# PEC 4 - Software para el Análisis de Datos

Enrique Gonzalo Martín / Otelo Pons Alonso

2025-05-31

## Contents

# 1 Sección 1. Contexto y objetivo del estudio.

Para realizar esta actividad hemos seleccionado el conjunto de datos "Diabetes data", cedidos por el Dr. Dr John Schorling de la Facultad de Medicina de la Universidad de Virgina y publicados en la página https: //hbiostat.org/data/. Este conjunto de datos consta de 19 variables medidas para 403 sujetos, y su objetivo original era analizar la prevalencia de la obesidad, la diabetes y otros factores de riesgo cardiovascular en individuos afroamericanos de Virginia central.Las variables que contien son: ID del paciente, nivel de colesterol, nivel de glucosa estable en sangre en ayunas, nivel de HDL, relación entre colesterol total y colesterol HDL, el nivel de hemoglobina glicosilada, la ubicación geográfica del paciente, la edd, el sexo, la altura, el peso, la complexión, el diámetro de la cintura, el diámetro de la cadera, la tensión sistólica y diastólica medidas en 2 ocasiones y el tiempo transcurrido entre la última comida y el análisis.

Nuestro propósito de partida era examinar las relaciones estadísticas entre múltiples marcadores o parámetros biológicos, en una N de sujetos lo más grande posible. Tras analizar distintos conjuntos, decidimos emplear el de Diabetes porque contiene múltiples variables que podemos representar y analizar en busca de correlaciones, así como una N de observaciones amplia (403). En definitiva, es un conjunto muy completo y permite hacer un buen estudio estadístico enmarcado en el ámbito biológico.

Tras examinar cuidadosamente el conjunto de datos, nos propusimos usar la información contenida en él para buscar posibles marcadores que permitan realizar un diagnóstico temprano de la diabetes, o predecir qué pacientes son susceptibles de desarrollar diabetes en el futuro. Por ello, intentaremos, mediante estadística inferencial crear un modelo para predecir la diabetes.

## 1.1 Inicialización entorno trabajo

# 2 Sección 2. Prospección y preparación de los datos

Ahora, observaremos el contenido y la estructura del conjunto de datos, las variables presentes y su resumen estadístico, empleando las distintas funciones que hemos ido estudiando. Además, comprobaremos si contiene NAs y, en caso afirmativo, cuántos de ellos hay en cada columna.

## 2.1 Descripción del conjunto de datos

Mostramos una imagen del repositorio web del que obtuvimos los datos, en la cual se puede ver una explicación del dataset con sus observaciones y variables.

# Data frame:diabetes

403 observations and 19 variables, maximum # NAs:262

| Name | Labels | Units | Levels | Storage | NAs |
|---|---|---|---|---|---|
| id | Subject ID | | | double | 0 |
| chol | Total Cholesterol | | | double | 1 |
| stab.glu | Stabilized Glucose | | | double | 0 |
| hdl | High Density Lipoprotein | | | double | 1 |
| ratio | Cholesterol/HDL Ratio | | | double | 1 |
| glyhb | Glycosolated Hemoglobin | | | double | 13 |
| location | | | 2 | integer | 0 |
| age | | years | | double | 0 |
| gender | | | 2 | integer | 0 |
| height | | inches | | double | 5 |
| weight | | pounds | | double | 1 |
| frame | | | 3 | integer | 12 |
| bp.1s | First Systolic Blood Pressure | | | double | 5 |
| bp.1d | First Diastolic Blood Pressure | | | double | 5 |
| bp.2s | Second Systolic Blood Pressure | | | double | 262 |
| bp.2d | Second Diastolic Blood Pressure | | | double | 262 |
| waist | | inches | | double | 2 |
| hip | | inches | | double | 2 |
| time.ppn | Postprandial Time when Labs were Drawn | minutes | | double | 3 |

