Introducción a dplyr

Armando Ocampo

dplyr

dplyr es una librería que nos permite manipular datos, ordenarlos y extraer información necesaria. Esta paquetería no forma parte de las paqueterías base de R. Por lo que tenemos que descargarla.

install.packages("dplyr")

Una vez instalada, la llamaremos para comenzar a trabajar.

library(dplyr)

dplyr pertenece a un conjunto de paqueterias llamado tidyverse, el cual nos permite manipular información. Es bueno que sepas esto, no obstante, el día de hoy solo trabajaremos con la paqueteria mencionada.

Las funciones principales de dplyr son filter(), arrange(), mutate(), select(), group_by() y summarise(). También se conocen como verbos. Además existe un elemento denominado pipe " %>%" que nos permite unir dos elementos y tabajar con ellos.

filter()

El primer verbo que utilizaremos se llama filter() y nos permite filtrar un conjunto de datos con una caracteristica en específico. Los ejemplos mostrados en este artículo se basaran en información de los data set *iris* y *mtcars*. Antes de comenzar, haremos un resumen de los elementos que contiene cada data set.

```
str(iris)
summary(iris)
glimpse(iris)


str(mtcars)
summary(mtcars)
glimpse(mtcars)
```

Nota_ glimpse() es una función de dplyr que permite observar un resumen de la información presente en el data frame de trabajo.

Comenzaremos a trabajar con el dataset de iris. De este data set nos quedaremos solo con las especies setosas. En este caso dentro de la función filter() se coloca el nombre de la variable y la condición a seguir.

Nota 1: el nombre de la variable debe coincidir con la forma en la cual está escrito en el df.

Nota 2: en caso de realizar una condición con elementos de tipo caracter, este va en comillas. Las comparaciones numéricas van sin comillas

Nota 3: el "pipe" %>% , nos permite tomar el objeto de la izquierda y ejercer un verbo o una acción en especifico. Esta última, se escribe del lado derecho.

```r
iris %>%
  filter(Species == "setosa")

# al igual que en ejercicios previos, al correr esta parte del codigo
# el resultado se observará en la consola. Si queremos guaradar el resultado
# debemos asignarlo a un objeto

df_setosas <- iris %>%
  filter(Species == "setosa")
```

Una de las ventajas de filter es realizar varios filtros en una sola función. A continuación filtraremos las iris de tipo setosas con longitud de sepalo mayor a 4.8. Solo debemos separar cada condición con una coma

```r
iris %>%
  filter(Species == "setosa",
         Sepal.Length > 4.8)
```

Siempre que respetemos el uso de comas, podemos realizar tantos filtros como nosotros queramos. Sin correr el siguiente código. ¿Qué crees que filtraría el siguiente argumento?

```r
iris %>%
  filter(Species == "virginica",
         Petal.Length > 4.5,
         Petal.Width < 2,
         Sepal.Length > 6,
         Sepal.Width < 2.8)
```

En los ejemplos previos realizamos comparaciones con mayor y menor que e igualdades. No obstante, ¿Cómo puedo hacer una comprativa con rangos? Por ejemplo, quiero obtener las iris versicolor con una longitud de petalo entre 3.5 y 4.8. Para este caso, realizamos comparaciones con los booleanos | OR, & AND o != NOT. Para ello colocamos la primer condición del lado izquiero el booleano en el centro y la segunda condición a la derecha.

```r
iris %>%
  filter(Species == "versicolor",
         Petal.Length > 3.5 & Petal.Length < 4.8)
```

Esta regla se debe seguir cuando utilices los booleanos. Además, hay que recordar que no hay una sola respuesta correcta en R. Como ejemplo filtraremos las especies de iris para generar un df que contenga solo setosas y virginicas a partir de diferentes códigos.

