Introducción gráficos y funciones

Armando Ocampo

Gráficos

Para esta clase utilizaremos la paquetería ggplot2. La cual forma parte del universo de tidyverse. Esta paquetería no forma parte de los paquetes base de R, por lo cual debemos descargarla.

install.packages("ggplot2")

Una vez descargado, llamaremos la librería para comenzar a trabajar.

library(ggplot)

Para realizar un gráfico necesitamos 3 cosas. El data set, la estética y el tipo de gráfico. El data set corresponde al conjunto de datos del cual partiremos para graficar. La estética se representa como aes() y corresponde a la información en el eje equis y ye. El tipo de gráfico está dado por la función geom_*(). Asimismo, podemos agregar una tercera variable y graficarla acorde a color o tamaño.

Realizaremos algunos gráficos con información de los data sets *iris*. Para más detalles sobre los grafos puedes checar the cheat sheet para ggplot2

En este primer ejemplo realizaremos un gráfico de puntos con información de *Sepal.Length* y *Petal.Length* de *iris*. Para comenzar a hacer graficos utilizamos la función ggplot(), y colocamos los tres elementos necesarios para realizar un gráfico.

```
ggplot(data = iris, mapping = aes(x = Sepal.Length, y = Petal.Length)) +
  geom_point()
```

Nota: las variables van sin comillas

En el gráfico anterior observamos que el data set corresponde a *iris*, la estética o aes() nos permite graficar en el eje equis los valores de *Sepal.Length* y en el eje de las y los valores de *Petal.length*. Debido a que especificamos cada elemento, el orden de los factores no altera el producto.

```
ggplot(mapping = aes(y = Petal.Length, x = Sepal.Length), data = iris) +
  geom_point()
```

No obstante, hay algunos argumentos que podemos omitir. Vamos a buscar el archivo de ayuda de ggplot

?ggplot

En este archivo observamos que hay un orden por default en la función. Primero van el data set, después aes() y dentro de aes() primero va el eje de las equis y luego el eje ye. Por lo que podemos respetar el orden de aparición de los elementos, sin ncesidad de utilizar algunos argumentos.

```
ggplot(iris, aes(Sepal.Length, Petal.Length)) +
  geom_point()
```

Los tres códigos que utilizamos generan el mismo resultado, e incluso puedes hacer una combinación entre tipo de código.

```
ggplot(iris, aes(x = Sepal.Length, y = Petal.Length)) +
  geom_point()
```

A continuación, vamos a cambiar el color de los puntos en el gráfico. El color de los puntos se puede realizar a partir de la información de una tercer variable, o de forma manual.

Como primer ejemplo, cambiaremos el color de los puntos a partir del tipo de especie. Para hacer esto, colocamos dentro de aes() la condición.

```
ggplot(iris, aes(x = Sepal.Length, y = Petal.Length, color = Species)) +
  geom_point()
```

En este jemplo el color de los puntos se modifica acorde al tipo de especie de iris. Para colorear los puntos de forma general se utiliza el argumento color dentro de la función geom_point()

```
ggplot(iris, aes(x = Sepal.Length, y = Petal.Length)) +
  geom_point(color = "red")
```

Mota1: el color en el apartado de geom_point() se coloca entre comillas al ser un vector de tipo caracter.

Nota2: el argumento color y colour se utilizan de la misma manera.

Existen otros elemntos que podemos modifcar en un gráfico. En este plot de puntos se puede modificar tamaño de puntos (size), forma (shape), transparencia (alpha). Ambos argumentos e pueden utilizar en la función aes() o geom_point() dependiendo de lo que busquemos hacer.

```
# modificando tamaño
ggplot(iris, aes(x = Sepal.Length, y = Petal.Length, size = Species)) +
  geom_point()

ggplot(iris, aes(x = Sepal.Length, y = Petal.Length)) +
  geom_point(size = 5)

# modificando forma
ggplot(iris, aes(x = Sepal.Length, y = Petal.Length, shape = Species)) +
  geom_point()

ggplot(iris, aes(x = Sepal.Length, y = Petal.Length)) +
  geom_point(shape = 5)

# modificando transparencia
ggplot(iris, aes(x = Sepal.Length, y = Petal.Length, alpha = Species)) +
  geom_point()

ggplot(iris, aes(x = Sepal.Length, y = Petal.Length)) +
  geom_point(alpha = 0.5)
```