### 2.2 Carga del dataset y exploración inicial

Cargamos el archivo que vamos a emplear para nuestro análisis y mostramos los tipos de datos que contiene:

```
# cargamos el archivo
diabetes_df <- read.csv("data/diabetes.csv")

# vemos los tipos de datos en él
str(diabetes_df)
```

```
## 'data.frame':    403 obs. of  19 variables:
##  $ id      : int  1000 1001 1002 1003 1005 1008 1011 1015 1016 1022 ...
##  $ chol    : int  203 165 228 78 249 248 195 227 177 263 ...
##  $ stab.glu: int  82 97 92 93 90 94 92 75 87 89 ...
##  $ hdl     : int  56 24 37 12 28 69 41 44 49 40 ...
##  $ ratio   : num  3.6 6.9 6.2 6.5 8.9 ...
##  $ glyhb   : num  4.31 4.44 4.64 4.63 7.72 ...
##  $ location: chr  "Buckingham" "Buckingham" "Buckingham" "Buckingham" ...
##  $ age     : int  46 29 58 67 64 34 30 37 45 55 ...
##  $ gender  : chr  "female" "female" "female" "male" ...
##  $ height  : int  62 64 61 67 68 71 69 59 69 63 ...
##  $ weight  : int  121 218 256 119 183 190 191 170 166 202 ...
##  $ frame   : chr  "medium" "large" "large" "large" ...
##  $ bp.1s   : int  118 112 190 110 138 132 161 NA 160 108 ...
##  $ bp.1d   : int  59 68 92 50 80 86 112 NA 80 72 ...
##  $ bp.2s   : int  NA NA 185 NA NA NA 161 NA 128 NA ...
##  $ bp.2d   : int  NA NA 92 NA NA NA 112 NA 86 NA ...
##  $ waist   : int  29 46 49 33 44 36 46 34 34 45 ...
##  $ hip     : int  38 48 57 38 41 42 49 39 40 50 ...
##  $ time.ppn: int  720 360 180 480 300 195 720 1020 300 240 ...
```

También sus primeras entradas:

```
  # Comprobamos las primeras entradas del dataset
head(diabetes_df)
```

```
## # A tibble: 6 x 19
##      id  chol stab.glu   hdl ratio glyhb location    age gender height
##   <int> <int>    <int> <int> <dbl> <dbl> <chr>     <int> <chr>   <int>
## 1  1000   203       82    56  3.60  4.31 Buckingh~    46 female     62
## 2  1001   165       97    24  6.90  4.44 Buckingh~    29 female     64
## 3  1002   228       92    37  6.20  4.64 Buckingh~    58 female     61
## 4  1003    78       93    12  6.5   4.63 Buckingh~    67 male       67
## 5  1005   249       90    28  8.90  7.72 Buckingh~    64 male       68
## 6  1008   248       94    69  3.60  4.81 Buckingh~    34 male       71
## # i 9 more variables: weight <int>, frame <chr>, bp.1s <int>,
## #   bp.1d <int>, bp.2s <int>, bp.2d <int>, waist <int>, hip <int>,
## #   time.ppn <int>
```

### 2.2.1 Análisis de variables numéricas

Para hacernos una buena idea de la distribución de los datos contenidos en este conjunto, vamos a calcular los principales estadísticos de sus variables numéricas. Para ello, creamos una función que nos devuelva el valor de todos ellos, empleando a su vez las pequeñas funciones específicas. Nos interesa conocer el porcentaje de NAs por columna, los valores mínimo y máximo, los cuartiles 1 y 3, la mediana, la media y la desviación estándar. En cada parámetro, empleamos el argumento "na.rm = TRUE" para descartar los NAs y evitar posibles complicaciones asociadas a ellos.