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

iris %>%
  filter(Species != "versicolor")

iris %>%
  filter(Species %in% c("setosa", "versicolor"))
```

Nota: Para realizar un filtro a partir de un vector de tipo caracter se utiliza %in% en lugar de ==

TAREA con el dataset mtcars responde lo siguiente 1. ¿Cuántos automóviles tienen solo 4 cilindros? 2. ¿Cuántos automóviles tienen motor con forma de V y desplazamiento mayor a 150? 3. ¿Qué automóviles tienen 4 carburadores, 6 cilindros, transmisión manual y menos de 200 caballos de fuerza?

arrange()

El verbo arrange() permite ordenar los datos de forma ascendente a partir de una variable. A continuación, ordenaremos los datos de iris a partir de la longitud de sepalo

```
iris

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

Para ordenar de forma descendente se agrega la funcion desc() dentro del verbo arrange(). Ahora ordenaremos de forma descendente los datos de iris a partir de la longitud de pétalo

```
iris

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

El pipe %>% permite combinar varios verbos de dplyr. De esta forma podemos realizar varios verbos y manipular los datos acorde a lo que necesitamos.

Nota: cada verbo se separa con un pipe

Vamos a filtrar los datos para obtener solo las especies versicolor y aquellas iris con longitud de pétalo entre 3.5 y 4.8. Por último, ordenaremos los datos de morma descendente a partir del ancgo del pétalo.

```
iris %>%
  filter(Species == "versicolor",
         Petal.Length > 3.5 & Petal.Length < 4.8) %>%
  arrange(desc(Petal.Width))
```

Nota: los pipes también permiten separar el código en acciones individuales. Puedes subrayar ciertas partes de tu código y evaluar cada paso. Esto es útil cuando hay algún error en tu script.

mutate()

El verbo mutate() genera una nueva variable a partir de una expresión de nuestros datos. Este verbo toma como argumento el nombre de la nueva variable en el lado izquierdo y del lado derecho la expresión de interés. Como ejemplo, crearemos una variable denominada multiplicación. La cual se conformará de multiplicar la longitude sépalo por 5.

Nota 1: recuerda que cada variable de un dataset puede utilizarse como un vector independiente. Y que cada vector puede utilizarse en una función matemática.

Nota 2: recuerda que para asignar un objeto en R puedes utilizar "<-" o "=". En mutate(), asiganamos a la nueva variable solo con "="

```
iris


iris %>%
  mutate(multiplicacion = Sepal.Length * 5)

# Este resultado solo estará presente en la consola. Para guardarlo deberás
# asignarlo a un objeto
```

Como puedes ver, la expresión de la derecha corresponde a las utilizadas al comparar y trabajar con vectores. Asimismo, puedes utilizar cada variable de tu df original como un vector independiente y realizar los análisis necesarios.

```
# restaremos el valor de longitud de pétalo con ancho de pétalo. De esta forma
# conoceremos si el pétalo de una flor es más ancho o más largo. Esta nueva
# se denominará petalo

iris %>%
  mutate(petalo = Petal.Length - Petal.Width)

# podemos complicar esto un poco más. Ahora generaremos una variable compuesta
# por TRUE y FALSE a partir de una condición. Investigaremos que tipo de iris
# presenta una longitud de petalo entre 3.5 y 4.8. Aquellas flores que cumplan
# esta condición obtendrán un TRUE, caso contrario, obtendremos un FALSE
# la nueva variable se denominará rango

iris %>%
  mutate(rango = Petal.Length > 3.5 & Petal.Length < 4.8)
```

Al igual que los verbos previos, podemos utilizarlos en conjunto. Seguiendo el último ejemplo, filtraremos los datos acorde a aquellos que cumplen la condición de tener una longitud de pétalo entre 3.5 y 4.8

```
# primero generaremos la variable rango

iris %>%
  mutate(rango = Petal.Length > 3.5 & Petal.Length < 4.8)

# ahora filtraremos los datos TRUE de la variable rango

iris %>%
  mutate(rango = Petal.Length > 3.5 & Petal.Length < 4.8) %>%
  filter(rango == TRUE)
```

El orden de los factores no altera el producto. Podemos usar los verbos de dplyr sin una jerarquía fija. E incluso repetirlos.

