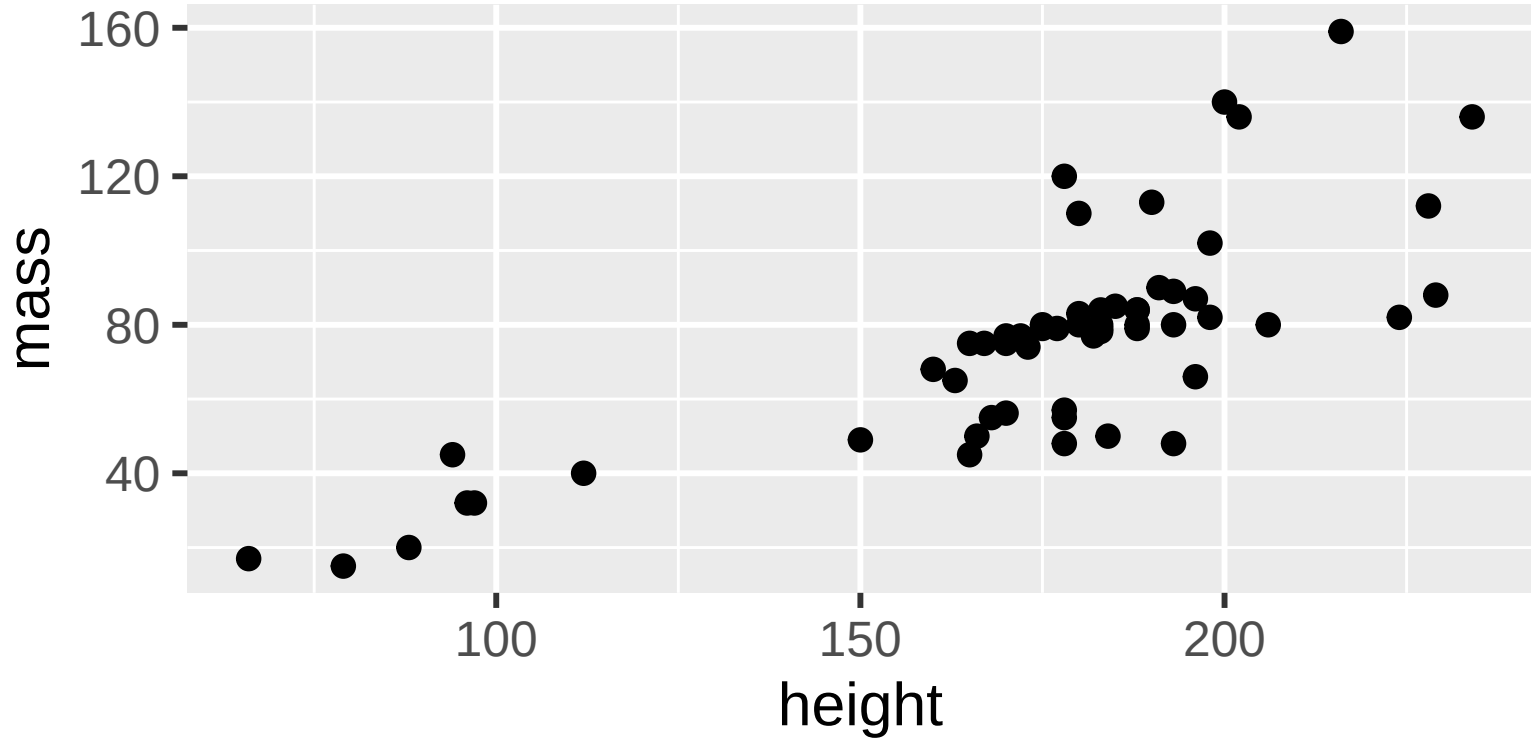
```
ModStarWars %>%
  filter(mass<1000) %>%
  ggplot(aes(x=height,y=mass))+
  geom_point()
```



Base starwars

Vamos a excluir a Jabba del análisis y ver cómo quedan los datos de altura y masa:

```
ModStarWars %>%
  filter(mass<1000) %>%
  ggplot(aes(x=height,y=mass))+
  geom_point()
```