Para conocer estos datos de todas las variables numéricas, empleamos la función "lapply()", que permite pasar nuestra función columna por columna. La hacemos dependiente de una estructura condicional que sólo ejecute la función si la columna es numérica. Al final, representamos los resultados en forma de tabla, omitiendo la variable ID (ya que, aunque es numérica, no aporta ninguna información más que el número de paciente).

```r
# Creamos una función para que la podamos reutilizar
estadisticos_columna <- function(col) {
  if (is.numeric(col)) {
    # Si es numérica, devolver resumen estadístico básico y el porcentaje de nulos
    # Para hacer el porcentaje basta con hacer la media del numero de nulos (nºnulos / nº
      ↪  valores de la columna)
    # Para los estadisticos eliminamos los valores faltantes con na.rm=TRUE
    return(c(
      Porcentaje_Nulos = round(mean(is.na(col)) * 100, 2),
      Min = round(min(col, na.rm = TRUE),2),
      Q1 = round(quantile(col, 0.25, na.rm = TRUE),2),
      Median = round(median(col, na.rm = TRUE),2),
      Mean = round(mean(col, na.rm = TRUE),2),
      Sd = round(sd(col, na.rm = TRUE),2),
      Q3 = round(quantile(col, 0.75, na.rm = TRUE),2),
      Max = round(max(col, na.rm = TRUE),2)
    ))
  }
}

# creamos otra función para aplicar el resumen estadístico a un data.frame
estadisticas_numericas <- function(df) {

# Escogemos solo las variables numéricas
columnas_numericas <- df[,sapply(df, function(x) is.numeric(x))]

# Aplicamos la funcion de estadísticos para cada variable
estadisticos <- lapply(columnas_numericas,estadisticos_columna)

# Convertir lista de vectores a data.frame
tabla_resumen <- do.call(rbind, estadisticos)

# Cramos el dataframe final añadiendo el nombre de las variables como primera columna
tabla_resumen <- data.frame(
  Variable = rownames(tabla_resumen),
  tabla_resumen, row.names = NULL)

# Mostrar la tabla de forma bonita
knitr::kable(tabla_resumen,
             caption = "Estadísticos de variables numéricas ",
             align = "lrrrrrrrrr")
}

estadisticas_numericas(diabetes_df[, -1])
```

Table 1: Estadísticos de variables numéricas

| Variable | Porcentaje_Nulos | Min | Q1.25. | Median | Mean | Sd | Q3.75. | Max |
|---|---|---|---|---|---|---|---|---|
| chol | 0.25 | 78.00 | 179.00 | 204.00 | 207.85 | 44.45 | 230.00 | 443.00 |
| stab.glu | 0.00 | 48.00 | 81.00 | 89.00 | 106.67 | 53.08 | 106.00 | 385.00 |
| hdl | 0.25 | 12.00 | 38.00 | 46.00 | 50.45 | 17.26 | 59.00 | 120.00 |
| ratio | 0.25 | 1.50 | 3.20 | 4.20 | 4.52 | 1.73 | 5.40 | 19.30 |
| glyhb | 3.23 | 2.68 | 4.38 | 4.84 | 5.59 | 2.24 | 5.60 | 16.11 |

| Variable | Porcentaje_Nulos | Min | Q1.25. | Median | Mean | Sd | Q3.75. | Max |
|---|---|---|---|---|---|---|---|---|
| age | 0.00 | 19.00 | 34.00 | 45.00 | 46.85 | 16.31 | 60.00 | 92.00 |
| height | 1.24 | 52.00 | 63.00 | 66.00 | 66.02 | 3.92 | 69.00 | 76.00 |
| weight | 0.25 | 99.00 | 151.00 | 172.50 | 177.59 | 40.34 | 200.00 | 325.00 |
| bp.1s | 1.24 | 90.00 | 121.25 | 136.00 | 136.90 | 22.74 | 146.75 | 250.00 |
| bp.1d | 1.24 | 48.00 | 75.00 | 82.00 | 83.32 | 13.59 | 90.00 | 124.00 |
| bp.2s | 65.01 | 110.00 | 138.00 | 149.00 | 152.38 | 21.71 | 161.00 | 238.00 |
| bp.2d | 65.01 | 60.00 | 84.00 | 92.00 | 92.52 | 11.56 | 100.00 | 124.00 |
| waist | 0.50 | 26.00 | 33.00 | 37.00 | 37.90 | 5.73 | 41.00 | 56.00 |
| hip | 0.50 | 30.00 | 39.00 | 42.00 | 43.04 | 5.66 | 46.00 | 64.00 |
| time.ppn | 0.74 | 5.00 | 90.00 | 240.00 | 341.25 | 309.54 | 517.50 | 1560.00 |