```
iris %>%
  filter(Species == "versicolor") %>%
  mutate(rango = Petal.Length > 3.5 & Petal.Length < 4.8) %>%
  filter(rango == TRUE) %>%
  arrange(Sepal.Length ) %>%
  mutate(longitud_sepalo_pulgadas = Sepal.Length/2.5)
```

select()

El verbo select() permite seleccionar variables en específico. Esto es útil cuando tenemos un data set con variables que no necesitamos y que solo generan un df pesado. Para esto solo colocamos el nombre de las variables dentro de la función. En el data set iris seleccionaremos *species* y *Sepal.Width*

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

# para guardar este nuevo df lo asignamos a un objeto

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

Si queremos omitir una variable podemos colocar los nombres dentro de la función select() o solo colocar el nombre de la variable que queremos omitir acompañado de un guión medio - En el siguiente ejemplo, tomaremos todas las variables y quitaremos *Petal.Length*

```r
# seleccionando las variables
iris %>%
  select(Sepal.Length, Sepal.Width, Petal.Width, Species)

# omitiendo solo petal.length

iris %>%
  select(-Petal.Length)

# la mayoría de los caminos conducen a Roma
```

Asimismo, si conocemos el orden de las variables podemos seleccionarlas con los dos puntos :, por ejemplo, seleccionar de la primera a la tercera variable

```r
iris %>%
  select(1:3)

# seleccionar de la variable 1 a la variable 4
iris %>%
  select(1:4)

# o hacer dos selecciones, de la variable 1 a la 2 y de la variable 4 a la 5
iris %>%
  select(1:2, 4:5)
# este último ejemplo genera el mismo resultado a los generados en el apartado
# previo, donde buscamos eliminar a Petal.Length del data set
```

Y sin importar el orden, podemos usar select() con el resto de los verbos de dplyr. A continuación seleccionaremos las variables *Petal.Length*, *Petal.Width*, *Species*. Seleccionaremos aquellos tipos de iris con longitud de pétalo entre 3.5 y 4.8 y generaremos una nueva variable llamada *condicion*. Seleccionaremos solo los valores que cumplan con la condición, ordenaremos los datos de forma descendente acorde a *Petal.width*. Por último, eliminaremos la variable condición.

```r
iris %>%
  select(Petal.Length,Petal.Width, Species) %>%
  mutate(condicion = Petal.Length > 3.5 & Petal.Length < 4.8) %>%
  filter(condicion == TRUE) %>%
  arrange(desc(Petal.Width)) %>%
  select(-condicion)
```

Nota: en este ejemplo generamos una variable temporal denominada condición, para determinar aquellos tipos de iris con un rango en la longitud de pétalo. No obstante, la eliminamos al no pertenecer al data set original.

group_by()

El verbo group_by() agrupa los datos acorde a una característica. No suele modificar el data ser original, pero indica cuantos grupos puede formar a partir de la variable de interés. En el siguiente ejemplo utilizando el data set mtcars evaluaremos cuantos grupos podemos generar a partir del numero de cilindros que contiene cada automovil.

```
mtcars %>%
  group_by(cyl)
```

El data set no se modificó. Sin embargo, indica que puede agrupar los datos en 3 a partir del numero de cilindros. La función group_by() suele acompañarse de la función count() o de summarise(). En el primer caso utilizando count() podemos contar el numero de elementos que se encuentran en cada grupo. Retomaremos el ejemplo previo, agruparemos el data set mtcars acorde al numero de cilindros y después contaremos cuantos autos se encuentran en cada grupo.

