Módulo V

Taller de IA y CD

Maciel-Cruz, EJ

2025-06-12

Contenidos

1 Introducción

La parte en la que realmente brilla R son sus gráficos, ya que tiene un amplio abanico de posibilidades para muchos tipos de esquemas, además de la tremenda personalización que se puede llevar a cabo con los gráficos de base.

Por si fuera poco, la librería más popular, que es ggplot2, incrementa muchísimo las opciones y la personalización de dichos gráficos, además de que ya cuenta con adiciones al paquete de base para incrementar argumentos o dar otras opciones.

Antes de empezar de lleno con el módulo, vamos a ver la sintaxis base de cada una de las librerías que usaremos.

No olvidemos cargar nuestras librerías:

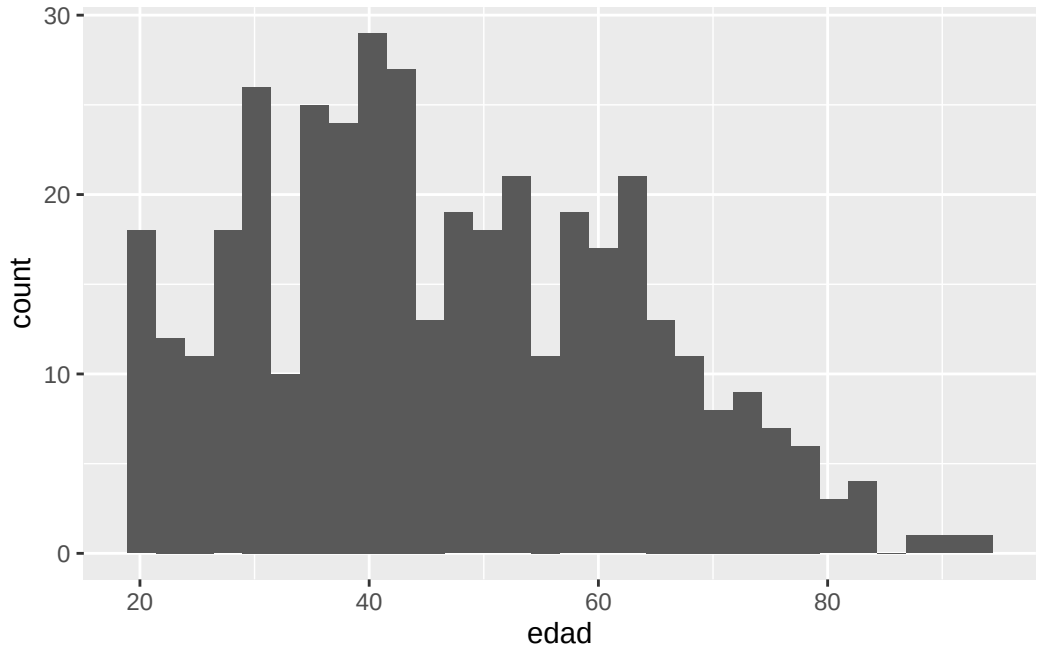
```
pacman::p_load(tidyverse, GGally, esquisse, ggThemeAssist,summarytools, readxl, viridis, hrbr
```

1.1 ggplot2

La idea de ggplot2 es tener una especie de múltiples lienzos en los que se pueden agregar o quitar capas a la base, que corresponde a grupos, estética, etiquetas, nombres, etc. De manera general, un gráfico se produce de la siguiente manera:

```
BD2_R <- read_excel("BD2_R.xlsx")
BD2 <- BD2_R
BD2 %>%
  ggplot(aes(x=edad)) +
  geom_histogram()
```

```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



La idea detrás consiste en lo siguiente: la primera línea selecciona la base de datos a trabajar; la segunda línea llama a la función gráfica (pone el lienzo), y ponemos el argumento de estéticos (aquí seleccionamos las variables a trabajar, pueden ser de una a cinco, sin embargo, más de 3 suele dar problemas); en la tercera línea agregamos el tipo de gráfico que queremos, que en este caso es un histograma.

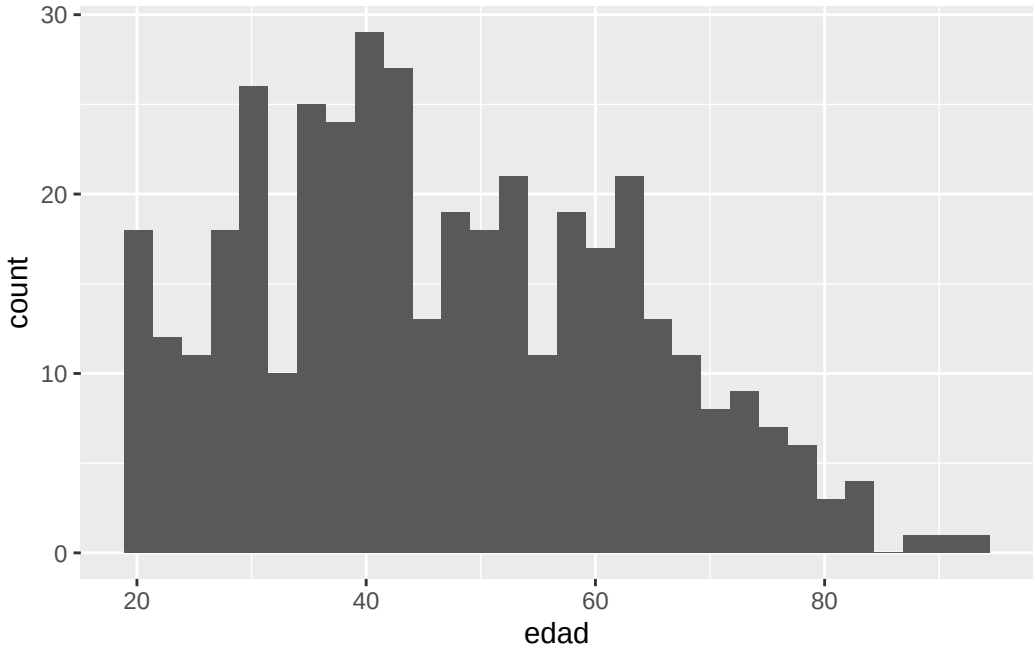
Y listo, esa es la sintaxis más simple para generar gráficos; sin embargo, debemos considerar la naturaleza del tipo de datos que sea congruente con el tipo de gráfico, algunos darán error, otros serán poco representativos, otros requieren de al menos dos variables, etc.

Otro punto importante, es que aquí añadimos funciones con el símbolo de "+", a diferencia del operador de pipa, ese solo lo utilizamos en la primera sección de código, puede ser un poco difícil al principio, pero es de utilidad recordar lo siguiente: para **más** y mejores gráficos, usamos "+", el resto que se vaya por una pipa "%>%", sé que es mala mnemotecnia, pero a mí me sirve.

Finalmente, no hay que olvidar que la sintaxis anterior solo genera el gráfico y no lo guarda en un objeto, para ello, debemos de agregar el nombre del objeto y el operador de asignación, por ejemplo:

```
plot <- BD2 %>%
  ggplot(aes(x=edad)) +
  geom_histogram()
plot
```

```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Y listo, así lo guardamos, con el nombre que deseemos. Tip: los gráficos los podemos mostrar solamente con el hecho de ponerlos en la consola y correr la celda.

1.2 esquisse

Este paquete es una interfaz de clicks que nos facilita mucho la generación de gráficos, es intuitivo y selecciona qué gráfico utilizar al considerar cuál es el más adecuado para nuestro tipo de datos; sin embargo, podemos hacer algunas modificaciones.

Ya que es una interfaz por clics, la mejor explicación es en vivo. Se abre con el siguiente código:

```
pacman::p_load(esquisse)
esquisser(BD2)
```

Cabe mencionar que podemos obtener el código base para hacer el gráfico para posteriormente hacer algunos ajustes que son más fáciles con R.

2 Gráficos específicos

Ya que tenemos todo en orden, vamos a empezar con el primer tipo de gráfico.

2.1 Distribución

De forma general, un gráfico de distribución tiene como objetivo graficar la forma en la que los datos se reparten a través de un determinado eje. Empecemos con el primer tipo.

2.1.1 Violín

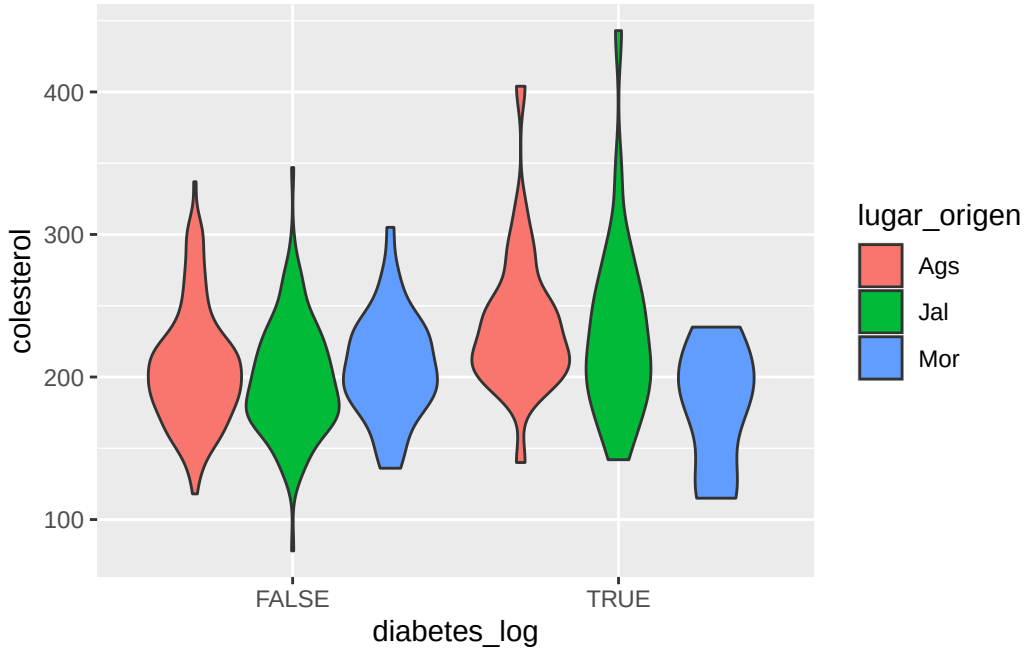
Si bien, el gráfico de violin no es el gráfico inicial para muchos (es el de caja y bigotes, o "boxplot"), lo pongo en primer lugar porque es el que arroja más información y está siendo mayormente utilizado para muchos artículos, además, hay un pequeño video que explica perfectamente este punto, lo dejo aquí. Vamos a hacer un ejemplo:

```
plot <- BD2 %>%
  drop_na()  %>%
  ggplot(aes(x = diabetes_log, y = colesterol)) +
  geom_violin()
plot
```

Aquí observamos la distribución de una variable numérica en dos o más grupos.

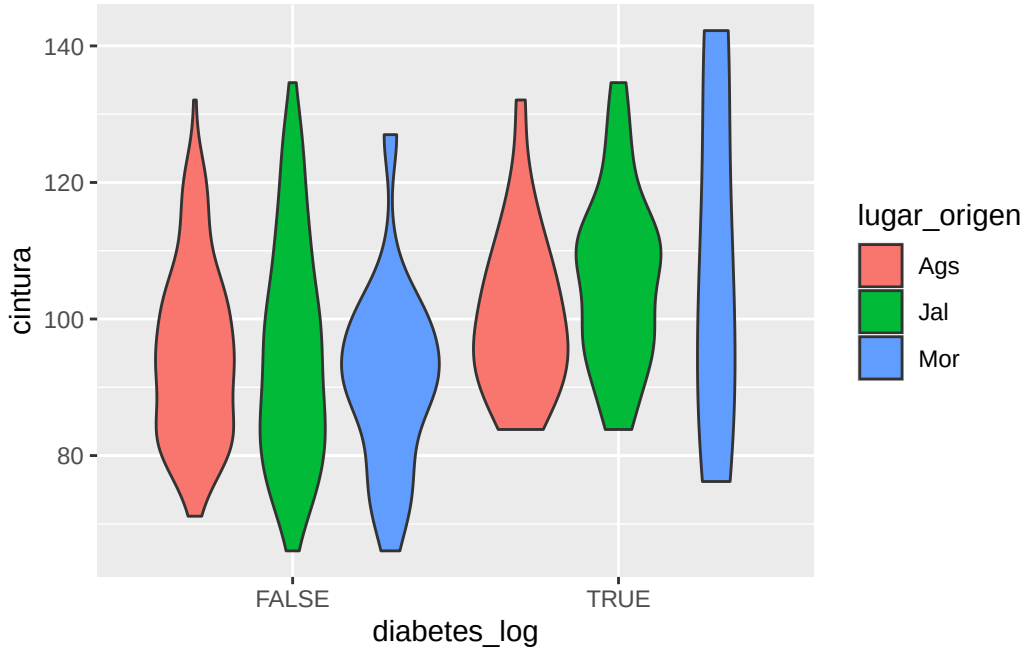
La función de "drop_na" tiene como objetivo eliminar los datos vacíos o nulos de nuestra base, y solo gráfica lo que sí tenemos. La podemos poner como F o eliminarla para que también haga los diagramas de violin.

```
plot <- BD2 %>%
  drop_na()  %>%
  ggplot(aes(x=diabetes_log, y = colesterol, fill = lugar_origen)) +
  geom_violin()
plot
```



El argumento "fill", que es relleno, nos da una opción de personalización, si utilizamos la misma variable por la que agrupamos, nos va a dar un color para cada tipo grupo, sin embargo, podemos elegir una tercera variable para agrupar. Vamos a intentarlo:

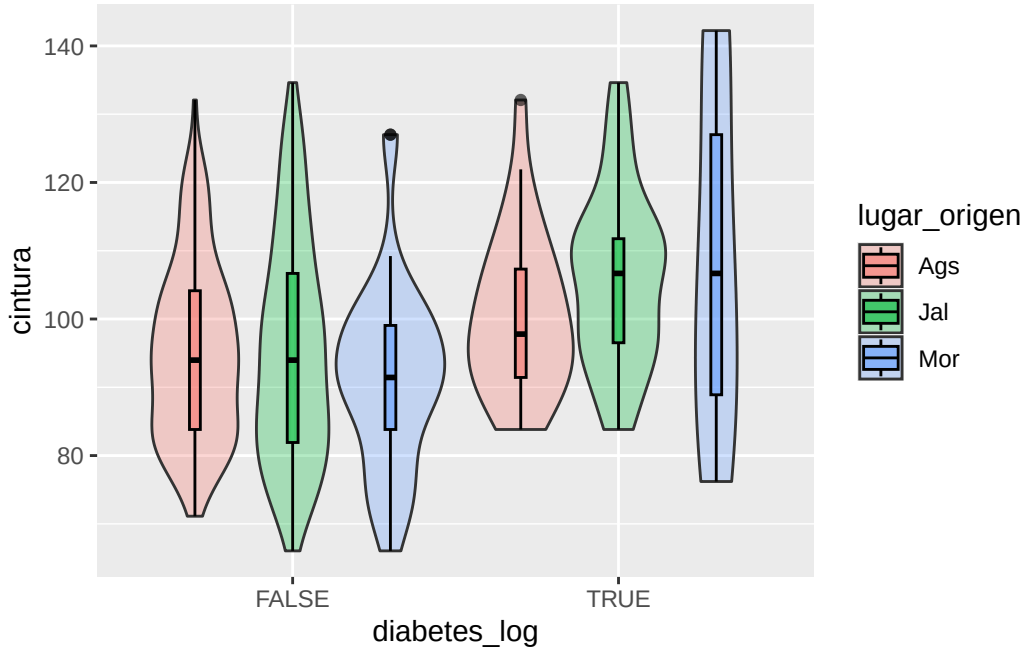
```
plot <- BD2 %>%
  drop_na()  %>%
  ggplot(aes(x=diabetes_log, y = cintura, fill = lugar_origen)) +
  geom_violin()
plot
```



Ahora, tenemos un tercer grupo que es "lugar_origen", aquí podemos observar que hay cambios en la distribución según si tienen diabetes, lugar de origen y el perímetro de su cintura.

Supongamos que queremos anidar un boxplot:

```
plot <- BD2 %>%
  drop_na()  %>%
  ggplot(aes(x=diabetes_log, y = cintura, fill = lugar_origen)) +
  geom_violin(width=1, position = position_dodge(0.9), alpha = 0.3) +
  geom_boxplot(width=0.1, color="black", alpha=0.6, position = position_dodge(0.9))
plot
```



Desglosemos el código que no hemos visto por partes: - “width”: como su nombre lo dice, es el ancho del diagrama, de violín o boxplot, según corresponda. - “position”: aquí agregamos el argumento “dodge”, que sirve para “esquivar” el otro gráfico y no se empalme por completo, agregamos un 0.9 para que se vea bien. - “alpha”: es la transparencia, es bastante útil para darle un tinte más estético a muchos de nuestros gráficos, más cerca a cero es más transparente.

> Tip
>
> Tip: estéticamente hablando, es más agradable ir aumentando las transparencias de los gráficos más grandes a los más pequeños cuando van anidados.