### 2.2.2 Transformaciones y creación de variables sintéticas

Vamos a descartar inicialmente las variables categóricas para poder hacer análisis estadísticos de las numéricas. Además, tenemos que transformar las magnitudes medidas para que estén en las unidades del sistema decimal, puesto que el peso está en libras y la altura en pulgadas.

```
# función que pasa de pulgadas a metros
inches_to_meters <- function(inches) {
  meters <- inches * 0.0254
  return(meters)
}

# función que pasa de libras a kilos
pounds_to_kilos <- function(pounds){
  kilos <- pounds * 0.453592
}

# altura, cintura y cadera de pulgadas a metros:

variables_en_pulgadas <- c("height","hip","waist")

diabetes_df[variables_en_pulgadas] <- lapply(
  diabetes_df[variables_en_pulgadas],
  inches_to_meters
)

# peso de libras a kilos
diabetes_df$weight <- pounds_to_kilos(diabetes_df$weight)
```

Asimismo, utilizaremos las variables numéricas disponibles para generar otras nuevas que, basándonos en la literatura, consideramos informativas. Concretamente, el índice de masa corporal (BMI; peso/altura^2) y la relación cadera/cintura (WHR).

```
# indice de masa corporal
diabetes_df$BMI <- diabetes_df$weight / diabetes_df$height^2

# ratio cintura cadera (Waist Hip Ratio)
diabetes_df$WHR <- diabetes_df$waist / diabetes_df$hip
```

Después de hacer las transformaciones necesarias y la creación de nuevas variables, veamos como queda finalmente nuestro conjunto de datos:

```
# estadísticos principales, quitamos columna de ID
estadisticas_numericas(diabetes_df[, -1])
```

Table 2: Estadísticos de variables numéricas

| Variable | Porcentaje_Nulos | Min | Q1.25. | Median | Mean | Sd | Q3.75. | Max |
|---|---|---|---|---|---|---|---|---|
| chol | 0.25 | 78.00 | 179.00 | 204.00 | 207.85 | 44.45 | 230.00 | 443.00 |
| stab.glu | 0.00 | 48.00 | 81.00 | 89.00 | 106.67 | 53.08 | 106.00 | 385.00 |
| hdl | 0.25 | 12.00 | 38.00 | 46.00 | 50.45 | 17.26 | 59.00 | 120.00 |
| ratio | 0.25 | 1.50 | 3.20 | 4.20 | 4.52 | 1.73 | 5.40 | 19.30 |
| glyhb | 3.23 | 2.68 | 4.38 | 4.84 | 5.59 | 2.24 | 5.60 | 16.11 |
| age | 0.00 | 19.00 | 34.00 | 45.00 | 46.85 | 16.31 | 60.00 | 92.00 |
| height | 1.24 | 1.32 | 1.60 | 1.68 | 1.68 | 0.10 | 1.75 | 1.93 |
| weight | 0.25 | 44.91 | 68.49 | 78.24 | 80.55 | 18.30 | 90.72 | 147.42 |
| bp.1s | 1.24 | 90.00 | 121.25 | 136.00 | 136.90 | 22.74 | 146.75 | 250.00 |
| bp.1d | 1.24 | 48.00 | 75.00 | 82.00 | 83.32 | 13.59 | 90.00 | 124.00 |
| bp.2s | 65.01 | 110.00 | 138.00 | 149.00 | 152.38 | 21.71 | 161.00 | 238.00 |
| bp.2d | 65.01 | 60.00 | 84.00 | 92.00 | 92.52 | 11.56 | 100.00 | 124.00 |
| waist | 0.50 | 0.66 | 0.84 | 0.94 | 0.96 | 0.15 | 1.04 | 1.42 |
| hip | 0.50 | 0.76 | 0.99 | 1.07 | 1.09 | 0.14 | 1.17 | 1.63 |
| time.ppn | 0.74 | 5.00 | 90.00 | 240.00 | 341.25 | 309.54 | 517.50 | 1560.00 |
| BMI | 1.49 | 15.20 | 24.13 | 27.80 | 28.79 | 6.61 | 32.24 | 55.79 |
| WHR | 0.50 | 0.68 | 0.83 | 0.88 | 0.88 | 0.07 | 0.92 | 1.14 |

