

Dplyr %in% Tidyverse

Sesión 1

Natalie Julian - www.nataliejulian.com

Estadística UC y Data Scientist en Zippedi Inc.

Bienvenida

Hola!

¿Cómo estás? Espero que muy bien!

Te doy la bienvenida al curso **R intermedio**, continuación del curso *R basics*. En el curso *R basics* partimos desde cero, conociendo los objetos y las funciones base, importación y limpieza de datos incluyendo análisis descriptivo y creación de gráficos, finalmente utilizamos algunas funciones del paquete *dplyr*, como por ejemplo `group_by` y `summarise`. En este curso lo ideal es profundizar en todos los puntos vistos en el curso *R basics*, en particular veremos las funciones más útiles del paquete *tidyverse*.

Mucho éxito!

Natalie

tidyverse: Paquete de R

Tidyverse es un conjunto de paquetes que incluye herramientas para la manipulación, exploración y visualización de datos.


Quizás no lo sabías pero dplyr es uno de estos paquetes incorporados en tidyverse, y es muy útil para el manejo de datos.



Hoy trabajaremos con dplyr y tidyr :)

The tidyverse

Components



The tidyverse is a collection of R packages that share common philosophies and are designed to work together. This site is a work-in-progress guide to the tidyverse and its packages.

PAQUETE DPLYR

Recordemos a dplyr

Ya trabajamos en el curso *R basics* con algunas funciones de dplyr, pero lo retomaremos nuevamente.



dplyr es un paquete que introduce una nueva sintaxis a R para la manipulación de datos.

¿Cómo accedemos a dplyr?

Como dplyr es un paquete, ya conocemos que para acceder a él necesitamos primero instalarlo y luego cargarlo. Una vez instalado no es necesario instalarlo nuevamente:

```
install.packages("dplyr") #Instala el paquete
```

```
library(dplyr) #Carga el paquete
```

Y listo! A continuación veremos algunas cosillas que podemos realizar con dplyr...

GROUP_BY: ESTADÍSTICAS POR GRUPO O
CATEGORÍA

Datos iris

Utilizaremos un set de datos bastante conocido, el set de datos *iris* que contiene datos de distintas flores, su especie, largo de pétalo, sépalo, etcétera. Para acceder a estos datos basta con correr la siguiente línea de código:

```
data(iris)
```

Y se cargará un objeto `data.frame` llamado *iris*:

```
head(iris)
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa

Species

Existen tres especies de flores en el dataset:

- setosa
- virginica
- versicolor

(Esto lo podemos obtener de la siguiente forma:

```
unique(iris$Species)
[1] setosa      versicolor virginica
Levels: setosa versicolor virginica
```

Si no lo recuerdas, puedes visitar el curso R basics).

Estadísticas descriptivas para cada especie

Podemos obtener estadísticas descriptivas de la variable `Petal.Length` para cada especie:

```
summary(iris$Petal.Length[iris$Species=="setosa"])
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
1.000	1.400	1.500	1.462	1.575	1.900

```
summary(iris$Petal.Length[iris$Species=="versicolor"])
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
3.00	4.00	4.35	4.26	4.60	5.10

```
summary(iris$Petal.Length[iris$Species=="virginica"])
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
4.500	5.100	5.550	5.552	5.875	6.900

Okay, obtuvimos estadísticas descriptivas, pero, el formato de salida es un poco engorroso de manejar, quizás sería mejor obtener estos resultados en una dataframe para poder comparar fácilmente las distintas especies... ¿existirá alguna forma alternativa de hacer lo mismo?

Por supuesto, con dplyr!

Podemos agrupar por `Specie` (con `group_by`) y resumir la información de cada grupo respecto a una variable (con `summarise`), funciones incorporadas en el paquete `dplyr`:

```
iris%>%  
  group_by(Species)%>%  
    summarise(Min=min(Petal.Length),  
              firsq=quantile(Petal.Length, 0.25),  
              Median=median(Petal.Length),  
              Mean=mean(Petal.Length),  
              thirdq=quantile(Petal.Length, 0.75),  
              Max=max(Petal.Length))
```

A tibble: 3 x 7

	Species <fct>	Min <dbl>	firsq <dbl>	Median <dbl>	Mean <dbl>	thirdq <dbl>	Max <dbl>
1	setosa	1	1.4	1.5	1.46	1.58	1.9
2	versicolor	3	4	4.35	4.26	4.6	5.1
3	virginica	4.5	5.1	5.55	5.55	5.88	6.9

Pero... ¿qué hace cada línea de código?