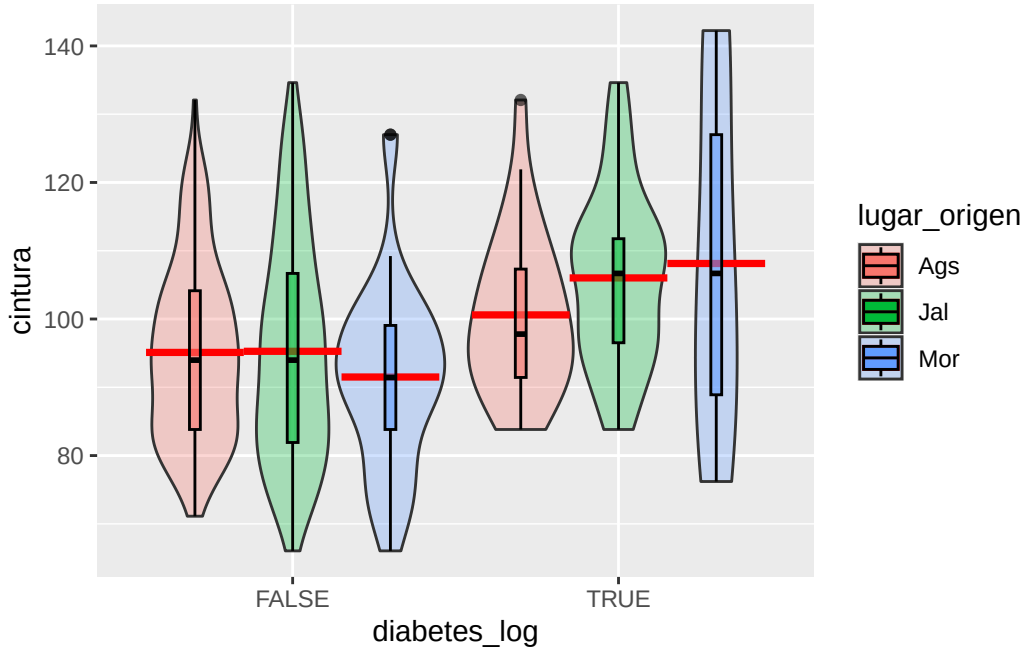
Ahora, supongamos que queremos agregar algunos valores como la media y la mediana.

```
plot <- BD2 %>%
  drop_na() %>%
  ggplot(
    aes(
      x = diabetes_log,
      y = cintura,
      fill = lugar_origen
    )
  ) +
```

```
  geom_violin(
    width = 1,
    position = position_dodge(0.9),
    alpha = 0.3
  ) +
  stat_summary(
    fun = "mean",
    geom = "crossbar",
    color = "red",
    size = 0.51,
    position = position_dodge(0.9)
  ) +
  geom_boxplot(
    width = 0.1,
    color = "black",
    alpha = 0.6,
    position = position_dodge(0.9)
  )
```

```
Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
i Please use `linewidth` instead.
```

```
plot
```



Primeramente, conforme se van haciendo más grandes nuestras líneas, debemos mantener un orden. Lo anterior, es sugerencia de programadores, dejar una línea por función, y cierre de paréntesis por línea, esto nos permite tener una lectura más ágil y organizada. Cada argumento se separa por coma, en cada coma, agregar un salto de línea, eso también nos permite tener organización.

No siempre lo utilizo, solamente cuando trabajaré en códigos por mucho tiempo o que sean trabajos de larga duración, o que vaya a compartir, suelo hacer lo que me resulta en menor tiempo y con la misma calidad de trabajo :).

Ahora, vamos a describir las funciones adicionales:

- "statsummary": es una función que permite agregar algunas estadísticas dentro de nuestro mismo lienzo o gráfico.
  - "fun": la función estadística que queramos, es este caso "mean".
  - "geom": el tipo de gráfico que queremos, pueden ser puntos o líneas, que es "crossbar".
  - "color": autodescriptivo.
  - "size": autodescriptivo.
  - "position": lo mismo que se ha descrito, para que se ajuste y no se empalme.

2.1.2 Densidad

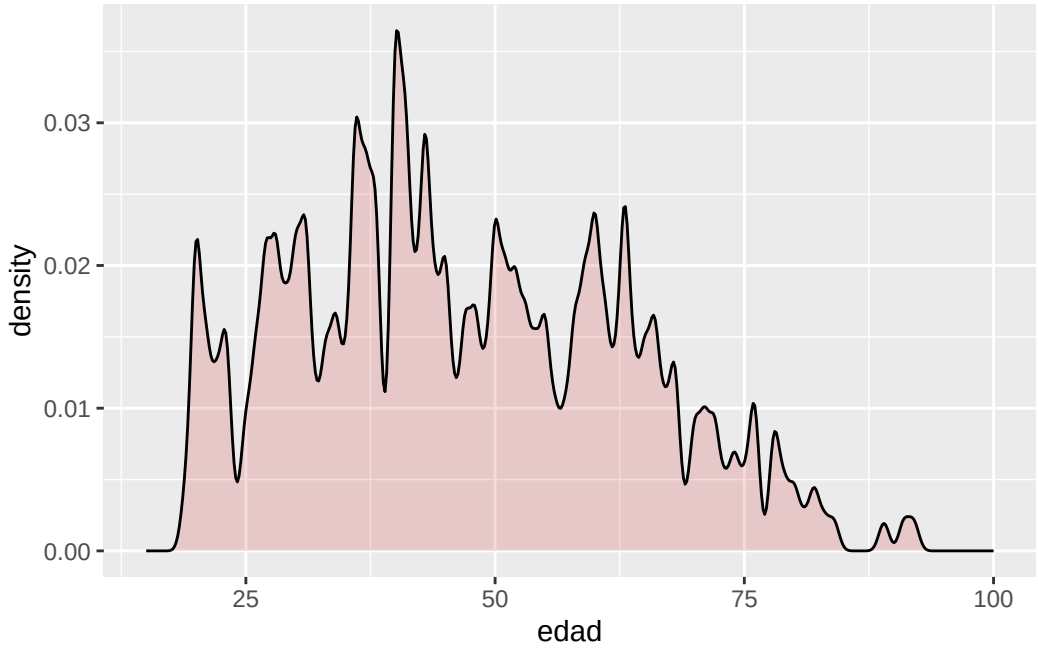
El gráfico de densidad tiene la cualidad de ser una representación de una variable numérica utilizando un kernel especial que es una función de probabilidad, es decir, que estima la distribución de datos esperados acorde a los observados en una distribución no paramétrica.

Vamos a hacer un ejemplo:

```
plot <- BD2 %>%
  ggplot(
    aes(
      x = edad
    )
  ) +
  geom_density(
    fill = "red3",
    alpha = 0.15,
    bw = 0.5
  ) +
  scale_x_continuous(
    breaks = pretty(
      BD2$edad, n = 4
    )
  ) +
  xlim(15, 100)
```

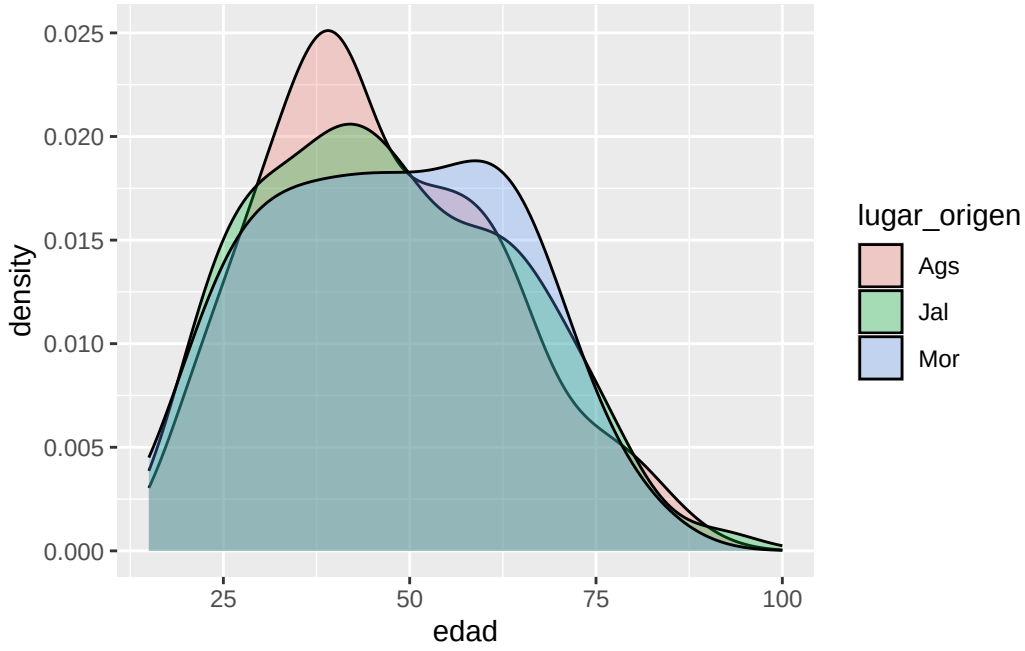
```
Scale for x is already present.
Adding another scale for x, which will replace the existing scale.
```

```
plot
```



- "bw": es el ancho de los bloques, a mayor número, veremos una distribución más suave de la densidad.
- "scale_x_continuous": nos permite seleccionar el número de separadores en el eje de las "x", al utilizar "pretty" seleciona automáticamente la cantidad de divisiones en múltiplos fáciles.
- "xlim": corresponde a los extremos de visualización, ya que el mínimo de edad en años de la base es de 19 y el máximo de 92, los rangos de 15 a 100 permitirían visualizar todos nuestros datos.

```
plot <- BD2 %>%
  ggplot(
    aes(
      x=edad,
      fill = lugar_origen
      )
    ) +
  geom_density(
    alpha = 0.3
    ) +
  xlim(15, 100)
plot
```

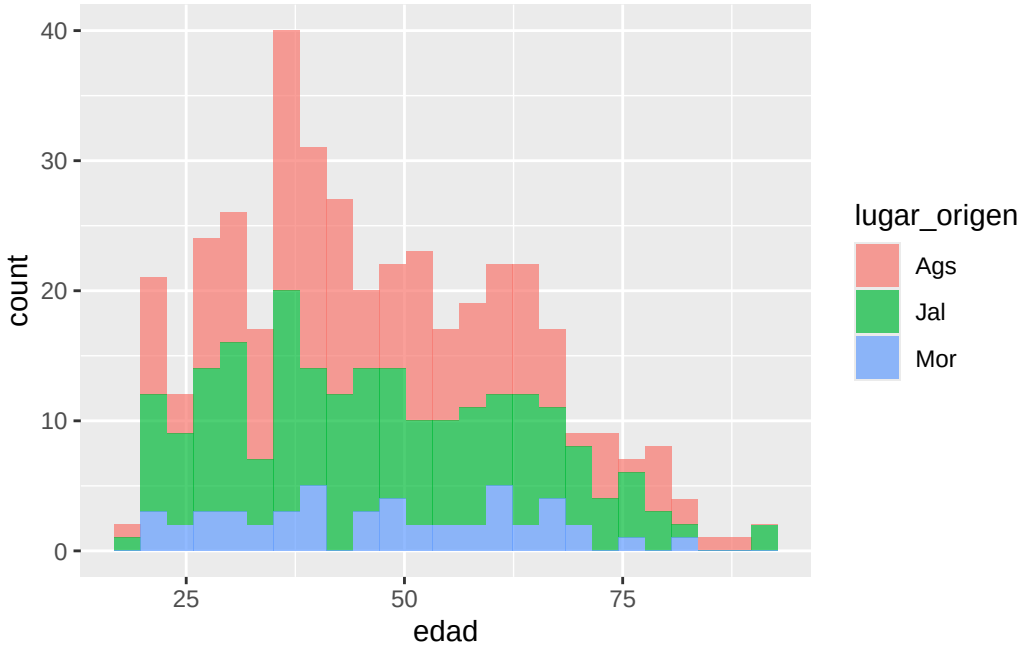


En este tipo de gráfico, solamente hicimos un ajuste, agregamos el argumento "fill", en el cuál podemos agregar un grupo adicional a nuestro lienzo y los va a separar con colores predeterminados, en este caso, por lugar de origen.

2.1.3 Histograma

El histograma es el gráfico más empleado para visualizar la distribución de una variable numérica. Vamos a hacer el mismo ejemplo que el anterior:

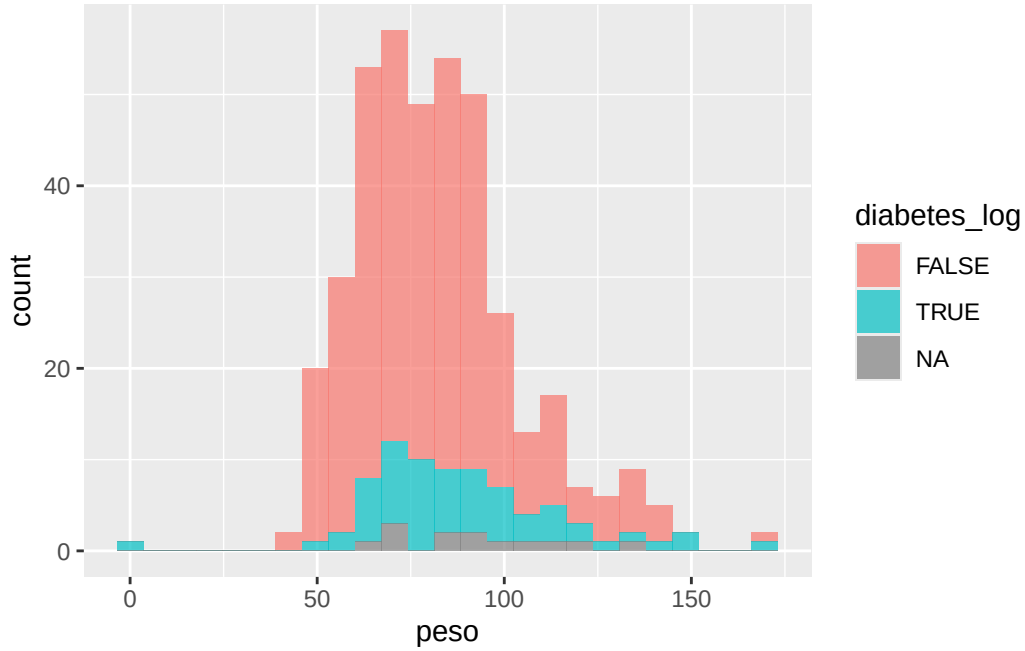
```
plot <- BD2 %>%
  ggplot(
    aes(
      x=edad,
      fill = lugar_origen
      )
    ) +
  geom_histogram(
    alpha = 0.7,
    bins = 25
    )
plot
```



Utilizamos el mismo código, pero modificamos el "geom_density" por "geom_histogram", y eliminamos "xlim", ya que el histograma se ajusta automáticamente. Agregamos un nuevo argumento, "bins" con el que decidimos el número de casilleros a elegir.

Hagamos otro ejemplo:

```
plot <- BD2 %>%
  ggplot(
    aes(
      x = peso,
      fill = diabetes_log
      )
    ) +
  geom_histogram(
    alpha = 0.7,
    bins = 25
    )
plot
```



En este ejemplo, utilizamos exactamente el mismo código previo, pero solo modificamos los estéticos, es decir, en el eje de las "x" graficamos el peso, y lo agrupamos ("fill") por diabetes_log.

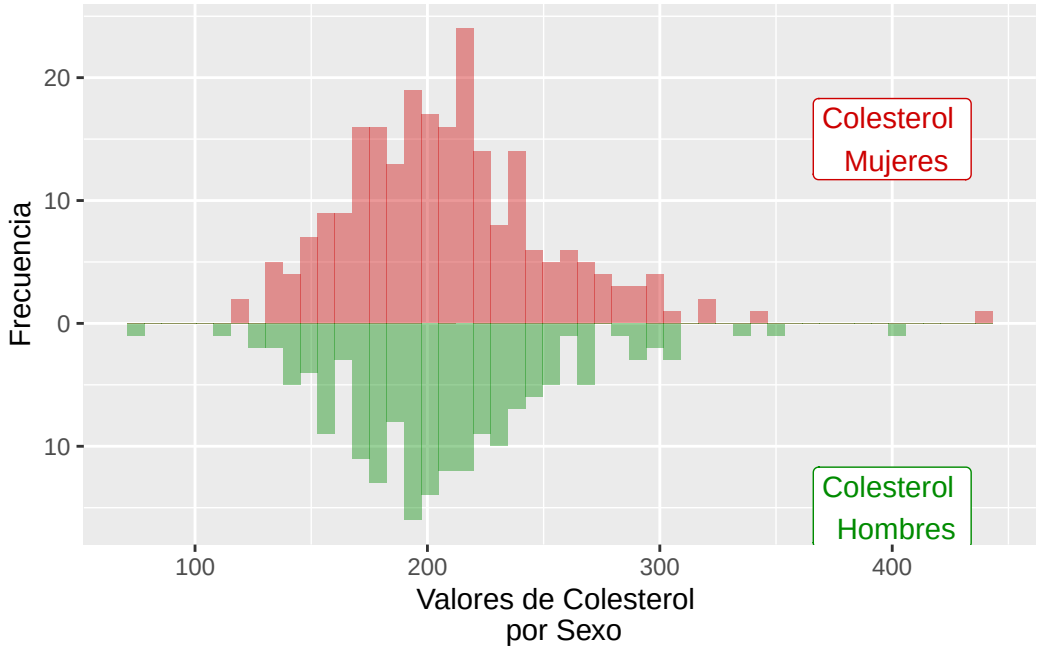
Vamos a crear un histograma en espejo con algunas funciones un poco avanzadas, pero que son bastante útiles y reproducibles para otras partes de nuestros códigos.

```
plot <- ggplot() +
  geom_histogram(
    data = subset(BD2, sexo == "mujer"),
    aes(x = colesterol, y = ..count..),
    fill = "red3", bins = 50, alpha = 0.4
  ) +
  geom_label(
    aes(x = 400, y = 15, label = "Colesterol \n Mujeres"),
    color = "red3"
  ) +
  geom_histogram(
    data = subset(BD2, sexo == "hombre"),
    aes(x = colesterol, y = -..count..),
    fill = "green4", bins = 50, alpha = 0.4
  ) +
  geom_label(
    aes(x = 400, y = -15, label = "Colesterol \n Hombres"),
```

```
    color = "green4"
  ) +
  xlab("Valores de Colesterol \n por Sexo") +
  ylab("Frecuencia") +
  scale_y_continuous(labels = abs)
plot
```

```
Warning: The dot-dot notation (`..count..`) was deprecated in ggplot2 3.4.0.
i Please use `after_stat(count)` instead.
```

```
Warning: Removed 1 row containing non-finite outside the scale range
(`stat_bin()`).
```



Este ejemplo es largo, y para no complicarlo, lo dejaré con este tipo de lectura.