Como en la mayoría de las funciones de R, podemos combinar los argumentos *color*, *shape*, *size*, *alpha* entre las funciones aes() y geom_*(). No obstante, no debes poner el mismo argumento en las dos funciones en la misma linea de código.

```r
# coloreando acorde al tipo de specie y cambiando el tamaño
ggplot(iris, aes(x = Sepal.Length, y = Petal.Length, color = Species)) +
  geom_point(size = 5)

# Forma de los puntos acorde a especie y color azul para todos los puntos
ggplot(iris, aes(x = Sepal.Length, y = Petal.Length, shape = Species)) +
  geom_point(color = "blue")

# color y forma acorde a especie
ggplot(iris, aes(x = Sepal.Length, y = Petal.Length, colour = Species, shape = Species)) +
  geom_point()

# tamaño de 4 y color verde para todos los puntos
ggplot(iris, aes(x = Sepal.Length, y = Petal.Length)) +
  geom_point(size = 4, color = "green")
```

ggplot2 y dplyr

Las funciones de las paqueterías que se encuentran en tidyverse se pueden utilizar en conjunto a la función %>% de dplyr. En este apartado utilizaremos lo aprendido en la clase de dplyr y lo combinaremos con la información de ggplot.

Antes de comenzar debemos llamar a nuestra paquetería de trabajo.

library(dplyr)

Comenzaremos a filtrar la información de iris acorde a una longitud de sépalo entre 4.5 y 7. Con un ancho de pétalo entre 0.5 y 1.8.

```r
iris %>%
  filter(Sepal.Length > 4.5 & Sepal.Length < 7,
         Petal.Width > 0.4 & Petal.Width < 1.8)
```

Con esta información realizaremos un gráfico de puntos comparando *Sepal.Length* en el eje de las equis y *Petal.Length* en el eje ye, coloreando los puntos acorde al tipo de especie, dando un tamaño de 4 para todos los puntos y con forma 2.

```r
iris %>%
  filter(Sepal.Length > 4.5 & Sepal.Length < 7,
         Petal.Width > 0.4 & Petal.Width < 1.8) %>%
  ggplot(aes(x = Sepal.Length, y = Petal.Length, color = Species)) +
  geom_point(size = 4, shape = 2)
```

Nota: dentro de la función ggplot(), colocamos de forma directa aes(), ya que la información del data set se encuentra en la parte superior del código.

Gráfico de barras

El gráfico de barras nos permite evaluar frecuencias o relaciones, acorde al numero de repeticiones de un elemento. Para este gráfico se utiliza la función geom_col(). Con información de *iris* realizaremos una tabla de frecuecias para determinar el numero de especies que hay de cada tipo y posteriormente realizaremos y gráfico de barras.

```r
iris %>%
  group_by(Species) %>%
  count() %>%
  ggplot(aes(x = Species, y = n))+
  geom_col()
```

Nota: a pesar de que la variable n no existe en el data set original de *iris*, esta se genera al realizar el agrupamiento y conteo de los datos.

Para agregar color a cada barra se utiliza el argumento fill = , esta es una variación. Las barras en geom_col() pueden tener un color para el borde y un color de relleno. Vamos a utilizar los argumentos fill y color para observar las diferencias.

```r
# utilizando fill
iris %>%
  group_by(Species) %>%
  count() %>%
  ggplot(aes(x = Species, y = n, fill = Species))+
  geom_col()

# utilizando color
iris %>%
  group_by(Species) %>%
  count() %>%
  ggplot(aes(x = Species, y = n, color = Species))+
  geom_col()
```

No obstante, puedes combinar ambos argumentos y diseñar tus gráficos de forma libre

```r
iris %>%
  group_by(Species) %>%
  count() %>%
  ggplot(aes(x = Species, y = n, fill = Species))+
  geom_col(color = "red")
```