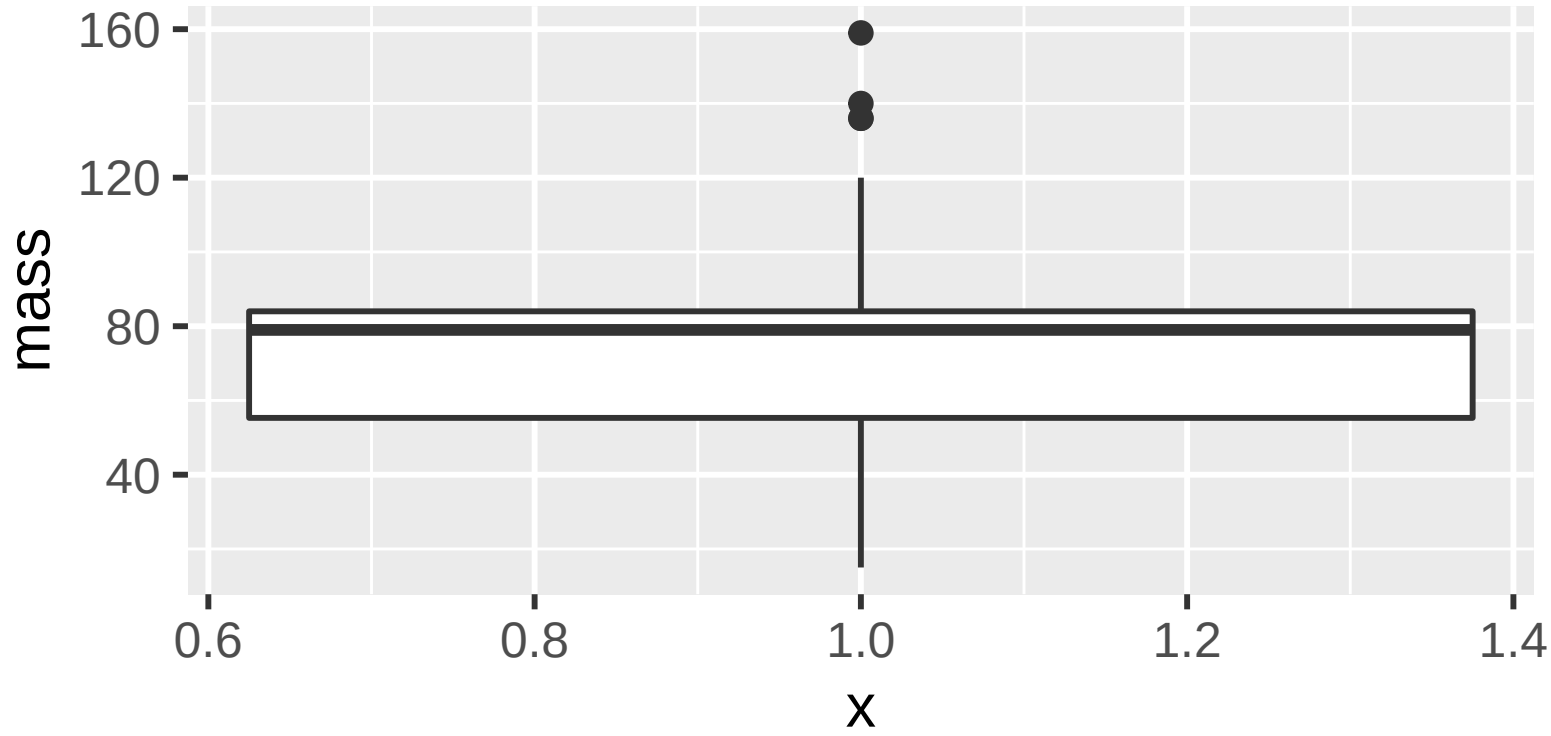


Notar la influencia gráfica que puede tener un outlier.

Base starwars

Vamos a excluir a Jabba del análisis y ver cómo queda el boxplot de masa:

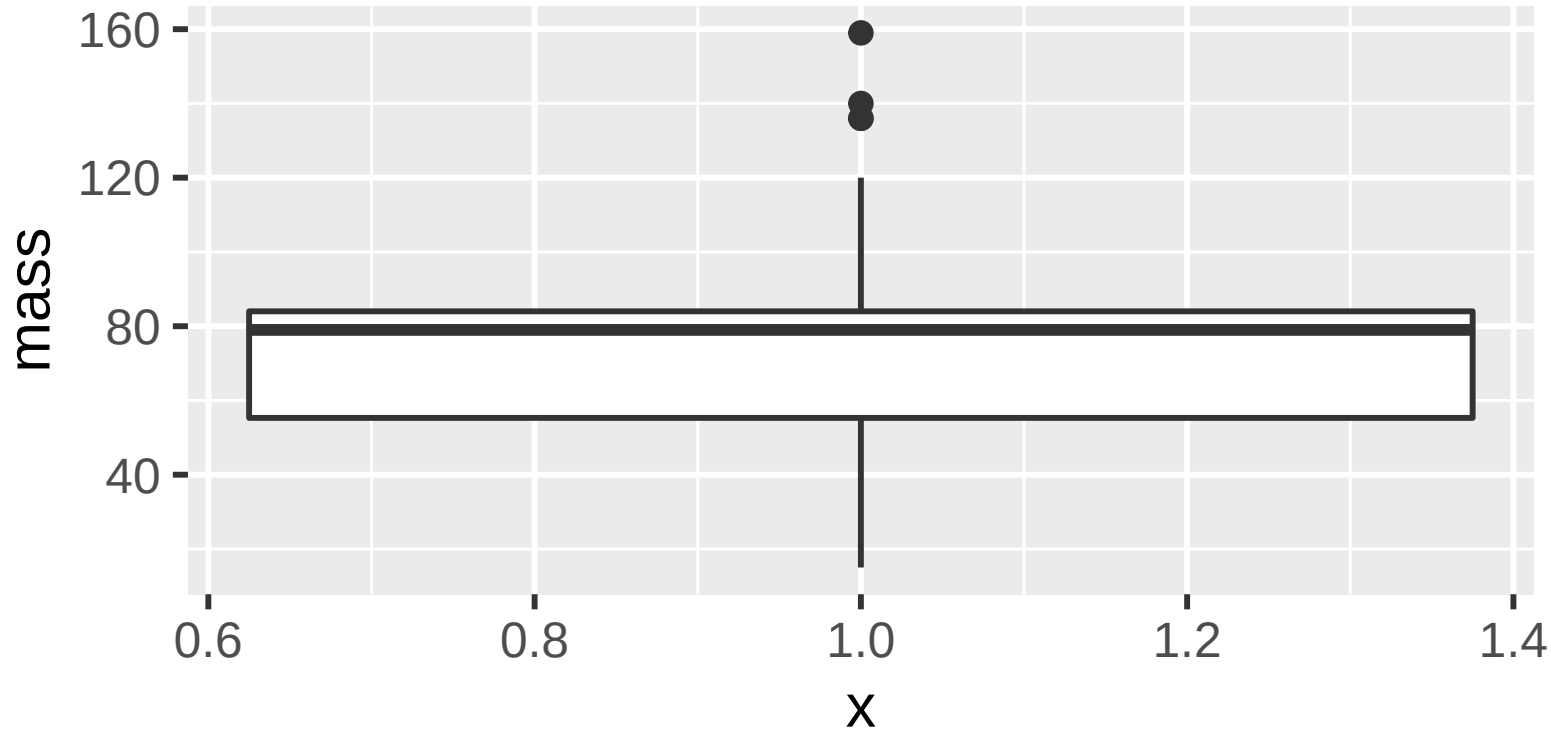
```
ModStarWars %>%
  filter(mass<1000) %>%
  ggplot(aes(x=1,y=mass))+
  geom_boxplot()
```