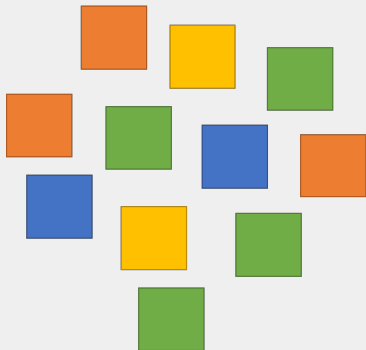
```
iris%>% #Partimos con la dataframe
  group_by(Species)%>% #Agrupamos por especie
  summarise(Min=min(Petal.Length), #Calculamos el minimo de petal.length por grupo
            firsq=quantile(Petal.Length, 0.25), #Calculamos el primer cuartil de
                                                    petal.length por grupo
            Median=median(Petal.Length), #Calculamos la mediana de petal.length por grupo
            Mean=mean(Petal.Length), #Calculamos la media de petal.length por grupo
            thirdq=quantile(Petal.Length, 0.75), #Calculamos el tercer cuartil de
                                                    petal.length
            Max=max(Petal.Length)) #Calculamos el maximo de petal.length por grupo
```

Recuerda que el # significa que comienza un comentario!

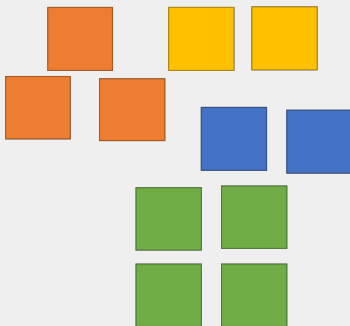
¿Qué hace group_by?

Visualmente hace algo así:

Datos



Datos con
group_by



¿Qué ganamos con dplyr?

El formato de salida es muy manejable: Podemos guardarlo en un objeto y extraer una columna, una fila y hacer cálculos.

¿Qué ganamos?

```
(tabla<-iris%>%
  group_by(Species)%>%
  summarise(Min=min(Petal.Length),
            firsq=quantile(Petal.Length, 0.25),
            Median=median(Petal.Length),
            Mean=mean(Petal.Length),
            thirdq=quantile(Petal.Length, 0.75),
            Max=max(Petal.Length)))
```

A tibble: 3 x 7

Species	Min	firsq	Median	Mean	thirdq	Max
<fct>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1 setosa	1	1.4	1.5	1.46	1.58	1.9
2 versicolor	3	4	4.35	4.26	4.6	5.1
3 virginica	4.5	5.1	5.55	5.55	5.88	6.9


```
tabla[2,-1] - tabla[3, -1 ] #diferencia versicolor - virginica
  Min firsq Median Mean thirdq Max
1 -1.5 -1.1 -1.2 -1.292 -1.275 -1.8
```

#Distancias:

```
sqrt(sum((tabla[2,-1] - tabla[3, -1 ])**2)) #versicolor - virginica
[1] 3.381551
```

```
sqrt(sum((tabla[1,-1] - tabla[2, -1 ])**2)) #setosa - versicolor
[1] 6.78984
```

```
sqrt(sum((tabla[1,-1] - tabla[3, -1 ])**2)) #setosa - virginica
[1] 10.12722
```

#setosa y virginica son las que más se diferencian en estadísticas

SLICE: PARTIENDO NUESTROS DATOS

Quitar las réplicas

Supongamos que nos interesa seleccionar la primera flor de cada tipo de especie, esto lo podemos lograr como sigue:

```
iris%>%
  group_by(Species)%>%
  slice(1)
# A tibble: 3 x 5
# Groups:   Species [3]
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
    <dbl>         <dbl>         <dbl>         <dbl> <fct>
1         5.1           3.5           1.4           0.2 setosa
2          7.0           3.2           4.7           1.4 versicolor
3         6.3           3.3           6.0           2.5 virginica
```

¿Y si queremos la última?

```
iris%>%
  group_by(Species)%>%
  slice(length(Sepal.Length))
# A tibble: 3 x 5
# Groups:   Species [3]
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
    <dbl>         <dbl>         <dbl>         <dbl> <fct>
1          5.4           4.7           1.4           0.2 setosa
2         5.7           2.8           4.1           1.3 versicolor
3         5.9           3.0           5.1           1.8 virginica
```

Slice más grande

También podríamos extraer varios registros, por ejemplo, los primeros 5 por grupo:

```
iris%>%
  group_by(Species)%>%
  slice(1:5)
# A tibble: 15 x 5
# Groups:   Species [3]
   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
   <dbl>         <dbl>         <dbl>         <dbl> <fct>
1      5.1         3.5         1.4         0.2 setosa
2      4.9         3         1.4         0.2 setosa
3      4.7         3.2         1.3         0.2 setosa
4      4.6         3.1         1.5         0.2 setosa
5      5          3.6         1.4         0.2 setosa
6      7          3.2         4.7         1.4 versicolor
7      6.4         3.2         4.5         1.5 versicolor
8      6.9         3.1         4.9         1.5 versicolor
9      5.5         2.3         4          1.3 versicolor
10     6.5         2.8         4.6         1.5 versicolor
11     6.3         3.3         6          2.5 virginica
12     5.8         2.7         5.1         1.9 virginica
13     7.1         3         5.9         2.1 virginica
14     6.3         2.9         5.6         1.8 virginica
15     6.5         3         5.8         2.2 virginica
```

Esta función es muy útil, por ejemplo, cuando manejamos **Fechas**, agrupamos por fecha, y filtramos los primeros o últimos registros del día, dependiendo del interés.