Al igual que en los apartados previos, el color puede aplicarse de forma general y no solo acorde a una variable. En este aspecto, el argumento fill = se coloca dentro de la función geom_col(). A continuación rellenaremos nuestras barras con el color *"dodgerblue"*.

```r
iris %>%
  group_by(Species) %>%
  count() %>%
  ggplot(aes(x = Species, y = n))+
  geom_col(fill = "dodgerblue")
```

Continuando con la combinación de verbos de dplyr, realizaremos una tabla de porcentajes. Filtraremos los datos iris con una longitu de sepalo entre 4.5-7 y un ancho de pétalo entre 0.4-1.8. Posteriormente, aruparemos los datos acorde al tipo de iris, contaremos cuantos elementos existen de cada grupo, obtendremos la relación y porcentaje de los datos. Por último, graficaremos la información de porcentaje con una gráfica de barras donde el color de cada barra estará dado por la especie.

```r
iris %>%
  filter(Sepal.Length > 4.5 & Sepal.Length < 7,
         Petal.Width > 0.4 & Petal.Width < 1.8) %>%
  group_by(Species) %>%
  count() %>%
  mutate(relacion = n/54,
         porcentaje = relacion * 100) %>%
  ggplot(aes(x = Species, y = porcentaje, fill = Species)) +
  geom_col()
```

Funciones

Las funciones son elementos que nos permiten realizar acciones dentro del lenguaje de R. Cada función contiene argumentos, que le permite a la misma ejecutar cada acción. A su vez, un conjunto de funciones se almacena en librerias.

Para conocer los argumentos de cada función se utiliza la función args(). A continuación observaremos los argumentos de la función mean().

```r
args(mean)
```

Para obtener la documentación de cada función se utiliza la función help() o el signo de interogación. En este apartado se enecuentra una descripción detallada de cada función, la definición de cada argumento y algunos ejemplos.

```r
help(mean)
?mean
```

Es probable que ya exista una función para el problema que tengas. No obstante, ¿cómo puedo realizar una función personalizada?. En este aspecto, puedes crear tu propia función, esto a partir de la función function(). Para esto checaremos el esqueleto básico de una función.

```r
nombre_funcion <- function(arg1, arg2, argn){
  cuerpo de la funcion
}
```

El nombre de la función como cualquier objeto de R debe seguir ciertas reglas. No debe ser un número aislado, no debe coincidir con el nombre de una función ya establecida, ni tener espacios.

El número y nombre de cada argumento es libre, siempre y cuando se respeten las reglas de escritura.

La primer función que generaremos se denominará mi primer función y tendra el objetivo de tomar cualquier numero y sumarle 3.

```r
mi_primer_funcion <- function(x){
  x + 3
}

mi_primer_funcion(x = 4)
```

En esta función, cada vez que le asignamos un valor al argumento x, a este se le sumarán 3 unidades. Por lo que podemos utilizarla con el número que queramos.

```r
mi_primer_funcion(5)

mi_primer_funcion(15)

mi_primer_funcion("a")
```

Los argumentos de una función pueden estar definidos o no. En el ejemplo previo x no está definido. Sin embargo, podemos generar argumentos con un valor preestablecido en la función. No obstante, este argumento se puede modificar al utilizar la función. Haremos una segunda funcion con un argumento definido.

```r
mi_segunda_funcion <- function(x, y=8){
  x + y + 5
}

# en esta funcion tenemos dos argumentos, "x" y "y". Esta funcion se puede
# utilizar solo asignandole un valor al argumento "x"

mi_segunda_funcion(x = 5)
mi_segunda_funcion(x = 20)

# esto por que el argumento "y" ya tiene un valor definido. Sin embargo,
# podemos modificar el valor del argumento "y", sin modificar el esqueleto
# de la funcion

mi_segunda_funcion(x = 5, y = 30)
```

En mi_segunda_funcion() tenemos un argumento definido, el argumento “y”. Y un argumento no definido, el argumento “x”