En el ejemplo anterior, en vez llamar a la base de datos como estamos acostumbrados, hacemos un subset de la base con función de…. si, así es "subset".

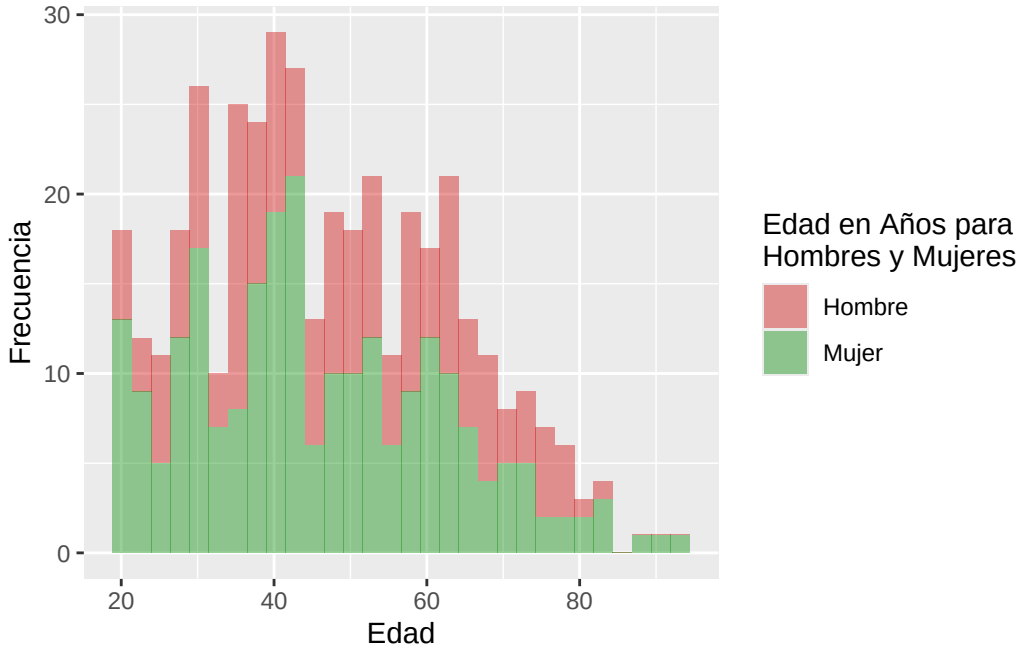
Vamos a desglosar el código por partes:

- "data = subset(BD2, sexo ==”hombre”)“: usamos la función de data para seleccionar los datos a trabajar. Usamos el argumento subset para hacer filtros.

- “aes(x = colesterol, y = ..count..),”: la única adición es que en el eje de las “y” pedimos que haga el conteo de la variable llamada al ponerlo en doble punto.
- “label”: argumento adicional que nos permite agregar la etiqueta que nosotros queramos, el símbolo “\n’ ’ nos permite hacer un salto de línea.
- “xlab” y “ylab”: poner leyendas en el eje de las “x” y “y”, respectivamente.
- “scale_y_continuous(labels = abs)”: con esto impedimos que las marcas de las frecuencias en el eje de las “y” sean negativos en la parte inferior del gráfico.

Hagamos algunas modificaciones del formato del histograma con otro ejemplo:

```
plot <- BD2 %>%
  ggplot(
    aes(
      x = edad,
      fill = sexo
      )
    ) +
  geom_histogram(
    alpha = 0.4,
    bins = 30,
    position = 'stack'
    ) +
  scale_fill_manual(
    values = c("red3",
               "green4"
               ),
    labels = c("Hombre",
               "Mujer"
               )
    ) +
  labs(
    fill = "Edad en Años para \nHombres y Mujeres"
    ) +
  xlab("Edad") +
  ylab("Frecuencia")
plot
```



En vez de hacerlo en espejo, podemos la posición de los casilleros del histograma, esto se logra con el argumento "position" y pasando diferentes opciones, se puede utilizar "dodge", "identity", "stack", y "fill", que son comúnmente utilizados.

Además, para agregar colores de forma manual, utilizamos la función de "scale_fill_manual", el argumento "values", concatenamos el número de colores que queremos (que sea congruente con el número de grupos que vamos a visualizar), ya sea con diferentes colores ya conocidos en "ggplot2", o en el formato hex, que se puede buscar aquí.

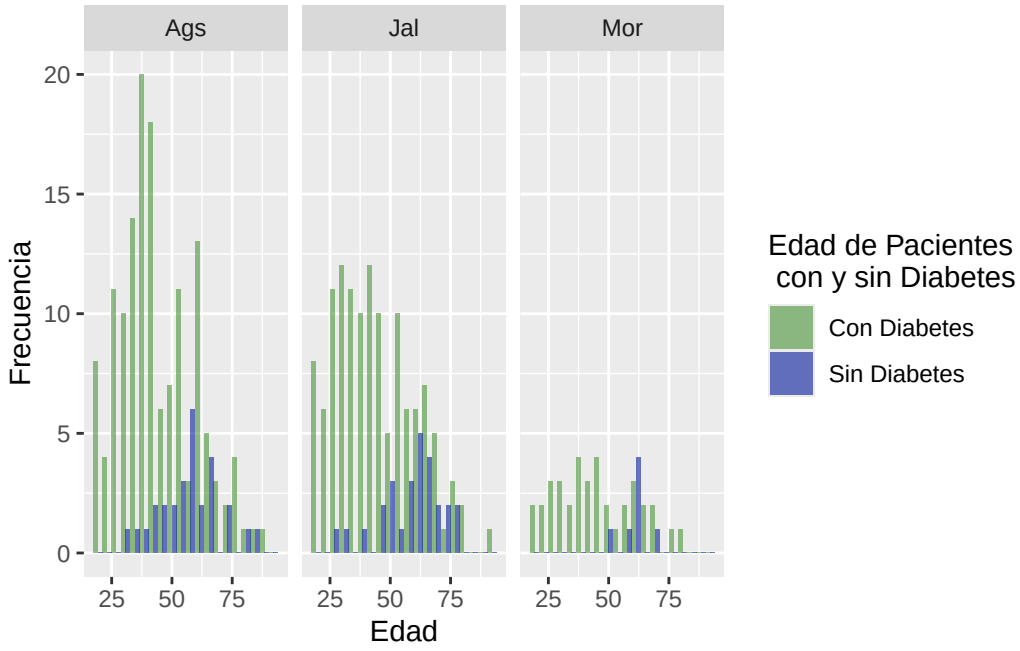
También podemos agregar diferentes etiquetas dentro de esta función con el argumento "labels".

> **Tip**
>
> Notarás que no dejé un espacio después del símbolo de salto de línea, si lo dejara, se vería también en el gráfico.

Asumamos que no nos gusta que tengamos ambos histogramas en un solo gráfico, o tenemos muchos grupos que visualizar y queremos varios histogramas. En vez de estar repetiendo el código, podemos crear múltiples histogramas con una sola función.

```
plot <- BD2 %>%
  drop_na() %>%
  ggplot(
```

```
    aes(
      x = edad,
      fill = diabetes_log
      )
    ) +
  geom_histogram(
    alpha = 0.6,
    bins=20,
    position = 'dodge'
    ) +
  scale_fill_manual(
    values = c("#4a9437cb",
               "#051c9eff"
               ),
    labels = c("Con Diabetes",
               "Sin Diabetes"
               )
    ) +
  labs(
    fill =
      "Edad de Pacientes \n con y sin Diabetes"
    ) +
  facet_wrap(
    ~lugar_origen
    ) +
  xlab("Edad") +
  ylab("Frecuencia")
plot
```



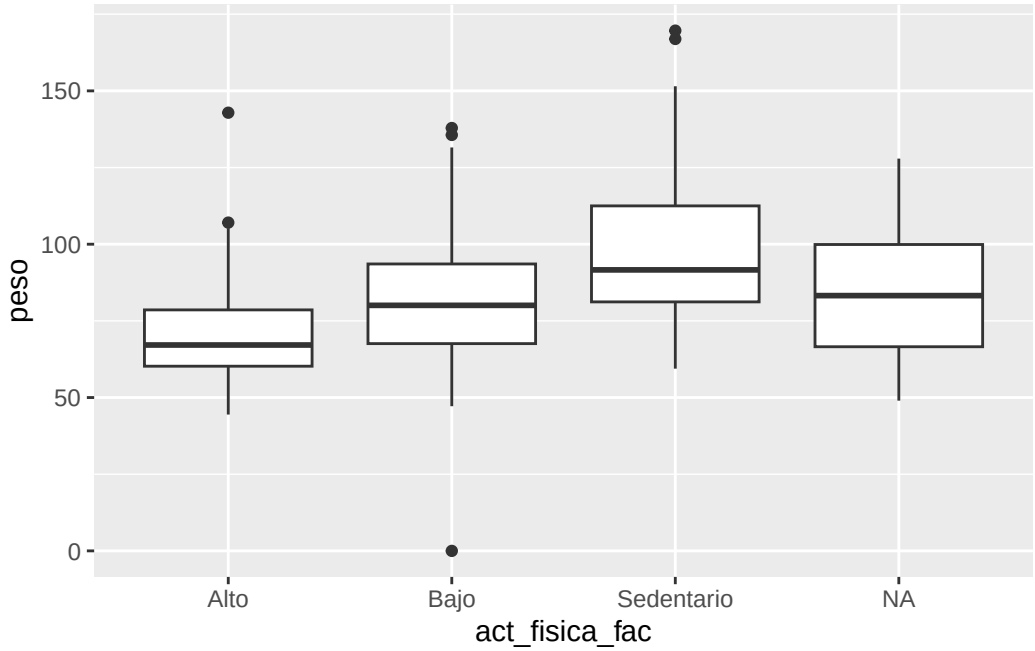
Y listo, ahora separamos los histogramas. Nótese que se puede utilizar para otras variables, por ejemplo, actividad física.

2.1.4 Boxplot

Por último, un gráfico ampliamente utilizado para representar la distribución de una variable numérica, que se puede agrupar por variables, si así de desea.

La generación de un boxplot, o gráfico de caja y bigotes, tiene el mismo método que cualquier gráfico de ggplot2, la diferencia, es que aquí requerimos dos ejes, un "x" y "y". Vamos a generar un boxplot simple y sencillo.

```
plot <- BD2 %>%
  ggplot(
    aes(
      x = act_fisica_fac,
      y = peso
      )
    ) +
  geom_boxplot(
  )
plot
```



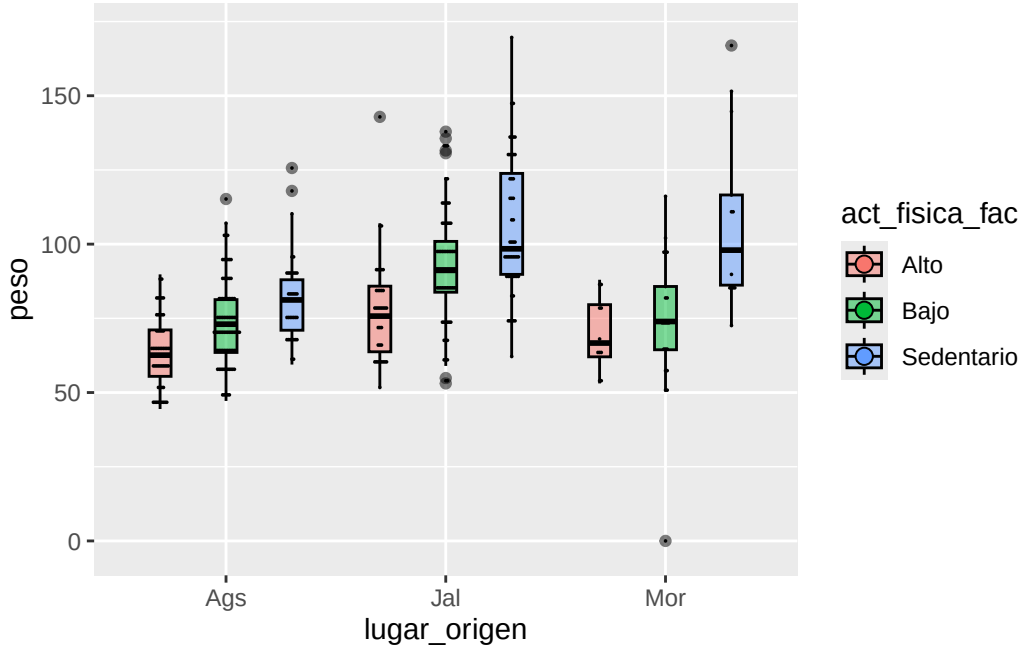
De esta forma, generamos nuestro boxplot. Recordemos que primero hay que crear nuestro objeto, luego llamos nuestra base de datos, creamos el lienzo base con los estéticos, poniendo en el eje de las "x" el "tipo_cancer" y en el eje "y" el peso, y en la siguiente línea de código seleccionamos el tipo de gráfico, que es boxplot. No olvidemos que para agregar nuevas capas utilizamos el signo "+". Ahora, si bien recordamos, el gráfico seleccionado es exactamente igual que el de violín que generamos en dicha sección, la gran ventaja de usar ggplot2 es que podemos ir probando diversos gráficos a ver si nos agrada más boxplot, violín u otro.

Recordemos que podemos agregar exactamente las misma estética que usamos previamente, solo hay que considerar que algunos gráficos pueden requerir algunos ajustes diferentes, pero vamos a probar las mismas estéticas que en el último gráfico de violín que hicimos. Para no hacer los códigos tan extensos, por motivos de tiempo, dejaremos de lado la "tuneada" de las etiquetas y leyendas.

```
plot <- BD2 %>%
  drop_na(
  ) %>%
  ggplot(
    aes(
      x = lugar_origen,
      y = peso,
      fill = act_fisica_fac
      )
    ) +
```

```
  geom_boxplot(
    width = 0.3,
    color = "black",
    alpha = 0.5,
    position = position_dodge(0.9)
    ) +
  geom_dotplot(
    binaxis = 'y',
    stackdir = 'centerwhole',
    position = position_dodge(0.9),
    dotsize = 0.1
    )
plot
```

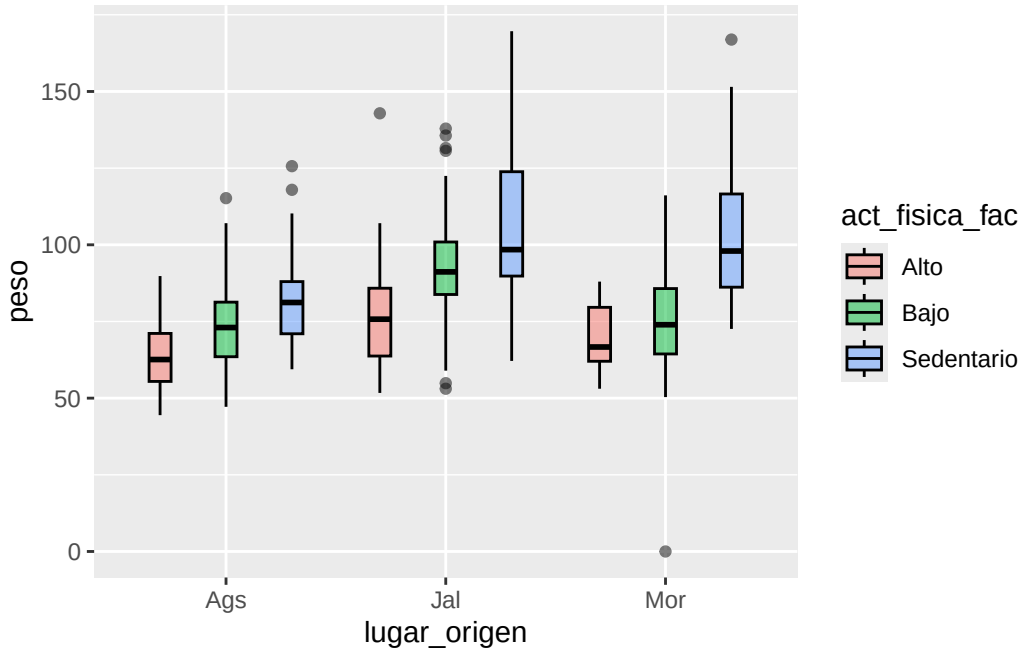
```
Bin width defaults to 1/30 of the range of the data. Pick better value with
`binwidth`.
```



Perfecto, solamente eliminamos la capa de violín y corremos el resto del código, y solamente cambiamos el tamaño del boxplot de 0.1 a 0.3.