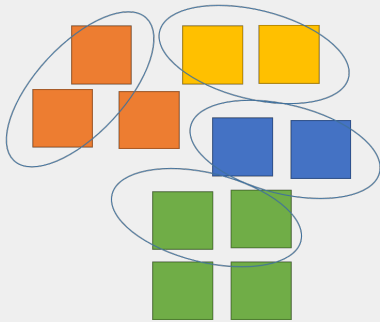
¿Qué hace group_by y slice?

Visualmente hace algo así:

Datos



Datos con
group_by y
slicing 2



FILTER: FILTRANDO DATOS

Filtrando por casos

Supongamos que nos interesa considerar en el análisis a aquellas plantas que son de tipo versicolor o setosa.

Esto se puede hacer fácilmente de las siguientes formas:

```
#Forma 1 selecciona registros versicolor o setosa
```

```
filtro<-iris[which(iris$Species=="versicolor"|iris$Species=="setosa"),]
```

```
#Forma 2 filtra registros iguales a versicolor o setosa
```

```
filtro<-subset(iris, Species=="versicolor"|Species=="setosa")
```

```
#Forma 3 quita los registros virginica
```

```
filtro<-iris[-which(iris$Species=="virginica"), ]
```

```
#Forma 4 filtra los casos distintos a virginica
```

```
filtro<-subset(iris, Species!="virginica")
```

Todas las formas generan el mismo resultado.

Con dplyr

Con dplyr también es bastante sencillo:

```
iris%>%  
  filter(Species=="versicolor"|Species=="setosa")
```

#O también:

```
iris %>%  
  filter(Species!="virginica")
```

Podemos añadir también más condiciones de forma sencilla:

```
iris %>%  
  filter(Species!="virginica", Sepal.Length>5)
```

¿Qué hace filter?

Visualmente hace algo así:

Datos

x1	x2	x3

Datos con filter

x1	x2	x3

SELECT: SELECCIONA COLUMNAS

Seleccionando columnas de interés

Supongamos que sólo nos interesan las columnas *Species* y *Sepal.Width* podemos seleccionar estas columnas de las siguientes formas:

#Forma 1: Selecciona el número de las columnas de interés

```
iris[,c(2,5)]
```

#Forma 2: Selecciona el nombre de las columnas de interés

```
iris[, c("Sepal.Width", "Species")]
```

#Forma 3: Quita las columnas que no son de interés

```
iris[, -c(1, 3, 4)]
```

#Forma 4: con dplyr, selecciona las columnas de interés

```
iris%>%  
  select(Sepal.Width, Species)
```

¿Qué hace select?

Visualmente hace algo así:

Datos

x1	x2	x3

Datos con
`select(x2,x3)`

x2	x3

Otras utilidades de la función select con el paquete tidyr

Si cargamos el paquete tidyr, accedemos a ciertas funciones muy útiles:

- Si quiero extraer las columnas que contienen cierto caracter, palabra o string. En este caso por ejemplo, para extraer las columnas que contienen la palabra *Length* utilizamos:

```
iris %>%  
  select(contains("Length"))
```

- Si quiero extraer las columnas que comienzan con cierto texto. Por ejemplo, para extraer las columnas que comienzan con *Sepal* utilizamos:

```
iris %>%  
  select(starts_with("Sepal"))
```

De manera similar, si queremos extraer las columnas que terminan con cierto texto, se puede utilizar la función `ends_with`.

