

R 2019

25/03/2019

1ER PARCIAL

- 8 de abril
- es sin nota
- es obligatorio
- los que puedan, traigan sus laptops (con carga)

TALLER R-LADIES EN MENDOZA: CIENCIA DE DATOS CON R TIDYVERSE

R-Ladies es una organización mundial para promover la diversidad de género en la comunidad R.



```
library(dplyr)  
rladies_global %>%  
  filter(city == 'Mendoza')
```

Martes

26/3/2019

17hs-20hs

Ciencia de datos con R - Taller

```
> info = read(IntroTaller)
```

```
> print(info)
```

- * Procesamiento de datos con tidyverse

- * Reportes con Rmarkdown

REPASO DE LA CLASE PASADA

ggplot2

```
p <- ggplot(mtcars)      # creo un objeto ggplot con los datos mtcars
p <- ggplot(mtcars) + aes(mpg, wt) # le agrego el mapeo de variables que quiero
p <- ggplot(mtcars) + aes(mpg, wt) + geom_point() # le agrego como quiero que represente
p                                     # imprimo la figura
```

```
ggplot(data = mi-data-frame) +
  geom_point/line/etc(mapping = aes(col1, col2, colour="blue", etc.))
```

solo data.frames

- **data:** el dataframe que contiene los datos a graficar
- **geoms:** el tipo de objeto geométrico que representa los datos: puntos, líneas, polígonos, etc.
- **aesthetics:** describe las características visuales que representan los datos, por ejemplo, posición, tamaño, color, forma, etc
- **scale:** para cada aesthetic, describe como se mapea la característica visual a valores por ejemplo, escala logarítmica, escala de color, de tamaño, de forma, etc.
- **stats:** describe transformaciones estadísticas que resumen los datos, e.g. una regresión
- **facets:** permite separar en gráficos distintos de acuerdo a variables categóricas

MANIPULACIÓN DE DATOS: PAQUETE DPLYR

Al igual que `ggplot2`, `dplyr` solo acepta *data.frames*.

```
> install.packages("dplyr")
```

```
> library(dplyr)
```


PRINCIPALES FUNCIONES DE DPLYR

- `select()`: selecciona columnas
- `arrange()`: reordena filas
- `filter()`: filtra observaciones, condicionalmente
- `mutate()`: crea variables a partir de otras
- `summarise()`: reduce varios valores a uno solo (generalmente agrupamos primero con `group_by`)

select()

Con select podemos elegir las columnas

```
install.packages("mosaicData") # instalo el paquete que tiene los datos
library(mosaicData) # cargo el paquete en la sesión, que me da acceso a los datos
b <- as.data.frame(Birthdays) # cambio el nombre y lo convierto a data.frame
str(b) # que pinta tiene el data.frame?

# quiero un df con dos de las columnas
select(b, state, births)

# o tres
select(b, state, births, year)

# podemos además sacar columnas
select(b, -wday, -date)

# si quisieran guardar el resultado para usarlo después
nombre_inventado_por_uds <- select(b, -wday, -date)
```

arrange()

Con `arrange` podemos ordenar según valores de columnas

```
b # el df está ordenado por año  
  
# ahora, además, ordeno por estados  
arrange(b, state)  
  
# ahora, por estados y por nacimientos  
arrange(b, state, births)  
  
# idem pero nacimientos, pero de mayor a menor  
arrange(b, state, desc(births))
```

`mutate()`

Con `mutate` podemos armarnos una nueva columna a partir de otra(s)

```
mutate(b, normalized = births/mean(births)) # normalizando
```

`filter()`

Con `filter` podemos filtrar observaciones mediante condiciones lógicas (`>`, `>=`, `<`, `<=`, `!=`, `==`, más los operadores `&` y `|`)

```
# devuelve solo estado de Washington
filter(b, state == "WA")

# Washington en 1974
filter(b, state == "WA" & year == 1974)

# idem, pero más de 180 nacimientos
filter(b, state == "WA" & year == 1974 & births > 180)
```

summarise() y group_by

Con summarise podemos resumir variables, usualmente agrupando.

```
# gran-promedio-gran
summarise(b, promedio = mean(births) )

# pero es muy eficiente para manipular datos agrupados
# por ejemplo, agrupo por mes
por_mes <- group_by(b, month)
summarise(por_mes, births_por_mes = mean(births))

# ahora agrupo por estado
por_estado <- group_by(b, state)
summarise(por_estado, births_por_estado = mean(births))
```

PRÁCTICA 3

Descargar [práctica 3](#).