```
mtcars %>%
  group_by(cyl) %>%
  count()
```

De esta forma podemos determiar que los automóviles en el data set pertenecen a uno de los tres grupos. 4, 6 y 8 cilindros. Asimismo, observamos que 11 automóviles presentan 4 cilindros, 7 automóviles con 6 cilindros y el último conformado por 14 autos que tienen 8 cilindros.

Los grupos se pueden hacer tan diversos como nosotros queramos. A continuación agruparemos los datos acorde a numero de cilindros, tipo de motor y transmisión

```
mtcars %>%
  group_by(cyl, vs, am)

# este argumento no cambia el data set, sin embargo, nos indica que hay 7 grupos
# agregaremos la función count() para saber como se conforman los grupos

mtcars %>%
  group_by(cyl, vs, am) %>%
  count()
```

Este df es diferente al original de *mtcars*. Sin embargo, podemos seguir utilizando los verbos de dplyr y manipular los datos. Agruparemos los datos por número de cilindros, tipo de motor y transmisión. Contaremos los datos, seleccionaremos los automóviles con transmisión manual e identificaremos a aquellos que tengan motor con forma de V a partir de una variable denominada motor que contenga TRUE o FALSE dependiendo el caso.

```
mtcars %>%
  group_by(cyl, vs, am) %>%
  count() %>%
  filter(am == 1) %>%
  mutate(motor = vs == 0)
```

Una vez generados los grupos, ¿cómo obtendrías el porcentaje de automoviles que pertenece a cada grupo?

```
mtcars %>%
  group_by(cyl, vs, am) %>%
  count() %>%
  filter(am == 1) %>%
  mutate(motor = vs == 0,
         porcentaje = n/13)
```

Con esta función podemos determinar que el 53% de los automóviles en el grupo generado tienen 4 cilindros, motor recto y transmisión manual.

summarise()

Este verbo colpasa la información de una variable para realizar un análisis estadístico como sumar los valores, obtener la media aritmética, mediana, desviación estándar, etc. Por lo cual debe de acompañarse de otras funciones. En el siguiente ejemplo obtendremos el promedio de los valores de *Petal.Length* en el dataset de iris.

```
iris %>%
  summarise(promedio = mean(Petal.Length))
```

Para obtener el promedio de *Petal.Length* utilizamos la función mean() dentro de summarise. Y asignamos el resultado al objeto *promedio*. De esta forma solo obtenemos el resultado de la función y eliminamos el resto de las variables de iris (colapso de información)

Nota: summarise() y summarize() reaalizan la misma función y utilizan los mismos argumentos.

Más ejemplos de summarise con *Petal.Length*

```
# desviación estándar
iris %>%
  summarise(ds = sd(Petal.Length))

# mediana
iris %>%
  summarise(mediana = median(Petal.Length))

# varianza
iris %>%
  summarise(varianza = var(Petal.Length))

# suma de los valores
iris %>%
  summarise(total = sum(Petal.Length))
```

Al igual que el resto de los verbos, summarise se puede combinar para realizar un análisis estadístico posterior a la limpieza de datos. No obstante, si utilizamos group_by() en nuestro código debemos deshacer este agrupamiento con ungroup() y colocar las variables a desagrupar. Retomaremos un ejemplo anterior y lo complementaremoss.

```r
mtcars %>%
  group_by(cyl, vs, am) %>%
  count() %>%
  filter(am == 1) %>%
  mutate(motor = vs == 0,
         porcentaje = n/13)

# si nuestro análisis es correcto, al sumar la variable porcentaje debemos
# obtner un valor igual a 1

mtcars %>%
  group_by(cyl, vs, am) %>%
  count() %>%
  filter(am == 1) %>%
  mutate(motor = vs == 0,
         porcentaje = n/13) %>%
  ungroup(cyl, vs, am) %>%
  summarise(total = sum(porcentaje))
```

Nota: Todas las funciones de las paqueterías presentes en tidyverse se pueden combinar con el pipe %>% de dplyr