MUTATE: CREA NUEVAS VARIABLES O REDEFINE
OTRAS

Variable radio

Suponga que nos interesa crear una nueva variable llamada radio que se calcule como sigue:

$$radio = \frac{2 * Petal.Length * Petal.Width}{petal.Length + Petal.Width}$$

Podemos hacerlo de las siguientes formas:

#Forma 1 usual:

```
iris$radio<-2*iris$Petal.Length*iris$Petal.Width/(iris$Petal.Length+iris$Petal.Width)
```

```
head(iris)
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species	radio
1	5.1	3.5	1.4	0.2	setosa	0.3500000
2	4.9	3.0	1.4	0.2	setosa	0.3500000
3	4.7	3.2	1.3	0.2	setosa	0.3466667
4	4.6	3.1	1.5	0.2	setosa	0.3529412
5	5.0	3.6	1.4	0.2	setosa	0.3500000
6	5.4	3.9	1.7	0.4	setosa	0.6476190

#Forma 2 con dplyr:

```
iris%>%  
  mutate(radio=2*Petal.Length*Petal.Width/(Petal.Length+Petal.Width))
```

¿Qué hace mutate?

Visualmente hace algo así:

Datos

x1	x2	x3

Datos con
mutate

x1	x2	x3	$F(x1, x2, x3)$

ARRANGE: ORDENA LAS FILAS DE ACUERDO A UN
CRITERIO

Ordenando datos

Un ejemplo muy sencillo es por ejemplo, ordenar las flores de mayor a menor respecto a Largo de pétalo, esto fácilmente se puede realizar de las siguientes formas:

#Forma 1:

```
iris[order(iris$Petal.Length, decreasing=TRUE), ]
```

#Forma 2 con dplyr:

```
iris%>%  
  arrange(desc(Petal.Length))
```

De forma creciente es:

#Forma 1:

```
iris[order(iris$Petal.Length), ]
```

#Forma 2 con dplyr:

```
iris%>%  
  arrange(Petal.Length)
```

¿Qué hace arrange?

Visualmente hace algo así:

Datos

x1	x2	x3
		5
		9
		6
		1
		2
		6
		3
		6
		8
		1

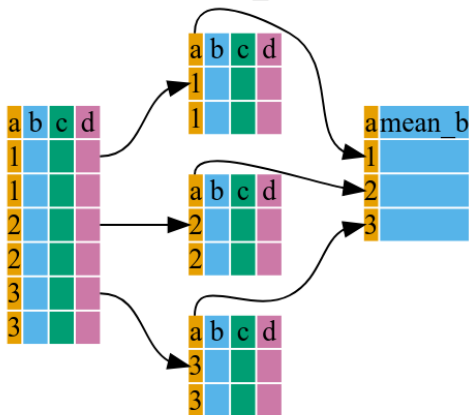
Datos con
arrange(x3)

x1	x2	x3
		1
		1
		2
		3
		5
		6
		6
		6
		8
		9

EL OPERADOR `% > %` LLAMADO PIPE DE PIPELINE
(TUBERÍA) NO ES COINCIDENCIA...

Ejemplo de una tubería sencilla

```
gapminder %>%  
group_by(a) %>%  
summarize(mean_b=mean(b))
```



Pipeline más compleja

Podemos ir realizando operaciones varias y seguir una tubería más compleja. Ejemplo:

```
iris%>%
  filter(Species!="versicolor")%>%
  mutate(radius=2*Petal.Length*Petal.Width/(Petal.Length+Petal.Width)) %>%
  group_by(Species)%>%
  summarise(Min=min(Petal.Length),
            firstq=quantile(Petal.Length, 0.25),
            Median=median(Petal.Length),
            Mean=mean(Petal.Length),
            thirdq=quantile(Petal.Length, 0.75),
            Max=max(Petal.Length),
            Avgradio=mean(radius))%>%
  arrange(desc(Max))
```

```
# A tibble: 2 x 8
  Species      Min firstq Median  Mean thirdq   Max Avgradio
  <fct>      <dbl> <dbl>  <dbl> <dbl>  <dbl> <dbl>    <dbl>
1 virginica   4.5   5.1   5.55  5.55   5.88   6.9     2.96
2 setosa      1     1.4   1.5   1.46   1.58   1.9     0.413
```

Pero... ¿qué hace cada línea de código?

```
iris%>% #Partimos con la dataframe iris
  filter(Species!="versicolor")%>% #Filtramos aquellas flores que no sean versicolor
  mutate(radio=2*Petal.Length*Petal.Width/(Petal.Length+Petal.Width)) %>% #creamos la
                                                                    variable radio
  group_by(Species)%>% #Agrupamos por especie
  summarise(Min=min(Petal.Length), #Calculamos estadísticas por especie (grupo)
            firsq=quantile(Petal.Length, 0.25),
            Median=median(Petal.Length),
            Mean=mean(Petal.Length),
            thirdq=quantile(Petal.Length, 0.75),
            Max=max(Petal.Length),
            Avgradio=mean(radio))%>%
  arrange(desc(Max)) #Finalmente, ordenamos de forma decreciente respecto al maximo
```

Esto, es súper potente! ¿Imaginas la tubería que sigue el código anterior?

En resumen, podemos hacer mucho con dplyr :)

Las operaciones por sí solas podemos hacerlas con o sin dplyr, pero cuando tenemos tuberías más complejas, resulta mucho más comprensible el proceso utilizando dplyr.