Regresemos al formato habitual, dejando los colores y el resto de los estéticos:

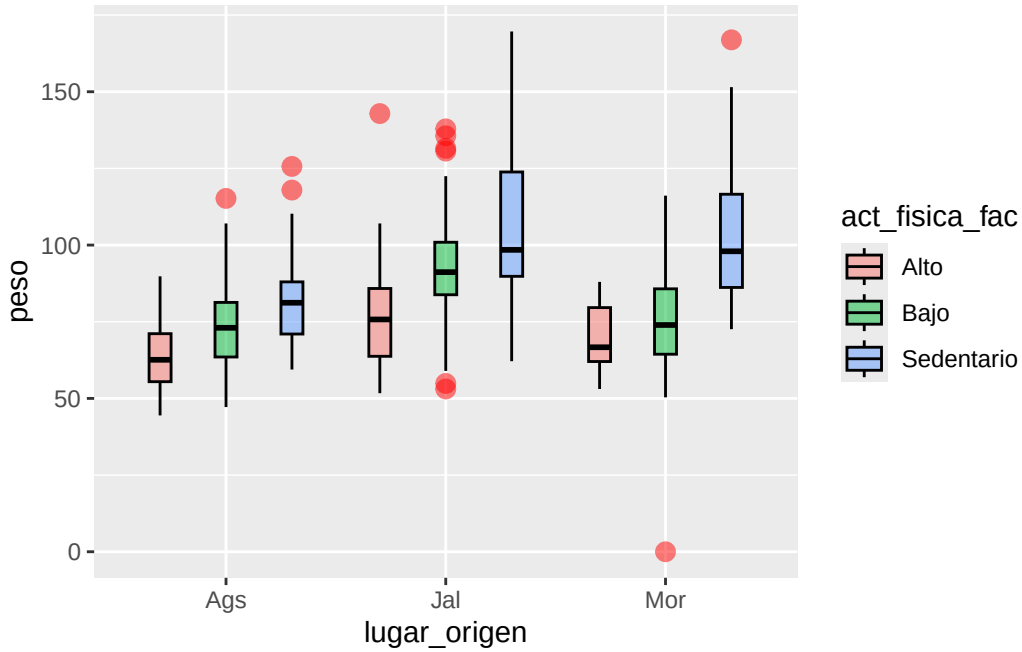
```r
plot <- BD2 %>%
  drop_na(
  ) %>%
  ggplot(
    aes(
      x = lugar_origen,
      y = peso,
      fill = act_fisica_fac
      )
    ) +
  geom_boxplot(
    width = 0.3,
    color = "black",
    alpha = 0.5,
    position = position_dodge(0.9)
    )
plot
```



Listo, con este gráfico base, vamos a anexar diversas características que pueden ser de utilidad.

Vamos a ajustar los puntos atípicos, veamos un ejemplo:

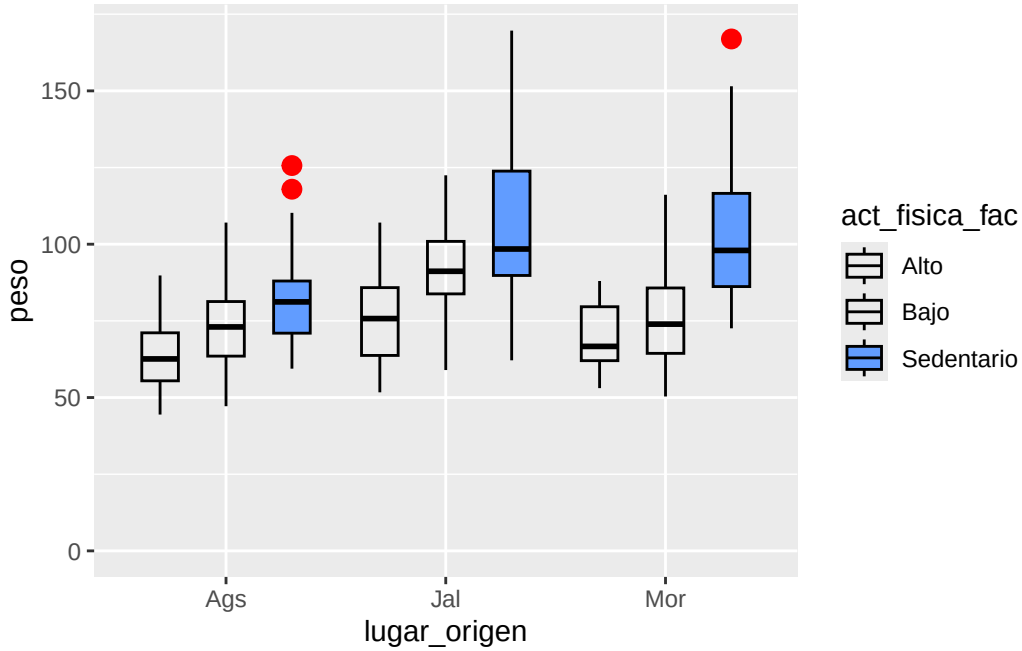
```r
plot <- BD2 %>%
  drop_na(
  ) %>%
  ggplot(
    aes(
      x = lugar_origen,
      y = peso,
      fill = act_fisica_fac
      )
    ) +
  geom_boxplot(
    width = 0.3,
    color = "black",
    alpha = 0.5,
    position = position_dodge(0.9),
    outlier.colour = "red",
    outlier.size = 3
    )
plot
```



Con lo anterior, podemos resaltar los puntos atípicos con los argumentos de "outlier." usando "colour" y "size", con lo que podemos poner diferentes colores y tamaños.

También es posible resaltar un grupo específico, esto se logra mediante el siguiente código:

```
plot <- BD2 %>%
  drop_na(
  ) %>%
  ggplot(
    aes(
      x = lugar_origen,
      y = peso,
      fill = act_fisica_fac,
      alpha = act_fisica_fac
      )
    ) +
  geom_boxplot(
    width = 0.5,
    color = "black",
    position = position_dodge(0.9),
    outlier.color = "red",
    outlier.size = 3) +
  scale_alpha_manual(
    values = c(0,0,1
               )
    )
plot
```



En el código anterior lo que hacemos es agregar un nuevo "aesthetic" mediante "alpha", al que le decimos que también sea la variable "act_fisica_fac", esto nos permite aumentar o disminuir la transparencia de ese grupo, Luego, con la función "scale_alpha_manual" pedimos que los grupos tengan una transparencia máxima (0) y el otro que no tenga transparencia (1), lo que hace que resalte que el grupo sedentario tenga color.

> **Tip**
>
> Aquí también podemos ajustar los niveles de transparencia, poniendo un 0.6, por ejemplo.

Hay que notar que los puntos atípicos desaparecen, para que se sigan mostrando habría que poner transparencias de 0.1 al menos.

Con lo anterior generamos muchos gráficos interesantes y altamente personalizados acorde a nuestra necesidades.

2.2 Correlación

Un gráfico de correlación, también conocido como diagrama de dispersión o "scatter plot", es una herramienta visual que nos ayuda a entender cómo se relacionan dos variables entre sí. Este tipo de gráficos requiere de dos variables (una en el eje de las "x" y la otra en las "y"), sean de tipo numérico. Son dos los principales gráficos de correlación, que son el "scatter plot" o el diagrama de dispersión, que es el clásico y ampliamente utilizado, pero también tenemos el

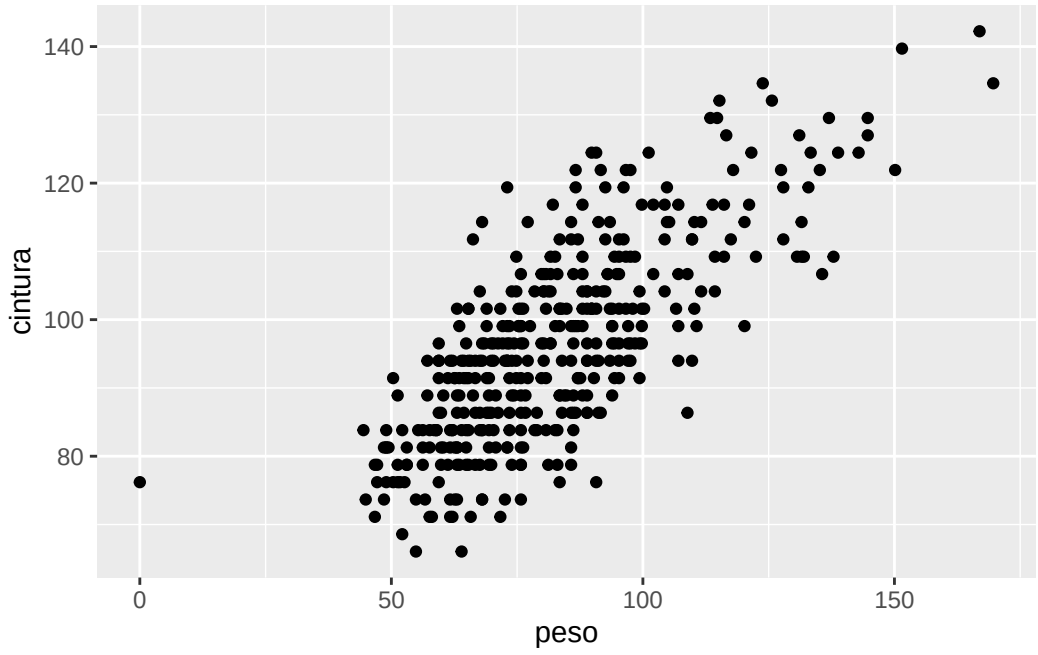
mapa de calor o “heatmap”, que nos indica mediante una escala de colores de menor a mayor correlación.

2.2.1 Scatterplot

El scatterplot básico, se realiza mediante el siguiente código:

```
plot <- BD2 %>%
  ggplot(
    aes(
      x = peso,
      y = cintura
      )
    ) +
  geom_point(
  )
plot
```

```
Warning: Removed 2 rows containing missing values or values outside the scale range
(`geom_point()`).
```

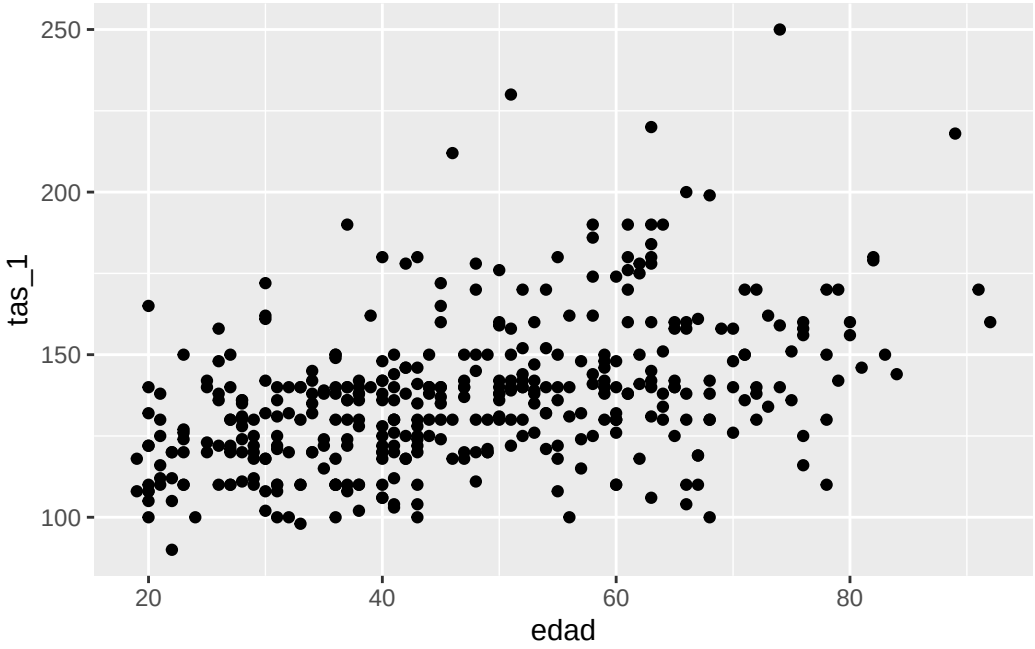


Y listo, tan sencillo como tres líneas de código y ya generamos un "scatter plot" que nos muestra la relación entre peso y cintura.

Hagamos otro ejemplo:

```
plot <- BD2 %>%
  ggplot(
    aes(
      x = edad,
      y = tas_1
      )
    ) +
  geom_point(
  )
plot
```

```
Warning: Removed 5 rows containing missing values or values outside the scale range
(`geom_point()`).
```



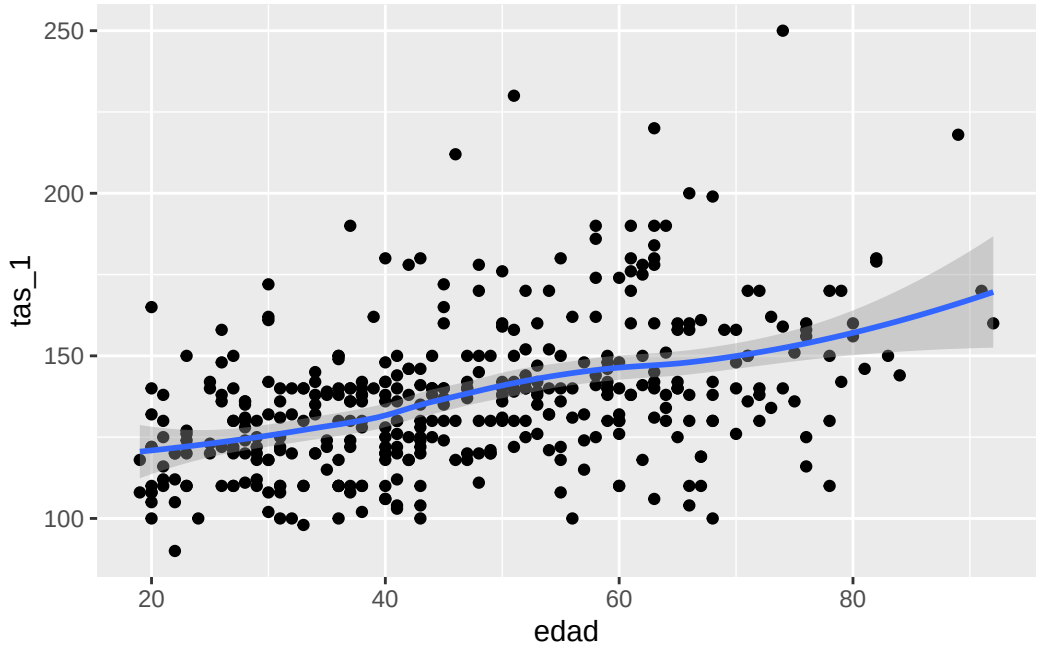
Ahora, este es un caso interesante, ya que tengo mucha imaginación (y conocimiento previo de la base de datos…), me parece ver una tendencia de que a mayor edad, mayor tas_1, lo cuál es un tanto esperado, para ello, podemos agregar una línea regresora con el siguiente código.

```r
plot <- BD2 %>%
  ggplot(
    aes(
      x = edad,
      y = tas_1
      )
    ) +
  geom_point(
  ) +
  geom_smooth(
  )
plot
```

```
`geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```

```
Warning: Removed 5 rows containing non-finite outside the scale range
(`stat_smooth()`).
```

```
Warning: Removed 5 rows containing missing values or values outside the scale range
(`geom_point()`).
```



Naturalmente, podemos ajustar muchas características a nuestro scatterplot, veamos algunas de ellas:

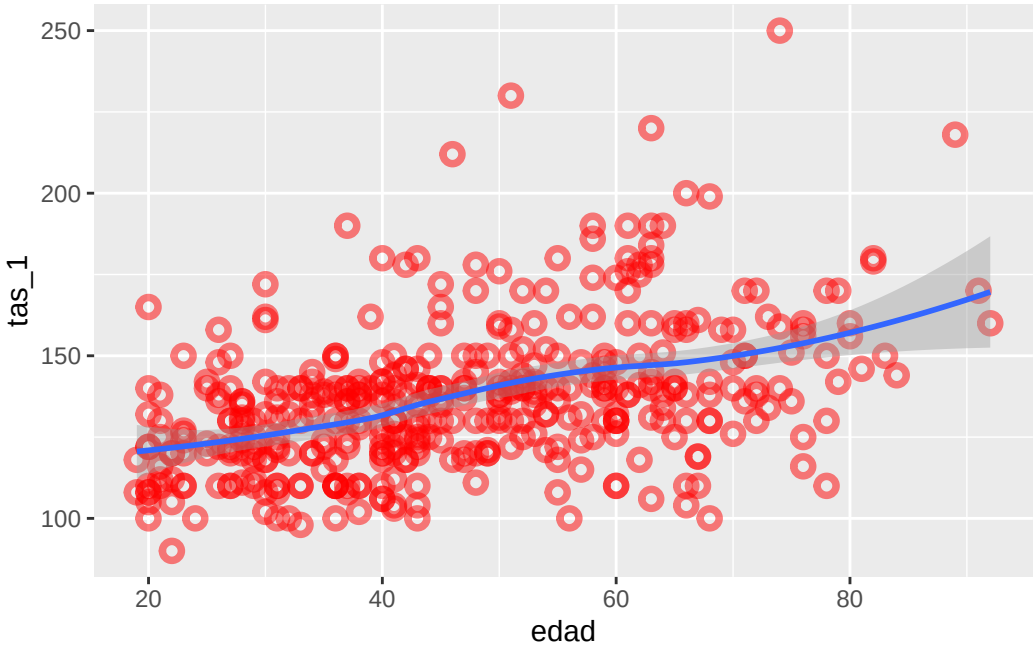
Hagamos el siguiente códidgo:

```
plot <- BD2 %>%
  ggplot(
    aes(
      x = edad,
      y = tas_1
      )
    ) +
  geom_point(
    color = "red",
              shape = 1,
              alpha = 0.5,
              size = 2,
              stroke = 2
    ) +
  geom_smooth(
  )
plot
```

```
`geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```

```
Warning: Removed 5 rows containing non-finite outside the scale range
(`stat_smooth()`).
```

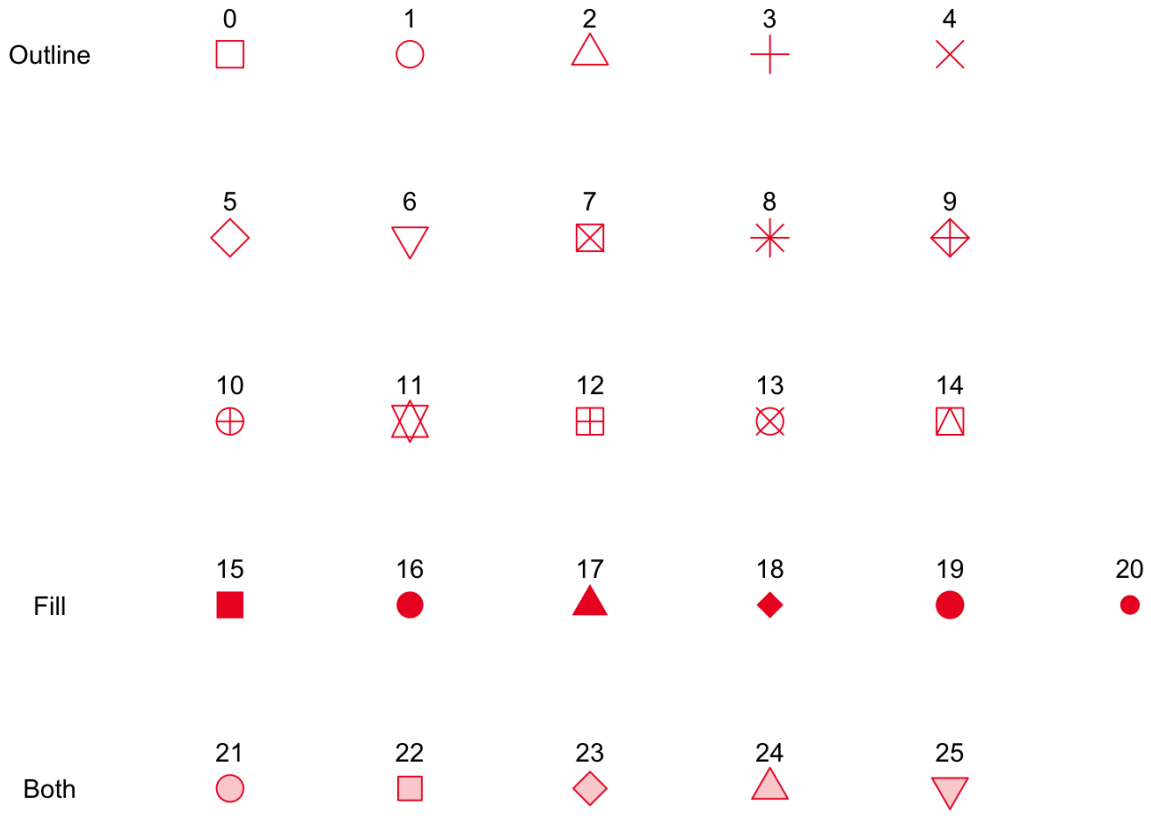
```
Warning: Removed 5 rows containing missing values or values outside the scale range
(`geom_point()`).
```



Ahora, recordemos que podemos jugar con colores, formas, transparencia, tamaños e incluso el grosor de líneas de las formas.

Considero que las funciones son bastantes autoexplicativas.

Ahora, ya que podemos elegir diferentes formas con el argumento “shape”, tenemos las siguientes:



La figuras marcadas con "Outline", que van del 1 al 14, solo pueden cambiarse el color del trazo o la línea de la figura; las figuras marcadas con "Fill" que son del 15 al 20 solo pueden cambiarse el color del relleno; mientras que las marcadas con "Both", que son del 21 al 25 se puede cambiar tanto el color del contorno, como del relleno.

> **Tip**
>
> Por cuestiones de personalización, sugiero utilizar siempre del 21 al 25, ya que se puede personalizar más fácil, y si llegaran a faltar, utilizar las formas que no se han usado, como son 3, 4, y del 7 al 14.

Vamos a hacer otro ejemplo:

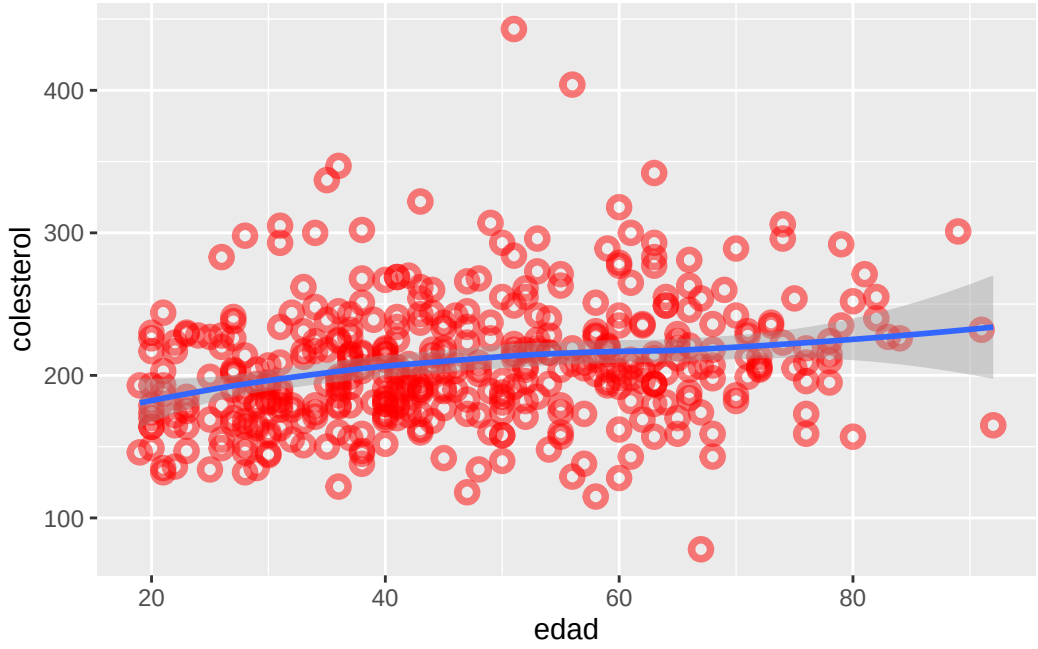
```
plot <- BD2 %>%
  ggplot(
    aes(
      x = edad,
      y = colesterol,
      )
```

```
    ) +
  geom_point(
    color = "red",
              shape = 1,
              alpha = 0.5,
              size = 2,
              stroke = 2
    ) +
  geom_smooth(
  )
plot
```

```
`geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```

```
Warning: Removed 1 row containing non-finite outside the scale range
(`stat_smooth()`).
```

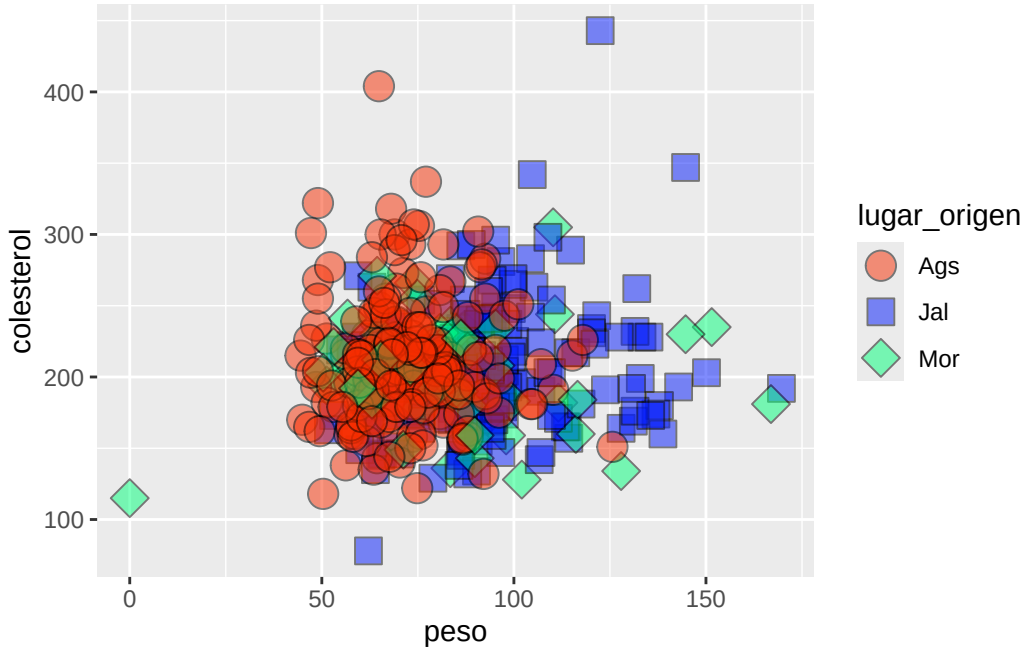
```
Warning: Removed 1 row containing missing values or values outside the scale range
(`geom_point()`).
```



Naturalmente, los niveles de personalización con las características “alpha”, “shape”, “fill”, “size”, son bastante, vamos a usarlas en el siguiente gráfico:

```r
plot <- BD2 %>%
  ggplot(
    aes(
      x = peso,
      y = colesterol,
      shape = lugar_origen,
      fill = lugar_origen
      )
    ) +
  geom_point(
    color = "black",
    alpha = 0.5,
    stroke = 0.5,
    size = 5
    ) +
  scale_fill_manual(
    values = c(
      "#f82d00ff",
      "#0019fcff",
      "#00ff7bff"
      )
    ) +
  scale_shape_manual(
    values = c(21:23)
    )
plot
```

```
Warning: Removed 1 row containing missing values or values outside the scale range
(`geom_point()`).
```



Ya hemos visto y descrito todas las funciones y argumentos, solo cabe mencionar que "scale_" ya sea con "fill" o "shape", nos permite seleccionar manualmente ya sea los colores o formas que deseemos, respectivamente.

Vamos a ver la función de separar los datos por variable:

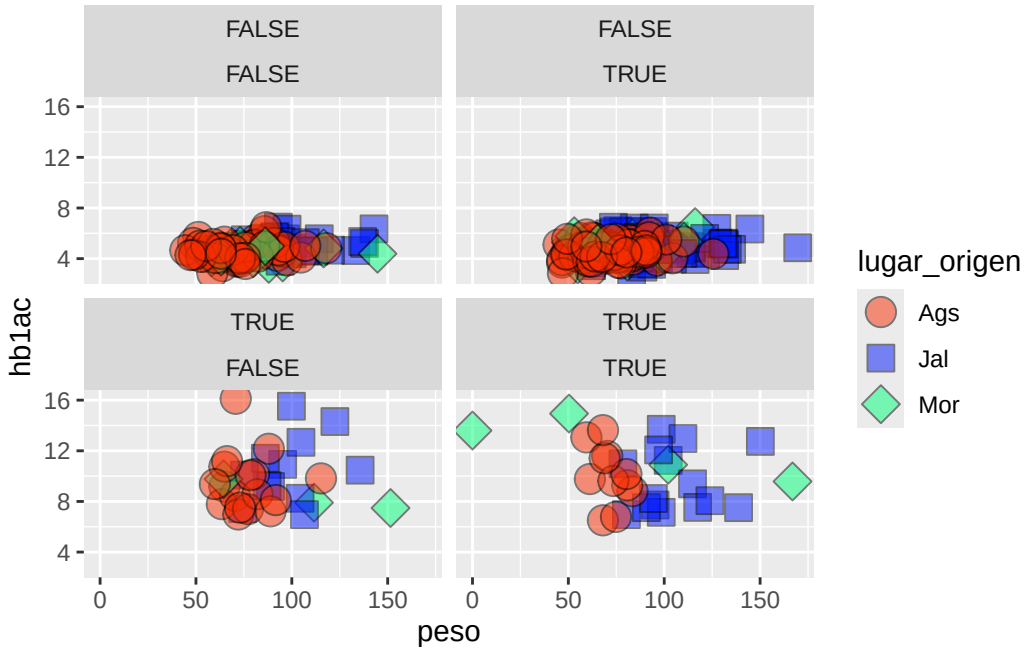
```
plot <- BD2 %>%
  drop_na() %>%
  ggplot(
    aes(
      x = peso,
      y = hb1ac,
      shape = lugar_origen,
      fill = lugar_origen
      )
    ) +
  geom_point(
    color = "black",
    alpha = 0.5,
    stroke = 0.5,
    size = 5
    ) +
  scale_fill_manual(
    values = c(
```

```
      "#f82d00ff",
      "#0019fcff",
      "#00ff7bff"
      )
    ) +
  scale_shape_manual(
    values = c(21:23)
    ) +
  facet_wrap(~diabetes_log)
plot
```

En lo anterior, simplemente agregamos la función "facet_wrap" para diabetes_log, y con eso tenemos dos gráficos, uno para cada observación que son sin y con diabetes (en la parte de arriba se ve la descripción). Ahora, agreguemos más variables:

```
plot <- BD2 %>%
  drop_na() %>%
  ggplot(
    aes(
      x = peso,
      y = hb1ac,
      shape = lugar_origen,
      fill = lugar_origen
```

```
      )
    ) +
  geom_point(
    color = "black",
    alpha = 0.5,
    stroke = 0.5,
    size = 5
    ) +
  scale_fill_manual(
    values = c(
      "#f82d00ff",
      "#0019fcff",
      "#00ff7bff"
      )
    ) +
  scale_shape_manual(
    values = c(21:23)
    ) +
  facet_wrap(~diabetes_log + tx_ta)
plot
```



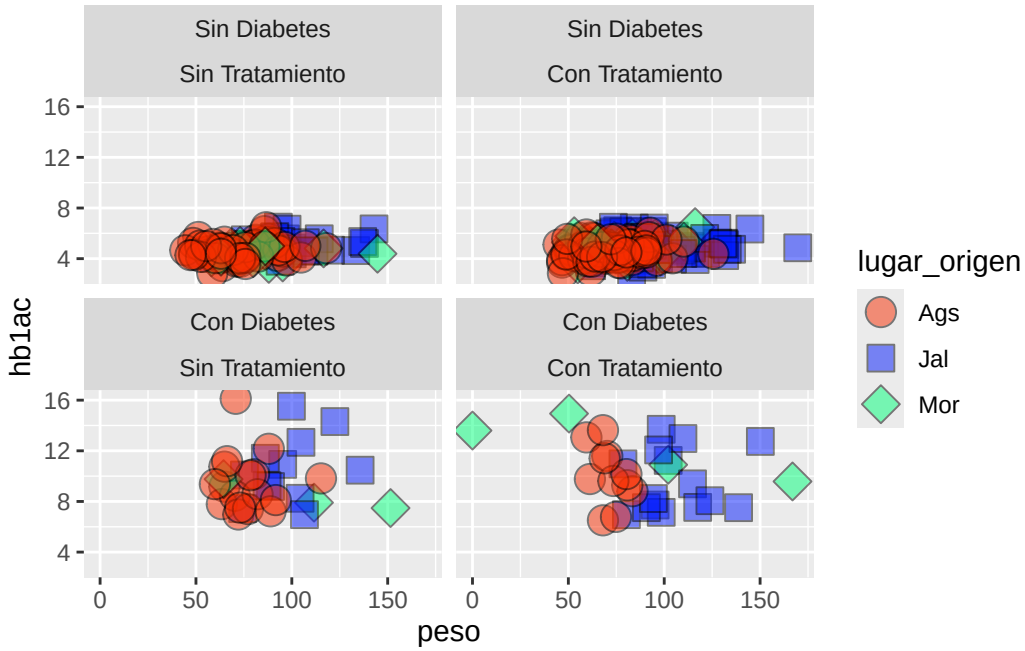
Y ahora tenemos cuatro gráficos, dividiendo en la primera línea para diabetes y la segunda para el tratamiento de la TA.

> **Tip**
>
> Se pueden agregar tantos grupos como se desee con “facet_wrap”, solamente se deben ir añadiendo con el símbolo de “+”

De entrada, puede ser algo difícil de entender, por lo que vamos a crear sus etiquetas con el siguiente código:

```
diabetes_labels <- c(
  `FALSE` = "Sin Diabetes",
  `TRUE`  = "Con Diabetes"
)
tx_ta_labels <- c(
  `FALSE` = "Sin Tratamiento",
  `TRUE`  = "Con Tratamiento"
)
facet_labels <- labeller(
  diabetes_log = as_labeller(diabetes_labels),
  tx_ta        = as_labeller(tx_ta_labels)
)
plot <- BD2 %>%
  drop_na() %>%
  ggplot(
    aes(
      x = peso,
      y = hb1ac,
      shape = lugar_origen,
      fill = lugar_origen
      )
    ) +
  geom_point(
    color = "black",
    alpha = 0.5,
    stroke = 0.5,
    size = 5
    ) +
  scale_fill_manual(
    values = c(
      "#f82d00ff",
      "#0019fcff",
      "#00ff7bff"
      )
    ) +
```

```
  scale_shape_manual(
    values = c(21:23)
    ) +
  facet_wrap(~diabetes_log
             + tx_ta,
               labeller = facet_labels
               )
plot
```



Ahora es mucho más entendible, si se tienen más grupos, se puede ir agregando las etiquetas en orden.

2.2.2 Mapa de calor

El otro gráfico de correlación es el mapa de calor o "heatmap", el cual es una representación gráfica de una matriz de correlación en la que un gradiente de color predeterminado nos indica si existe o no una correlación entre dos variables numéricas.

Se deben de llevar a cabo algunas transformaciones de la base de datos, ya que debe de contener solamente datos numéricos, vamos a quitar todo lo que no sea numérico:

```r
BD_clases <- sapply(BD2, class)
info_cols <- data.frame(
  Columna = names(BD_clases),
  Clase = unname(BD_clases)
  )
print(info_cols)
```

```
          Columna     Clase
1              id character
2      colesterol   numeric
3         glucosa   numeric
4             hdl   numeric
5   col_hdl_ratio   numeric
6           hb1ac   numeric
7    diabetes_log   logical
8    lugar_origen character
9            edad   numeric
10           sexo character
11         altura   numeric
12           peso   numeric
13            imc   numeric
14 act_fisica_fac character
15          tx_ta   logical
16          tas_1   numeric
17          tad_1   numeric
18          tas_2   numeric
19          tad_2   numeric
20        cintura   numeric
21         cadera   numeric
```

Primero hay que ver de nuevo las clases para ver qué vamos a quitar, te dejo un pequeño código para que la visualización sea mucho más amigable.

Ya vemos cuáles son numéricas, vamos a seleccionarlas, tranformarlas en una matriz y quitar lo valores NA.