Base starwars

Vamos a excluir a Jabba del análisis y ver cómo queda el boxplot de masa:

```
ModStarWars %>%
  filter(mass<1000) %>%
  ggplot(aes(x=1,y=mass))+
  geom_boxplot()
```

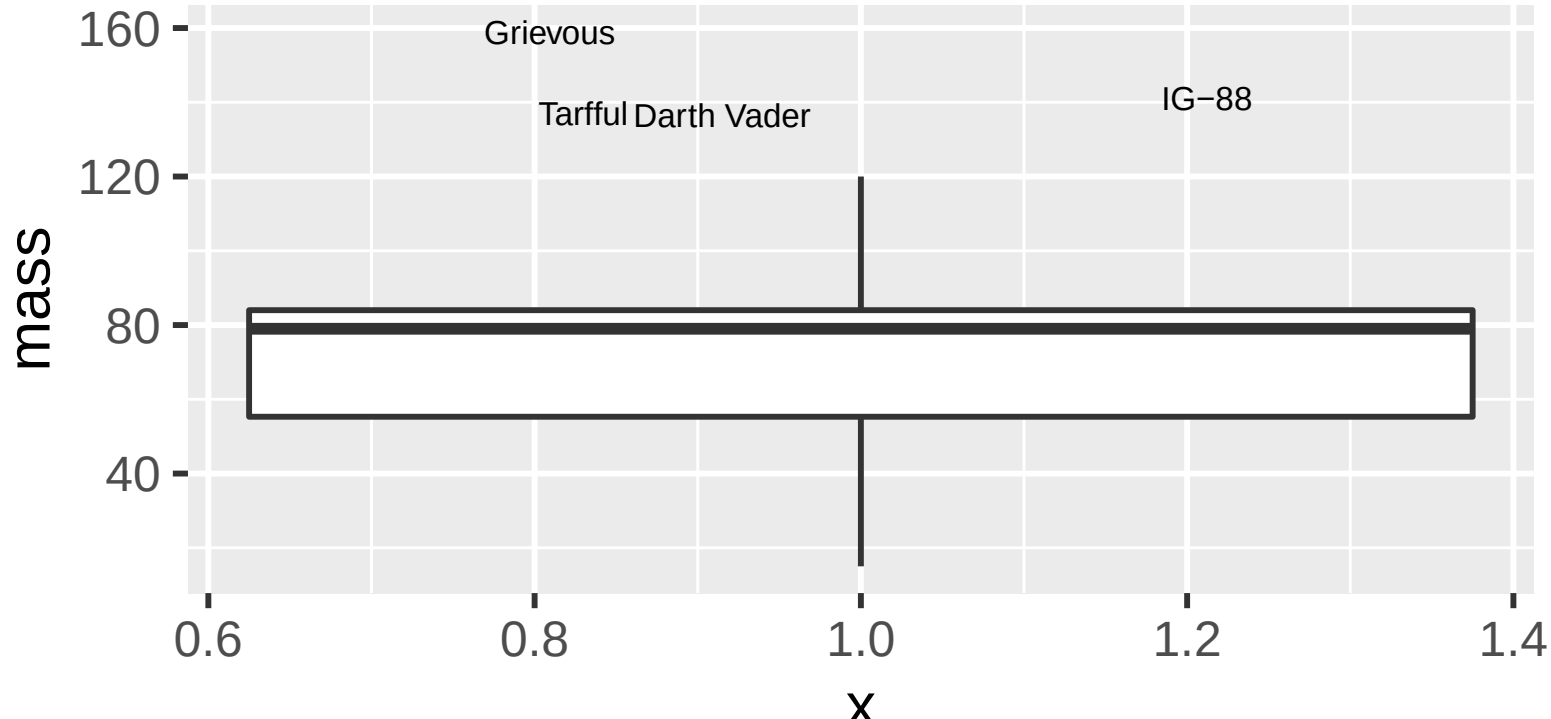


Notar la influencia gráfica que puede tener un outlier.