Dentro de las funciones podemos generar objetos para realizar una acción. Estos objetos no se guardarán en el ambiente, solo pertenecerán al esqueleto de la función. En el siguiente ejemplo generaremos una función con el argumento “x”. Dentro de la función se generará un objeto denominado resultado con la suma de x + 4 . Por último agregaremos una línea de código con 20+ resultado.

```r
mi_tercera_funcion <- function(x){
  resultado <- x + 4
  20 + resultado
}

mi_tercera_funcion(5)
```

Nota: la salida de una función siempre será el último elemento de la línea de código

Dentro de una función se pueden agregar tantos objetos como sea necesario. No existe un límite.

```r
cuarta_funcion <- function(x){
  objeto1 <- x * x
  objeto2 <- x + 5
  objeto3 <- 20 + 8

  objeto1 + objeto2 + objeto3
}
```

```
cuarta_funcion(5)
```

Dentro de la funcion puedes colocar cualquier línea de código. Y utilizar funciones ya establecidas, como los verbos de dplyr y ggplot.

A continuación, realizaremos una función para filtrar los tipos de iris del df *iris*.

```
filtrando <- function(x){
  iris %>%
    filter(Species == x)
}


filtrando(x = "setosa")
filtrando(x = "versicolor")
filtrando(x = "virginica")
```

De hecho, el resultado puede ser un gráfico. En este ejemplo realizaremos una función para filtrar los datos por tipo de iris, y posteriormente hacer un gráfico de puntos con información de *Sepal.Length* en el eje equis y Petal.Length en el eje ye. Cada punto tendrá un tamaño de 3 y color rojo

```
funcion_grafico <- function(x){
  iris %>%
    filter(Species == x) %>%
    ggplot(aes(Sepal.Length, Petal.Length)) +
  geom_point(size = 3, color = "red")
}


funcion_grafico(x = "setosa")
funcion_grafico(x = "virginica")
funcion_grafico(x = "versicolor")
```

Y cada elemento del codigo anterior puede ser modificado con un argumento. Por ejemplo, agregaremos un argumento para el color de los puntos.

```
graficando_ando <- function(x,y){
  iris %>%
    filter(Species == x) %>%
    ggplot(aes(Sepal.Length, Petal.Length)) +
  geom_point(size = 3, color = y)
}

graficando_ando(x = "setosa", y = "forestgreen")
graficando_ando(x = "versicolor", y = "salmon")
graficando_ando(x = "virginica", y = "dodgerblue")
```

TAREA:

1. Realiza una función para calcular el índice de masa corporal
2. Automatiza el siguiente código para que el usuario solo mueda modificar el rango de *Petal.Length*, y nombra a esta función como funcion_tarea()

```
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) %>%
  ggplot(aes(Petal.Length, Petal.Width, color = Species))+
  geom_point()
```

Graficar sin código

Existen dos paqueterias que nos permiten graficar sin necesidad de conocer sobre código de programación estas son **ggThemeAssist** y **esquisse**. Estas paqueterias no pertenecen a las funciones base de R por lo que debemos descargarlas.

install.packages("ggThemeAssist")

install.packages("esquisse")

Comenzaremos con ggThemeAssist. Vamos a llamar a nuestra libreria

library(ggThemeAssist)

Una vez realizado esto, necesitamos colocar los tres elementos básicos para realizar un plot y nombrar este plot como objeto. A continuación utilizamos la función *ggThemeAssistGadget()* y colocamos el nombre del plot.

```
plot_chevere <- ggplot(iris, aes(Sepal.Length, Petal.Length)) +
  geom_point()

ggThemeAssistGadget(plot_chevere)
```

Con esta herramienta podemos modificar nuestro gráfico. Al finalizar genera los cambios necesarios en el código para graficar el plot que hemos desarrollado.

esquisse es un GUI (graphical user interface) que nos permite cargar el data set del ambiente o de nuestro sistema. Este ambiente gráfico permite filtrar los datos y realizar nuestro gráfico. En primer lugar llamamos a nuestra libreria y después utilizamos la función *squisser()* para utilizar la interfaz gráfica.

library(esquisse)

```
esquisser()
```