```r
BD3 <- BD2 %>%
  select(colesterol,
         glucosa,
         hdl,
         col_hdl_ratio,
         hb1ac,
```

```
         edad,
         altura,
         peso,
         imc,
         tx_ta,
         tas_1,
         tad_1,
         tas_2,
         tad_2,
         cintura,
         cadera)
BD3 <- as.matrix(BD3)
BD4 <- na.omit(BD3)
colnames(BD4)
```

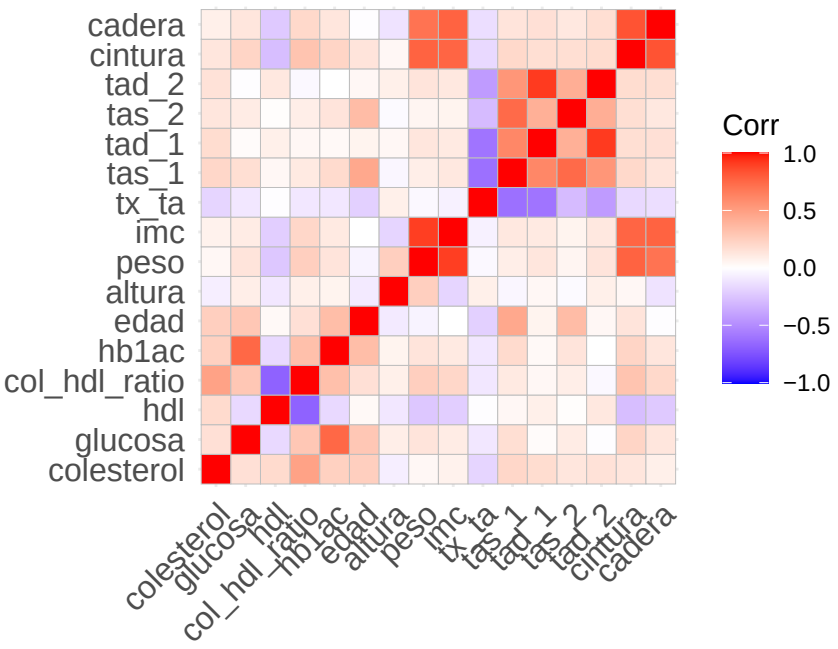
```
 [1] "colesterol"    "glucosa"       "hdl"           "col_hdl_ratio"
 [5] "hb1ac"         "edad"          "altura"        "peso"
 [9] "imc"           "tx_ta"         "tas_1"         "tad_1"
[13] "tas_2"         "tad_2"         "cintura"       "cadera"
```

Con "select" seleccionamos las variables de interés. Ya que solo tenemos variables de interés numéricas, procedemos a hacer la transformación de la estructura de datos a matriz con la función "as.matrix". Finalmente, los datos nulos dificultan la correlación, por lo que podríamos tener datos vacíos, así que haremos una segunda base eliminando dichos datos vacíos.

> **Tip**
>
> Una alternativa, si es que es menos lo que hay que quitar (seis variables) puedes hacerlo con poniendo un signo de menos antes de cada nombre de variable.

Vamos a generar ahora nuestra matriz de correlación con el siguiente código:

```
pacman::p_load(ggcorrplot)
corr_plot <- round(cor(BD4), 3)
corr_p_val <- cor_pmat(BD4)
ggcorrplot(corr_plot)
```



El código es bastante simple, generamos un objeto que es la matriz de datos usando la BD4.

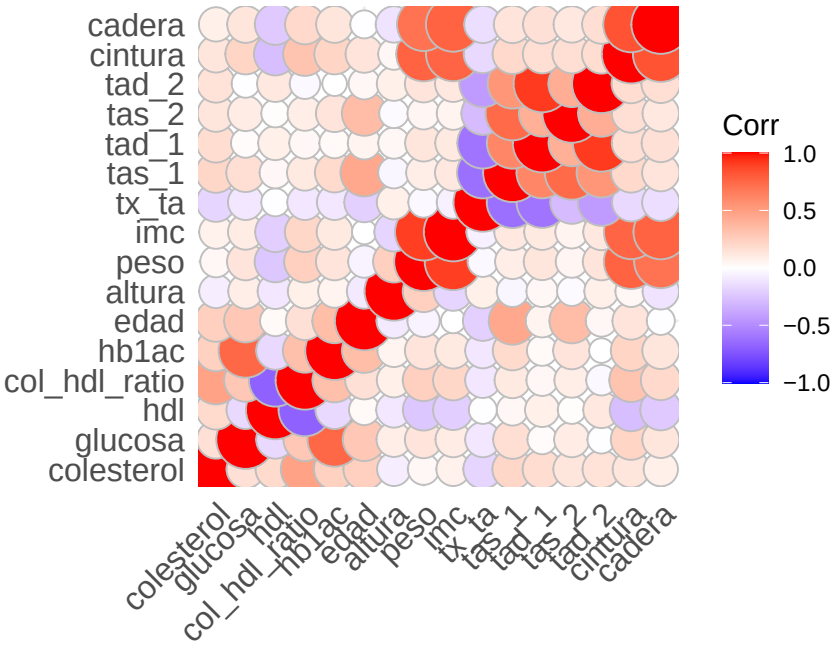
El argumento "round" redondea los resultados a un tres decimales, en este caso, pero se puede ajustar a lo que ustedes deseen. Luego, se agrega la función "cor" para generar la matriz de correlación.

En el segundo código generamos una matriz de valores de p para generar una segundo gráfico con la función "cor_pmat", que obtiene los valores de significancia mediante la prueba de t de Student. Se puede ajustar el tipo de prueba y el tipo de correlación, así como el valor de significancia estadística.

Y listo, con pocas líneas de código tenemos nuestro mapa de calor.

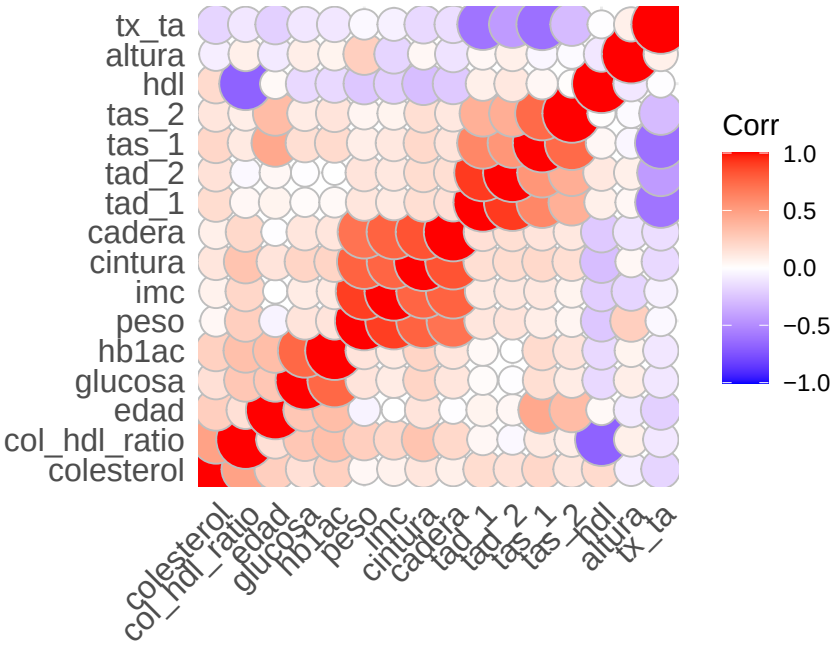
Vamos a hacer unos pequeños cambios:

```
ggcorrplot(corr_plot,
           method = "circle")
```



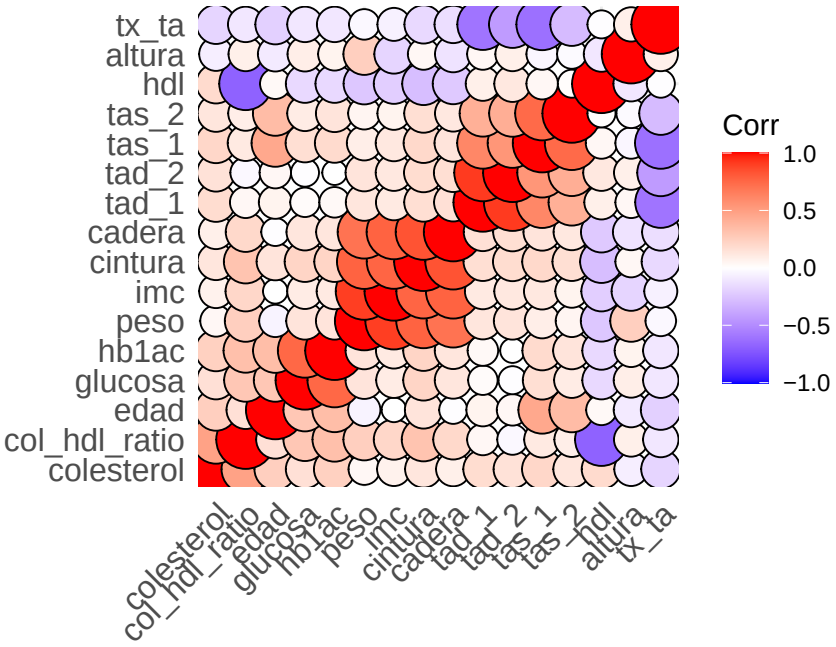
Con el código anterior, elegimos el método y cambiamos a círculos. Además de ver el color de la correlación con la leyenda de la derecha, vemos que el tamaño del círculo también indica el nivel de correlación.

```
ggcorrplot(corr_plot,
           method = "circle",
           hc.order = T)
```



Ahora, cabe la posibilidad de que deseemos agrupar por niveles de correlación, en vez de orden como vienen las variables, esto se logra pidiendo la función "hc.order" a verdadero. Lo que hace más fácil distinguir las variables agrupadas por correlaciones positivas y negativas.

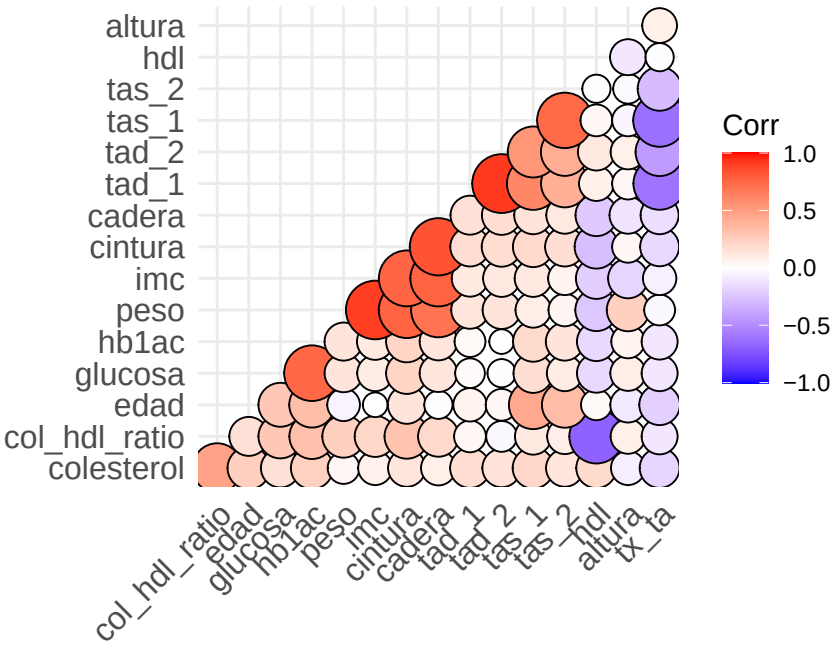
```
ggcorrplot(corr_plot,
           method = "circle",
           hc.order = T,
           outline.color = "black")
```



Con el argumento "outline.color" podemos cambiar el color que rodea cada círculo de correlación, lo que da más contraste.

Naturalmente, vemos la matriz de correlación en espejo, lo que hace una "duplicación de datos" a través de la diagonal, es decir, lo de arriba, es lo mismo de abajo, por lo que podemos mostras solamente una parte y seguimos representado la misma cantidad de información:

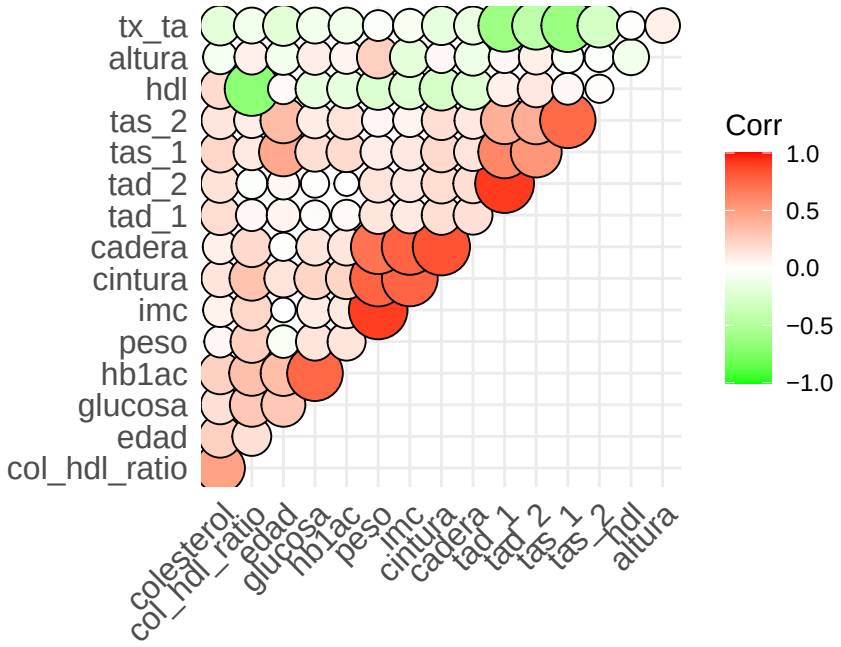
```
ggcorrplot(corr_plot,
           method = "circle",
           hc.order = T,
           outline.color = "black",
           type = "lower")
```



Con lo anterior, mostramos solo la mitad inferior, usando el argumento "type" y la opción "lower", por lo tanto, la parte de arriba se mostraría con "upper".

Si deseamos cambiar los colores, podemos hacer lo siguiente:

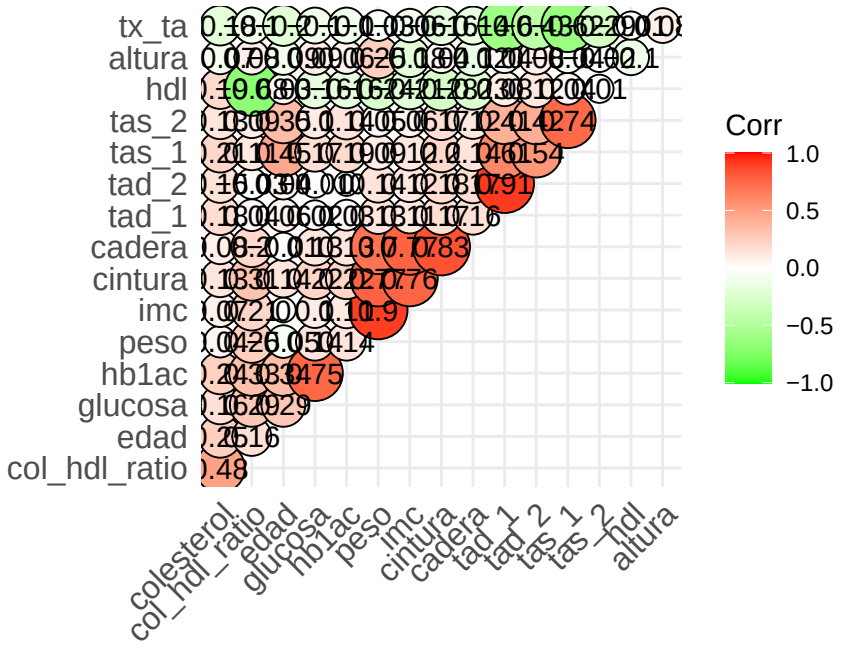
```
ggcorrplot(corr_plot,
           method = "circle",
           hc.order = T,
           outline.color = "black",
           type = "upper",
           colors = c("green","white","red"))
```



Y usando otros colores, una correlación negativa se indica con verde, ausencia de correlación con blanco, y una alta correlación con rojo. Esto se logra con el argumento "colors", recordemos que debemos concatenar o combinar los valores, y debemos de utilizar tres, y se sugiere usar el color blanco para indicar una no correlación.

Ahora, podemos utilizar los valores de nuestra matriz de correlación normalizada, lo que nos indicaría numéricamente una ausencia de correlación, correlación baja, media o alta, si ponemos puntos de corte en tercios. Se logra de la siguiente manera:

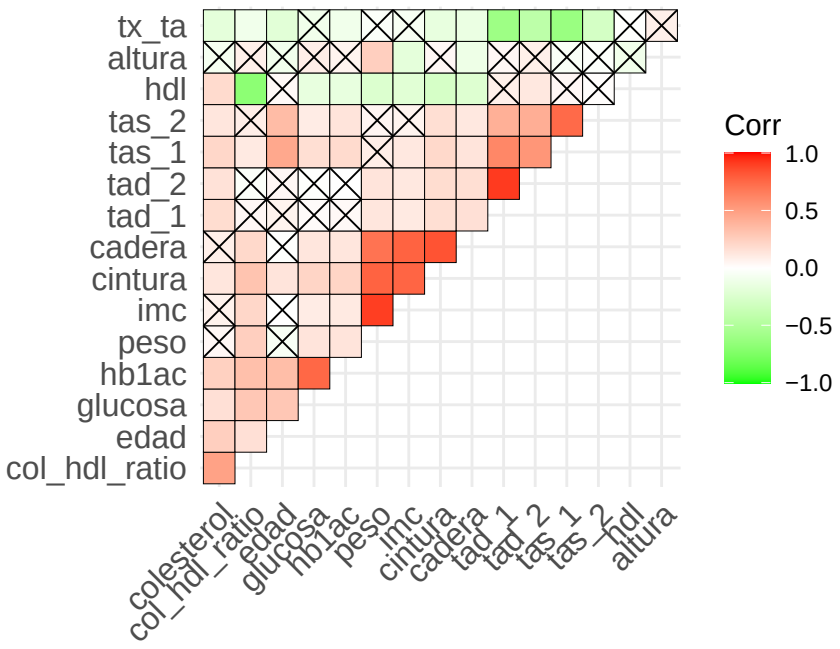
```
ggcorrplot(corr_plot,
           method = "circle",
           hc.order = T,
           outline.color = "black",
           type = "upper",
           colors = c("green","white","red"),
           lab = T
           )
```



Al utilizar el argumento verdadero de “lab”, agregamos los valores que mencionamos, también es de notarse que cambiamos el método por “square”, el cual es el predeterminado, esto se debe a que por el tamaño de los círculos puede generar conflicto en la visualización y no ser adecuado en la estética.

Ahora, en vez de utilizar las etiquetas predeterminadas del valor de la correlación, mejor utilicemos los valores de nuestra matriz de significancia.