Base starwars

Vamos a buscar cómo se llaman los outliers:

```
set.seed(1);ModStarWars %>%
  filter(mass<1000) %>%
  ggplot(aes(x=1,y=mass))+
  geom_boxplot(outlier.alpha = 0)+
  ModStarWars %>%   filter(mass<1000) %>%
  mutate(q3=quantile(mass,0.75),uW=q3+1.5*IQR(mass)) %>%
  filter(mass>uW) %>%
  geom_text(data=.,aes(x=1,y=mass,label=name),size=2,
            position = position_jitter(width = 0.25))
```



Base starwars

Notemos además que hay muchos datos faltantes:

```
colSums(is.na(ModStarWars[,1:6]))
```

```
##       name     height       mass hair_color skin_color  eye_color
##          0          6         28          5          0          0
```

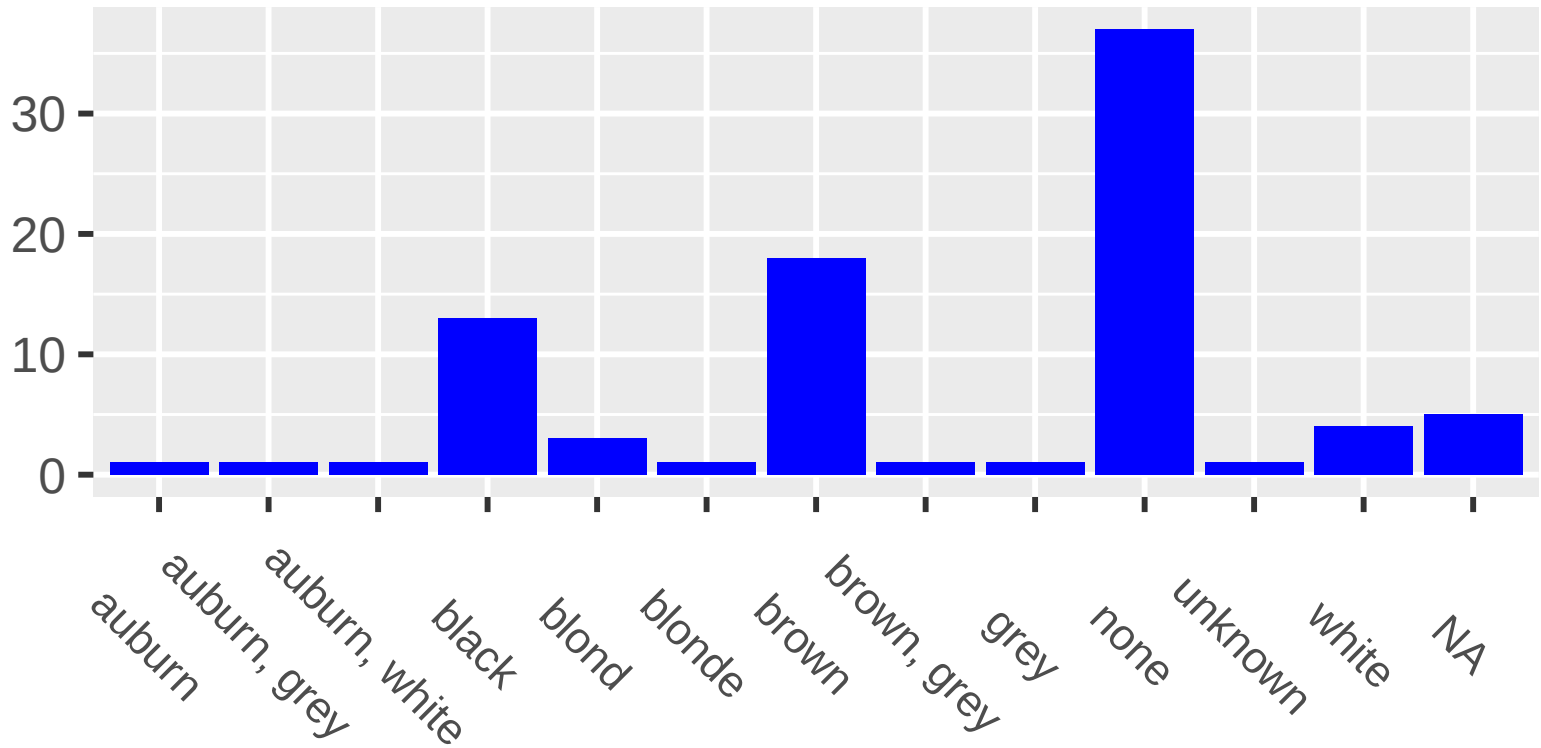
```
colSums(is.na(ModStarWars[,7:12]))
```

```
## birth_year        sex     gender  homeworld    species  starships
##         44          4          4         10          4          0
```

Base starwars

Por ejemplo, podemos hacer un diagrama de barras según el color de pelo:

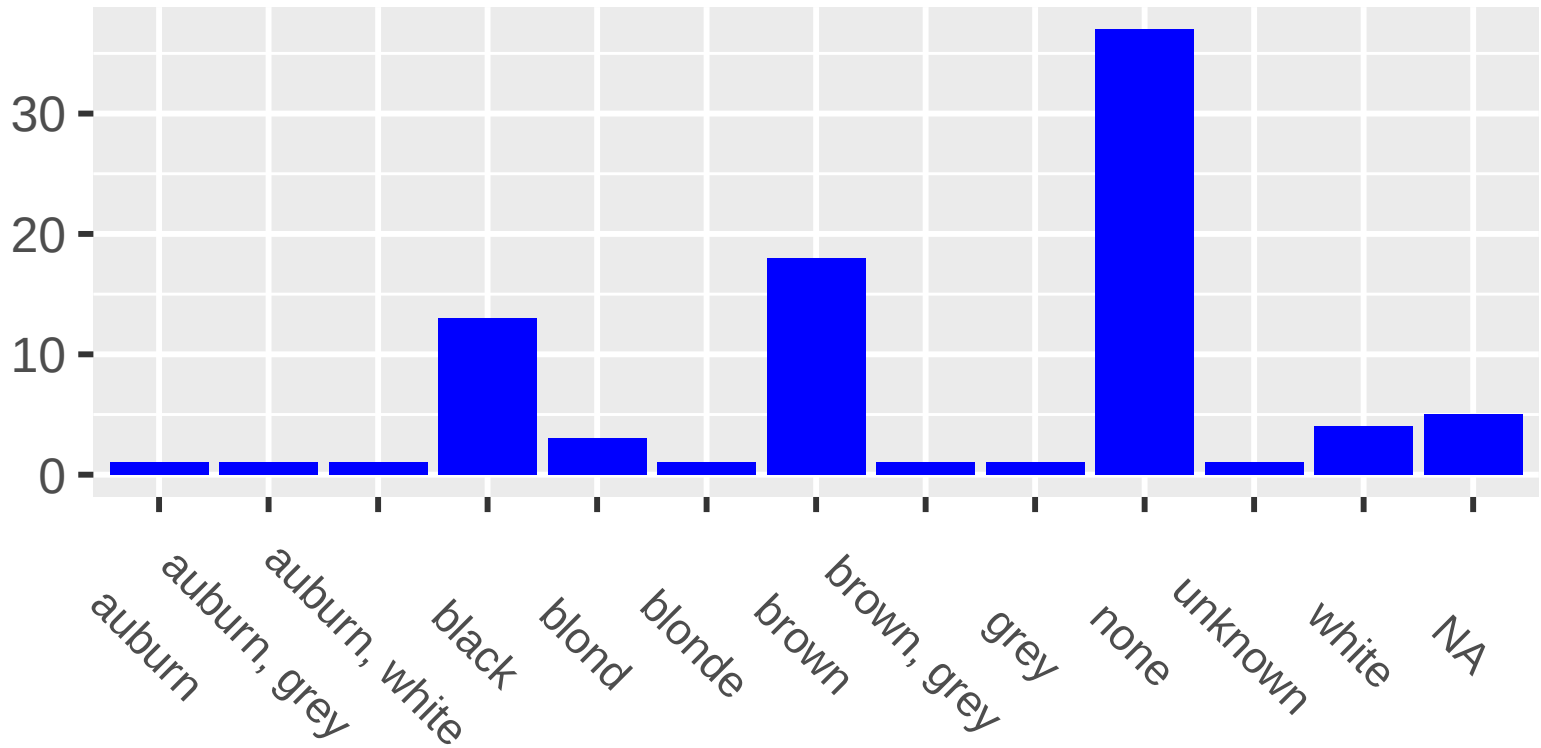
```
ModStarWars %>%
  ggplot(aes(x=hair_color))+
  geom_bar(fill="blue")+
  theme(axis.text.x = element_text(angle=-45,size=8),
        axis.title = element_blank())
```



Base starwars

Por ejemplo, podemos hacer un diagrama de barras según el color de pelo:

```
ModStarWars %>%
  ggplot(aes(x=hair_color))+
  geom_bar(fill="blue")+
  theme(axis.text.x = element_text(angle=-45,size=8),
        axis.title = element_blank())
```

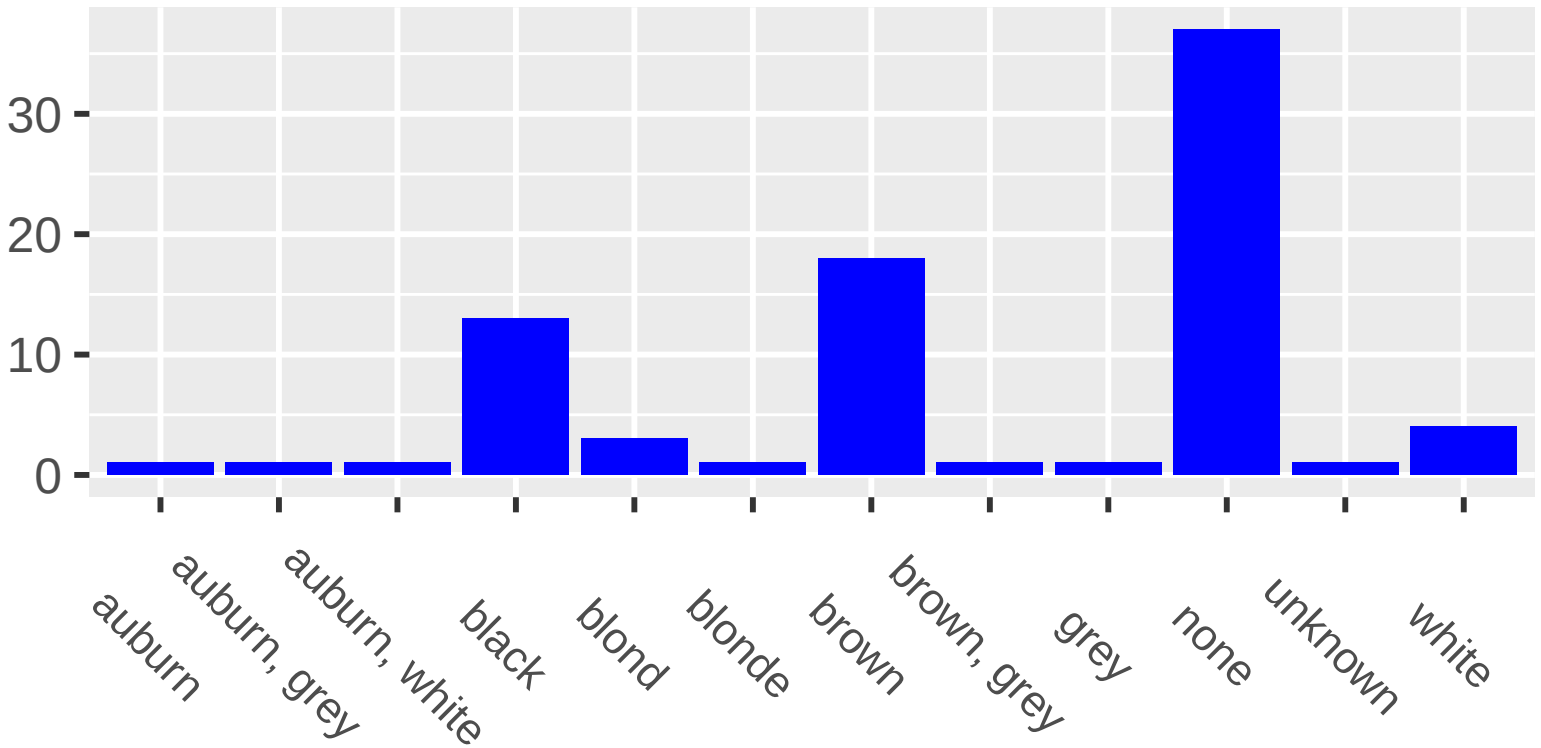


Notemos que aparece la cantidad de datos faltantes para la variable hair_color.

Base starwars

Si queremos sacar los datos faltantes, podemos apelar al comando filter y nos sacamos el problema de encima:

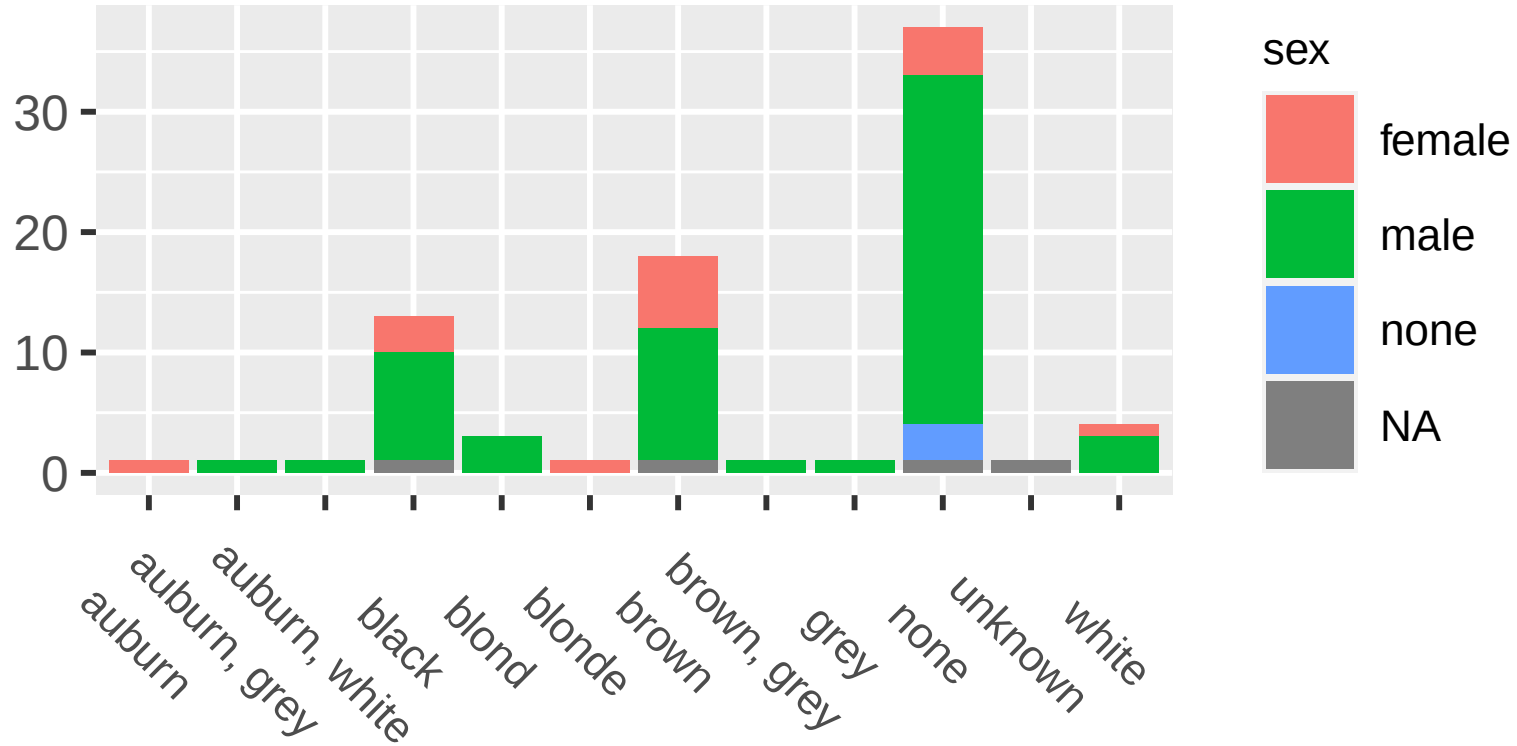
```
ModStarWars %>%
  filter(!is.na(hair_color)) %>%
  ggplot(aes(x=hair_color))+
  geom_bar(fill="blue")+
  theme(axis.text.x = element_text(angle=-45,size=8),
        axis.title = element_blank())
```



Base starwars

Pero los problemas nos siguen adonde vamos, si queremos agregar una variable fill, vamos a volver a tener datos faltantes:

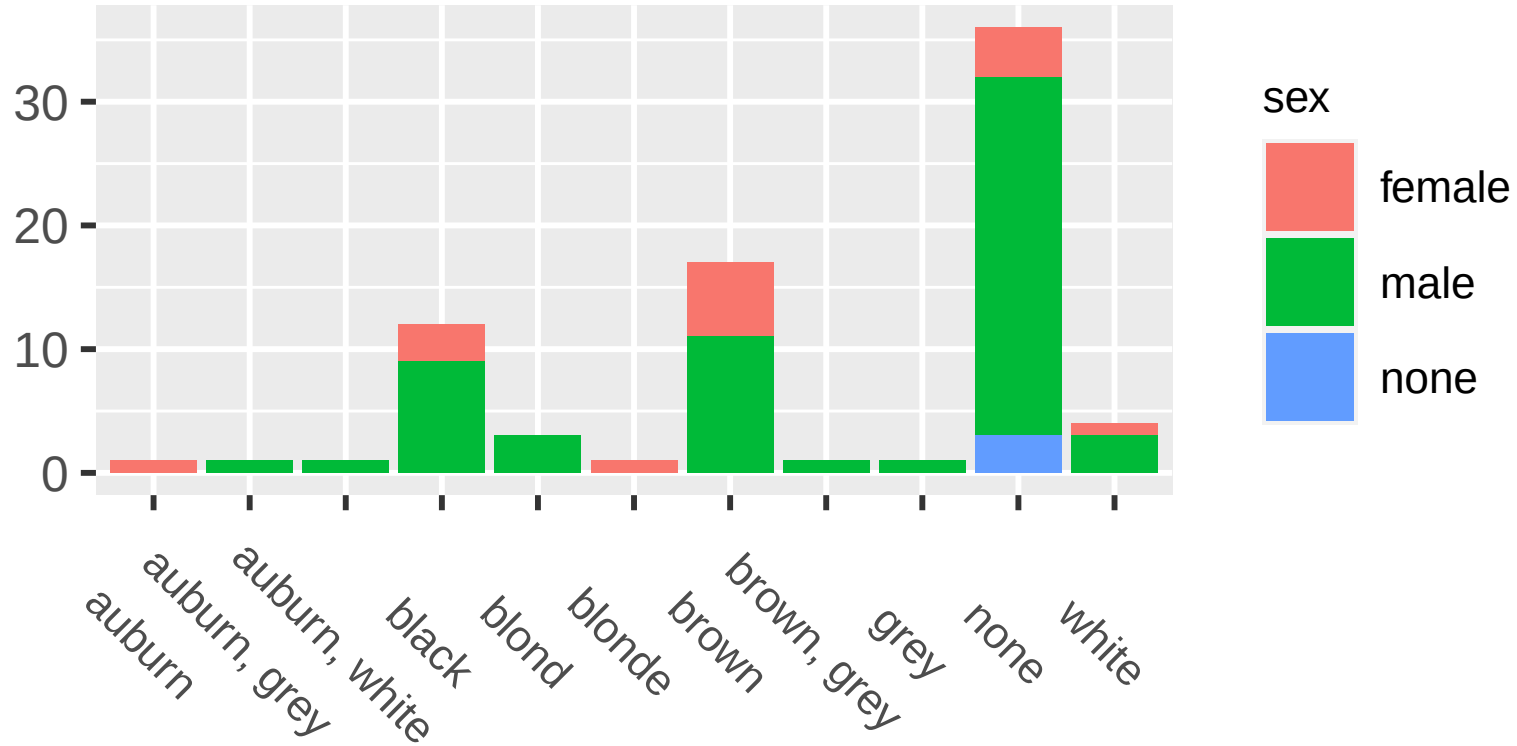
```
ModStarWars %>% filter(!is.na(hair_color)) %>%
  ggplot(aes(x=hair_color))+geom_bar(aes(fill=sex))+
  theme(axis.text.x = element_text(angle=-45,size=8),
        axis.title = element_blank(),
        legend.text = element_text(size=8),
        legend.title = element_text(size=8))
```



Base starwars

Si quiero sacar los datos faltantes a ambas variables, tengo dos opciones: puedo volver a filtrar

```
ModStarWars %>% filter(!is.na(hair_color)) %>%  filter(!is.na(sex)) %>%
  ggplot(aes(x=hair_color))+geom_bar(aes(fill=sex))+
  theme(axis.text.x = element_text(angle=-45,size=8),
        axis.title = element_blank(),
        legend.text = element_text(size=8),
        legend.title = element_text(size=8))
```



Base starwars

Otra opción es usar rowSums para ver cuáles de las filas tienen datos faltantes en alguna de las variables deseadas (la suma será positiva):