```
ggcorrplot(corr_plot,
           hc.order = T,
           outline.color = "black",
           type = "upper",
           colors = c("green","white","red"),
           p.mat = corr_p_val
           )
```

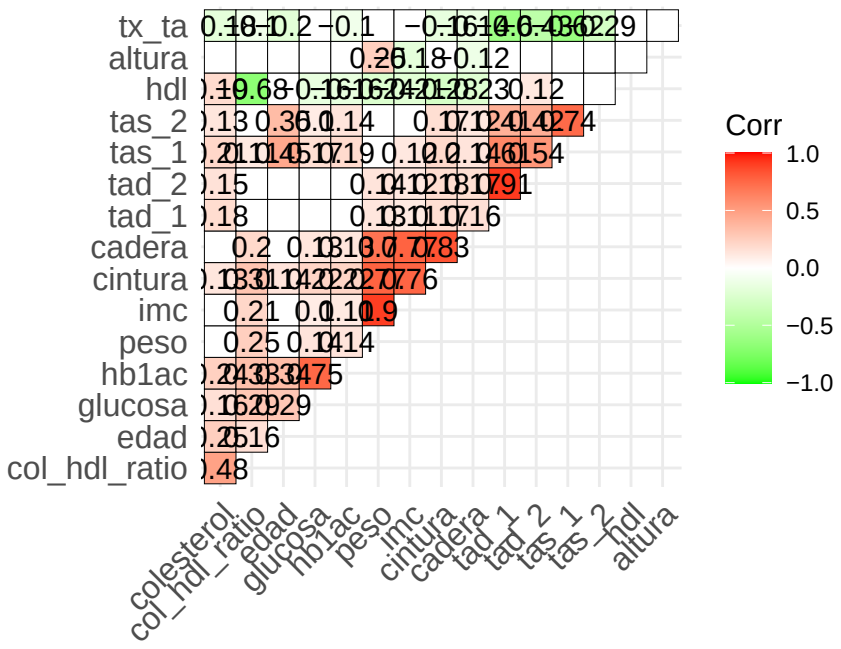


Ahora quitamos el argumento "method", para que se quede en predeterminado o "square", quitamos el argumento "lab", ya que queremos marcar solo las correlaciones significativas, y agregamos el argumento "p.mat" con la matriz de correlaciones de valores de p que habíamos creado.

Lo anterior nos produce que aquellas observaciones o correlaciones que no tengan un valor significativo (ya sea positivo o negativo), se marcarán con una "X".

Si bien, puede darnos la falsa impresión de que aquello marcado es significativo, mejor intentemos dejando en blanco o vacío aquello que no es significativo y pongamos los valores en aquellas que sí.

```
ggcorrplot(corr_plot,
           hc.order = T,
           outline.color = "black",
           type = "upper",
           colors = c("green","white","red"),
           p.mat = corr_p_val,
           insig = "blank",
           lab = T
           )
```



Y listo, con esto se ve mucho más intuitivo y fácil de identificar cuáles correlaciones presentaron una significancia estadística.

2.3 Ranking

El grupo de gráficas de Ranking o rango/categoría nos permite visualizar y comparar las frecuencias o porcentajes de un grupo de elementos por métricas. Nos permiten destacar las posiciones de dichos elementos dentro de un conjunto de datos; además, nos facilita destacar un determinado subgrupo dependiendo de su frecuencia. Se utiliza una variable numérica y una categórica para una comparación, o una sola variable categórica para mostrar su frecuencia de distribución.

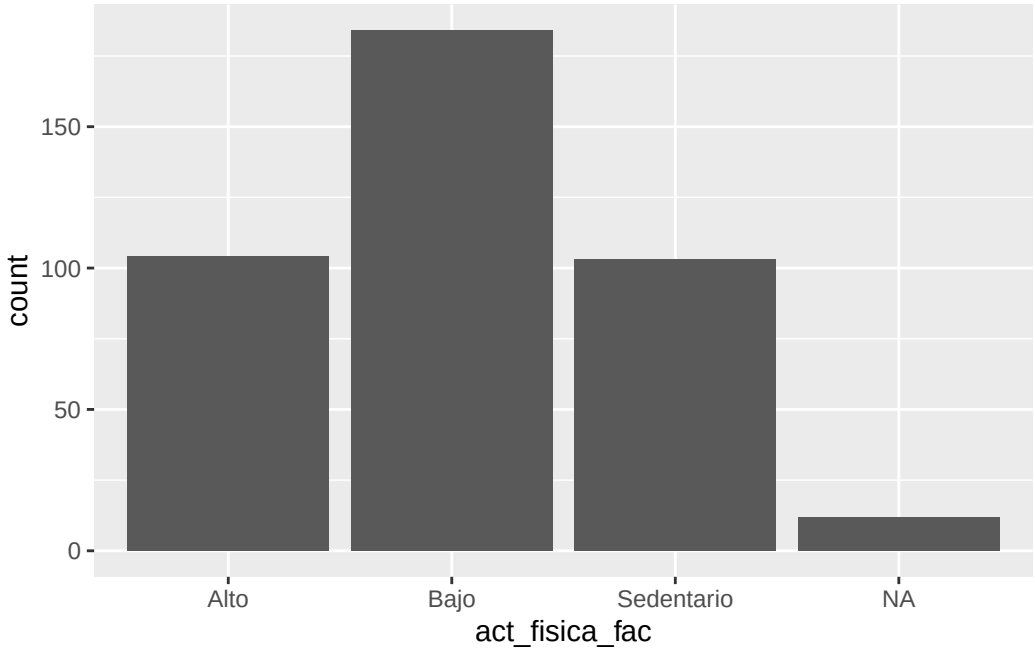
2.3.1 Gráfico de Barras o Barplot

El gráfico más empleado en esta área, es muy similar al histograma, pero aquí lo que se busca es agrupar por factores.

Hagamos un gráfico de barras básico y sencillo:

```
plot <- BD2 %>%
  ggplot(
    aes(
```

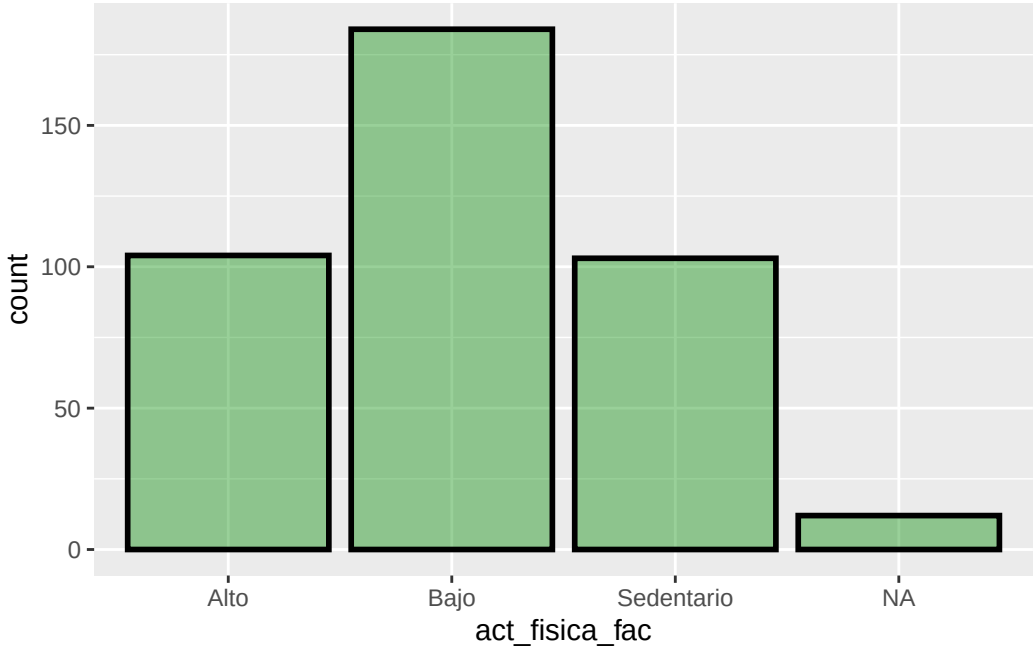
```
      x = act_fisica_fac
    )
    ) +
  geom_bar(
  )
plot
```



Con lo anterior tenemos un gráfico de barras o "barplot" sencillo y básico, podemos elegir transparencia, colores de relleno y de contorno, lo cual puede ayudar para resaltar el contraste:

```
plot <- BD2 %>%
  ggplot(
    aes(
      x = act_fisica_fac
      )
    ) +
  geom_bar(
    color = "black",
    fill = "green4",
    linewidth = 1,
    alpha = 0.4
```

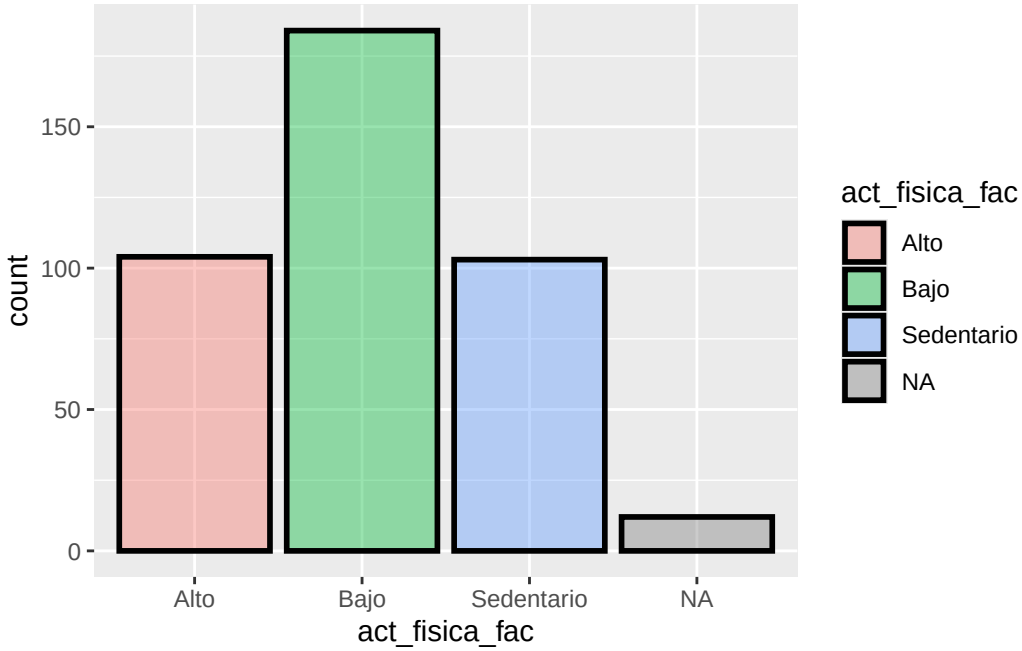
```
    )
plot
```



Con lo anterior, agregamos color negro al contorno de las barras, un relleno verde, una transparencia de 0.4, y un grosor de línea de 1, lo generar un mayor contraste.

Podemos seleccionar la misma variable para colorear los grupos:

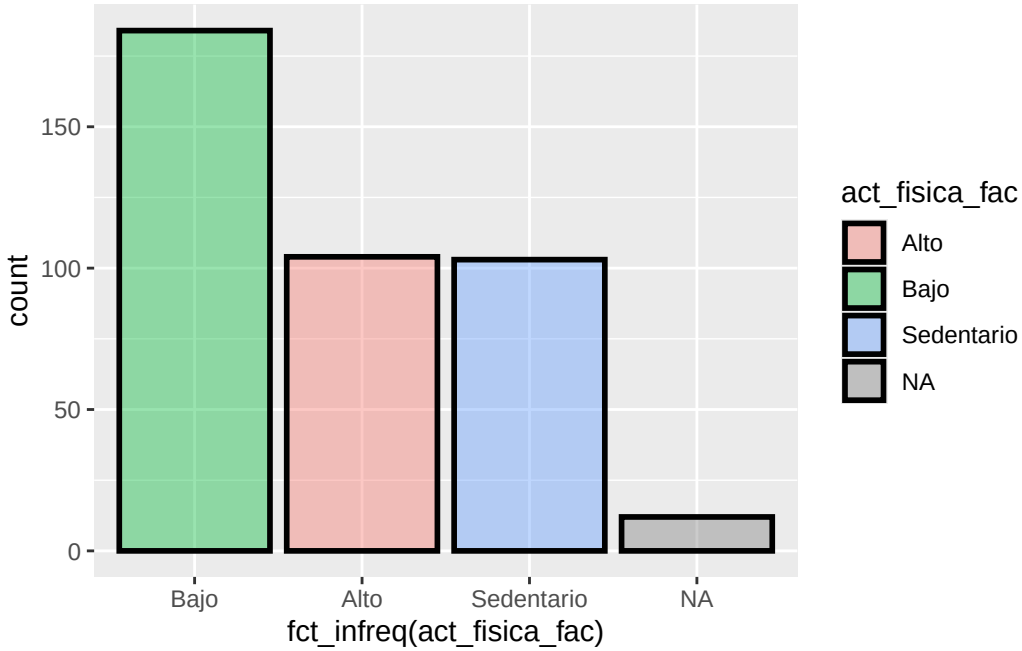
```
plot <- BD2 %>%
  ggplot(
    aes(
      x = act_fisica_fac,
      fill = act_fisica_fac,
      )
    ) +
  geom_bar(
    color = "black",
    linewidth = 1,
    alpha = 0.4
    )
plot
```



> **Tip**
>
> Para que los colores tomen efecto, debemos eliminar el argumento "fill" de "geom_bar".

Podemos reordenar nuestro gráfico de diferentes formas, supongamos que queremos verlo de mayor a menor:

```
plot <- BD2 %>%
  ggplot(
    aes(
      x = fct_infreq(act_fisica_fac),
      fill = act_fisica_fac
      )
    ) +
  geom_bar(
    color="black",
    linewidth = 1,
    alpha = 0.4
    )
plot
```

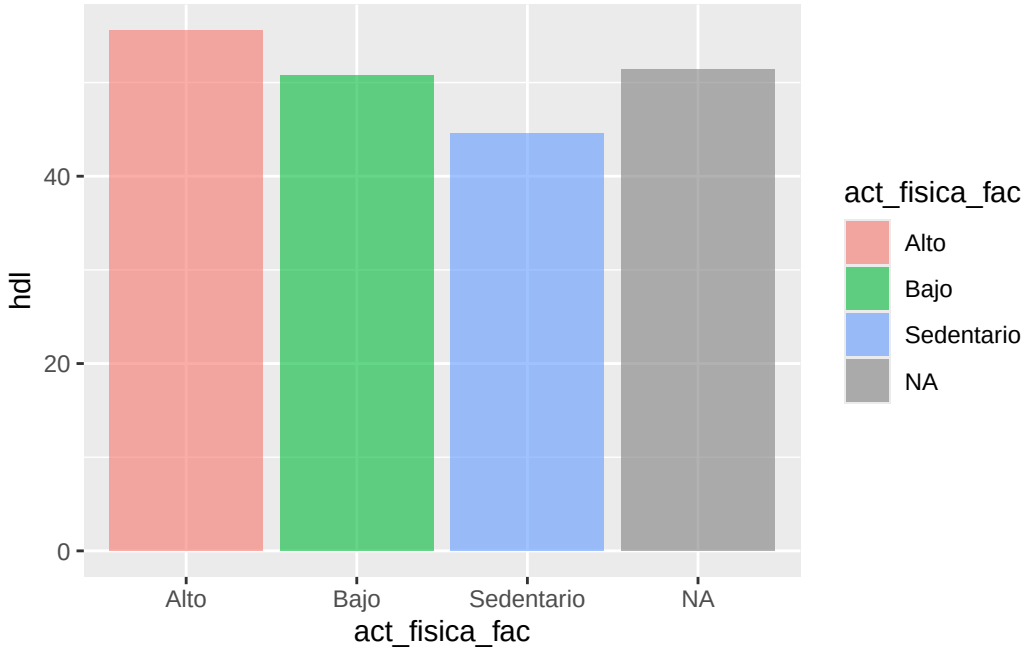


En el código anterior solamente agregamos el argumento "fct_infreq" justo después del estético de "x", y ahí seleccionamos la variable que deseamos que se ordene.

Ahora, vamos a crear gráficos de barras con dos variables para mostrar los niveles de una variable numérica por grupos:

```
plot <- BD2 %>%
  ggplot(
    aes(
      x = act_fisica_fac,
      y = hdl,
      fill = act_fisica_fac
      )
    ) +
  geom_bar(stat = "summary",
           fun = "mean",
           alpha = 0.6
           )
plot
```

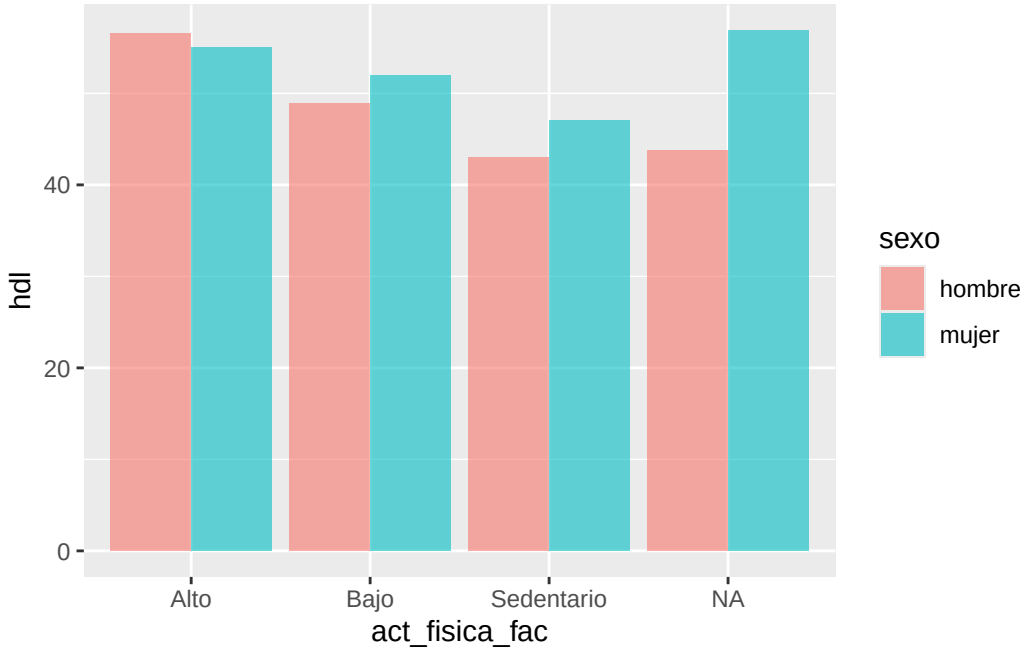
```
Warning: Removed 1 row containing non-finite outside the scale range
(`stat_summary()`).
```



Podemos anexar más variables:

```
plot <- BD2 %>%
  ggplot(
    aes(
      x = act_fisica_fac,
      y = hdl,
      fill = sexo
      )
    ) +
  geom_bar(stat = "summary",
           fun = "mean",
           position = "dodge",
           alpha = 0.6
           )
plot
```

```
Warning: Removed 1 row containing non-finite outside the scale range
(`stat_summary()`).
```

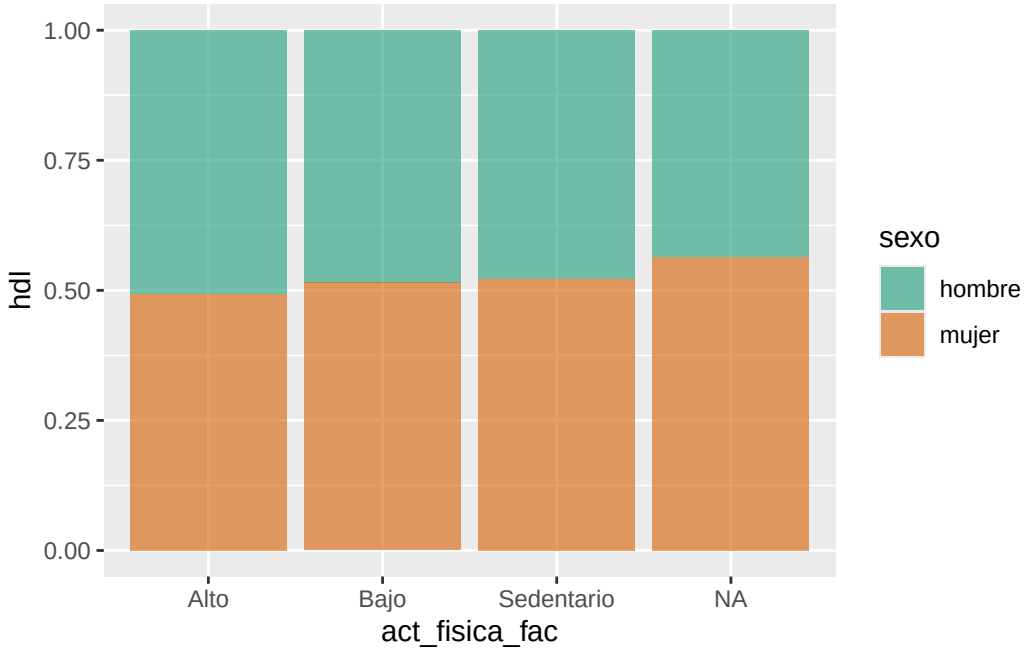


Agregamos un relleno y esquivamos las gráficas para que sean grupos paralelos, podemos anidar, poner uno sobre otro, etc.

Una cualidad bastante útil en los gráficos de barras, es mostrarlo en formato de porcentaje, lo que nos da una mayor claridad de la distribución por grupo:

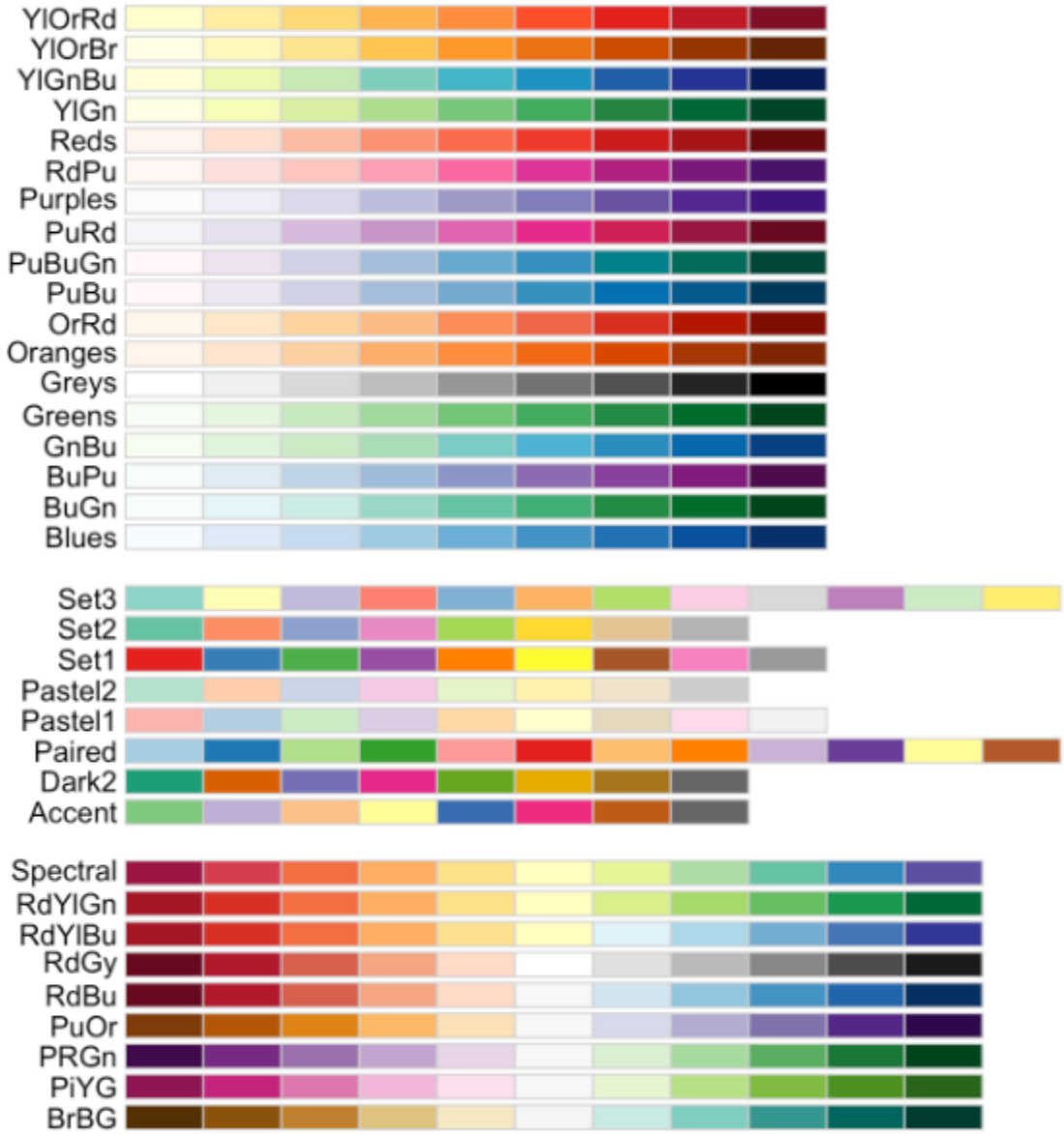
```
plot <- BD2 %>%
  ggplot(
    aes(
      x = act_fisica_fac,
      y = hdl,
      fill = sexo
      )
    ) +
  geom_bar(
    stat = "summary",
    fun = "mean",
    position = "fill",
    alpha = 0.6
    ) +
  scale_fill_brewer(
    palette = "Dark2"
    )
plot
```

```
Warning: Removed 1 row containing non-finite outside the scale range
(`stat_summary()`).
```



Lo anterior nos una vistazo inicial de si pudiera haber una diferencia entre grupos. Además, utilizamos una paleta de colores diferente con la función "scale_fill_brewer".

Para más paletas de colores, aquí hay algunos ejemplos:

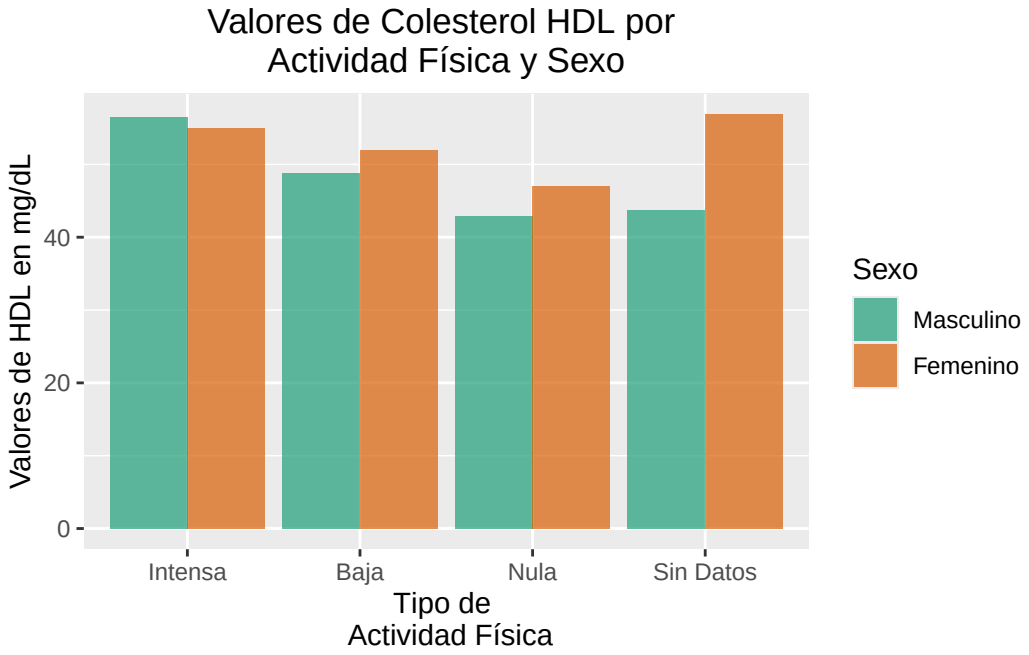


Ahora, es tiempo de ajustar los nombres de leyendas y títulos, esto se logra de la siguiente manera:

```
plot <- BD2 %>%
  ggplot(
    aes(
      x = act_fisica_fac,
      y = hdl,
      fill = sexo
```

```
      )
    ) +
  geom_bar(
    stat = "summary",
    fun = "mean",
    position = "dodge",
    alpha = 0.7
    ) +
  labs(
    title = "Valores de Colesterol HDL por \nActividad Física y Sexo",
    y = "Valores de HDL en mg/dL"
    ) +
  scale_fill_brewer(
    palette = "Dark2",
    name = "Sexo",
    labels = c(
      "Masculino",
      "Femenino"
      )
    ) +
  scale_x_discrete(
    name = "Tipo de \n Actividad Física",
    labels = c(
      "Intensa",
      "Baja",
      "Nula",
      "Sin Datos"
      )
    ) +
  theme(
    plot.title = element_text(hjust = 0.5
                              )
    )
plot
```

```
Warning: Removed 1 row containing non-finite outside the scale range
(`stat_summary()`).
```



Muy bien, en esta sección del código anidamos todas los posibles ajustes que podemos necesitar de etiquetas y nombres. Veamos paso a paso:

1. Función "labs": modifica todas las etiquetas necesarias ("title" para título principal", "y" para el nombre de la variable en dicho eje".
2. Función "scale_fill_brewer": modifica la etiqueta correspondiente al "aes" "fill", con "name" cambiamos el nombre de la variable, con "labels" cambiamos los nombres de los factores (recordar que el número debe de ser congruente con la cantidad de factores, en este caso son cinco).
3. Función "scale_x_discrete": modifica la etiqueta correspondiente al "aes" "x", mismos argumentos, aquí son cuatro factores.
4. Función "theme": modifica ajustes del tema, en este caso "plot.title" con "element_text" y "hjust", hace una justificación horizontal para que se centre usando el valor de "0.5".

Y básicamente son los principales ajustes que podríamos necesitar para nuestros gráficos.

No nos olvidemos de guardar nuestro avance:

```
pacman::p_load("writexl")
write_xlsx(BD2, "BD2_R.xlsx")
```