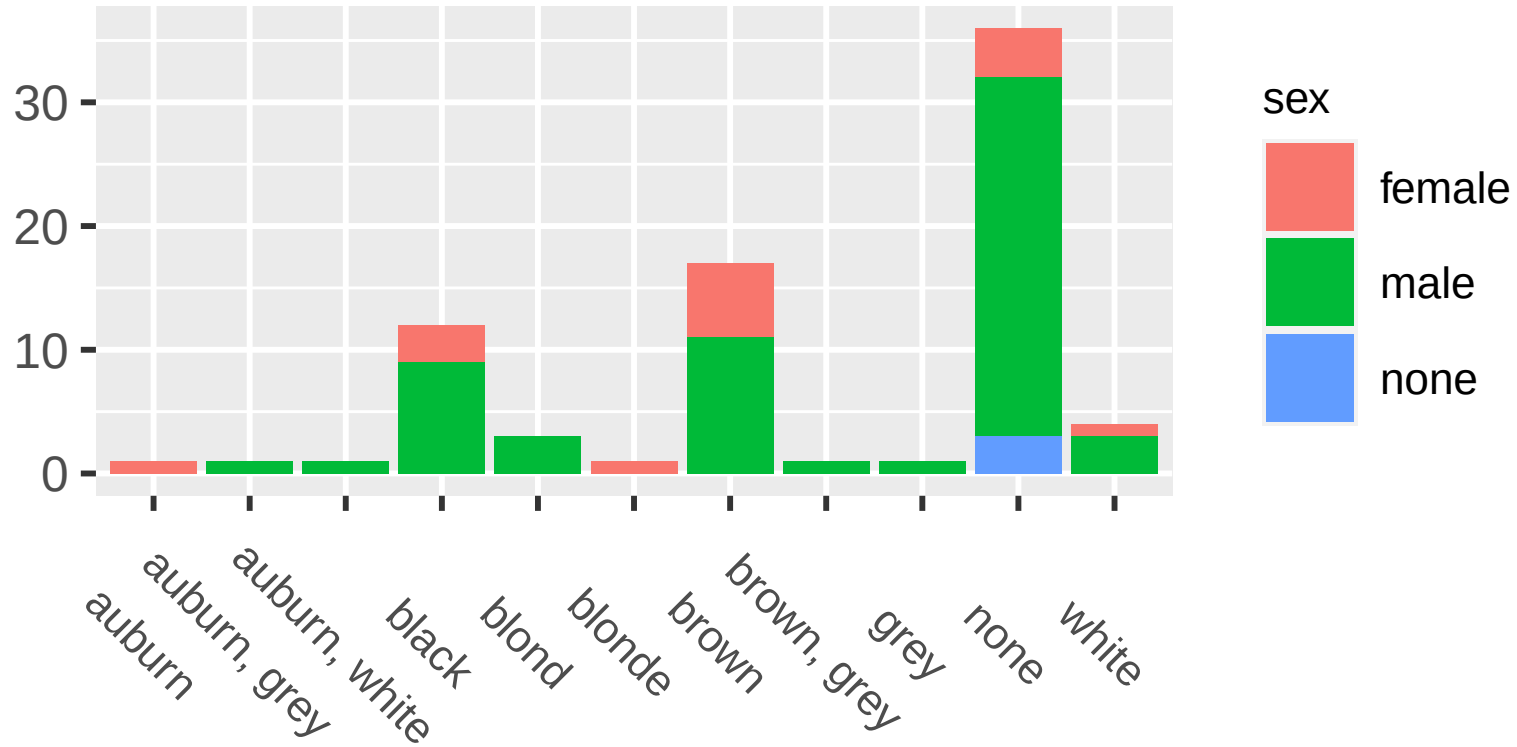
```
rS=ModStarWars %>%
  select(hair_color,sex) %>%
  is.na() %>%
  rowSums()
NAStarWars=ModStarWars %>%
  select(hair_color,sex) %>%
  mutate(rSum=rS)
head(NAStarWars)
```

```
## # A tibble: 6 x 3
##   hair_color  sex     rSum
##   <chr>       <chr>  <dbl>
## 1 blond       male       0
## 2 <NA>        none       1
## 3 <NA>        none       1
## 4 none        male       0
## 5 brown       female     0
## 6 brown, grey male       0
```

Base starwars

Con esta nueva base podemos hacer el filtrado de rSum==0:

```
NAStarWars %>% filter(rSum==0) %>%
  ggplot(aes(x=hair_color))+geom_bar(aes(fill=sex))+
  theme(axis.text.x = element_text(angle=-45,size=8),
        axis.title = element_blank(),
        legend.text = element_text(size=8),
        legend.title = element_text(size=8))
```



Base starwars

La estrategia de rowSums parece más rebuscada. Sin embargo, es útil cuando se quieren filtrar observaciones con algún dato faltante:

```
rS=ModStarWars %>%
  is.na() %>%
  rowSums()
FiltStarWars=ModStarWars %>%
  mutate(rSum=rS) %>%
  filter(rS==0)
nrow(FiltStarWars)
```

```
## [1] 29
```

Base starwars

Por último, no siempre los NA son un problema, a veces aportan información. Por ejemplo, podemos quedarnos con los que son datos faltantes de hair_color a ver si hay algún motivo por el cual los datos falten:

```
ModStarWars %>%
  filter(is.na(hair_color)) %>%
  select(name,hair_color,skin_color,species)
```

```
## # A tibble: 5 x 4
##   name                  hair_color skin_color       species
##   <chr>                 <chr>      <chr>            <chr>
## 1 C-3PO                 <NA>       gold             Droid
## 2 R2-D2                 <NA>       white, blue      Droid
## 3 R5-D4                 <NA>       white, red       Droid
## 4 Greedo                <NA>       green            Rodian
## 5 Jabba Desilijic Tiure <NA>       green-tan, brown Hutt
```

Base starwars

Por último, no siempre los NA son un problema, a veces aportan información. Por ejemplo, podemos quedarnos con los que son datos faltantes de hair_color a ver si hay algún motivo por el cual los datos falten:

```
ModStarWars %>%
  filter(is.na(hair_color)) %>%
  select(name,hair_color,skin_color,species)
```

```
## # A tibble: 5 x 4
##   name                  hair_color skin_color       species
##   <chr>                 <chr>      <chr>            <chr>
## 1 C-3PO                 <NA>       gold             Droid
## 2 R2-D2                 <NA>       white, blue      Droid
## 3 R5-D4                 <NA>       white, red       Droid
## 4 Greedo                <NA>       green            Rodian
## 5 Jabba Desilijic Tiure <NA>       green-tan, brown Hutt
```

En este caso, es porque les falta pelo o simplemente son droides.