Es cierto que hay muchas variables que contienen NAs, pero en la mayoría de los casos son minoritarios. Las excepciones son las variables bp.2s y bp.2d, de las cuales faltan más de la mitad de observaciones. Por esta razón, hemos decidido descartar estas 2 variables de nuestro análisis.

### 2.2.3 Análisis de variables categoricas

Con el objeto de poder trabajar mejor con los datos, eliminamos los NAs y factorizamos las variables categóricas. Esto permite evitar errores en los cálculos y representaciones, y facilitar la creación de gráficos con una variable independiente no numérica, respectivamente.

```
columnas_factorizables <- c("location", "gender", "frame")

diabetes_df[columnas_factorizables] <- lapply(
  diabetes_df[columnas_factorizables],
  as.factor)

  # Eliminamos el nivel extra de Frame
diabetes_df <- diabetes_df[diabetes_df$frame != "", ]  # Filtra filas con factor
  ↪  diferente de ""
diabetes_df$frame <- droplevels(diabetes_df$frame)    # Elimina el nivel vacío de los
  ↪  factores


str(diabetes_df[columnas_factorizables])
```

```
## 'data.frame':    391 obs. of  3 variables:
##  $ location: Factor w/ 2 levels "Buckingham","Louisa": 1 1 1 1 1 1 1 1 1 1 ...
##  $ gender  : Factor w/ 2 levels "female","male": 1 1 1 2 2 2 2 2 2 1 ...
##  $ frame   : Factor w/ 3 levels "large","medium",..: 2 1 1 1 2 1 2 2 1 3 ...
```

```
  # Factorizamos la edad
diabetes_df$age <- cut(diabetes_df$age,   # partimos la variable...
  breaks = seq(0, 100, by = 10),          # ...en intervalos de 10 años
  right = FALSE,                 # intervalo cerrado a la izquierda [x,y)
  include.lowest = TRUE          # incluye el primer valor en el primer grupo
)
```

### 2.2.4 Objetivos del estudio

Una vez observados los datos, nos hemos planteado las siguientes preguntas/objetivos:

1) En base al nivel de hemoglobina glicosilada, queremos saber, basándonos en la literatura, si el paciente puede ser clasificado como diabético. Hemos visto que normalmente se considera que una persona es diabética si presenta un nivel de GlyHb del 6,5% o más.

2) En base a la tensión arterial, investigaremos si el paciente es hiper- o hipotenso. Se considera hipertenso si la PS es igual/mayor de 140 mmHg y la PD es igual/mayor de 90 mmHg. Se considera hipotenso si la PS es menor de 90 y la PD menor de 60.

3) Estudiaremos la relación cintura/cadera (WHR) para ver cómo se relaciona este parámetro con la presencia/ausencia de diabetes diagnosticada.

4) Estudiaremos el índice de masa corporal (BMI) para compararlo el WHR y con la presencia/ausencia de diabetes diagnosticada.

5) Estudiaremos las relaciones entre las variables para ver qué parámetro/combinación de parámetros guarda mayor relación con la presencia o ausencia de diabetes.

6) Intentaremos crear un modelo, basado en estos datos, que sirva para clasificar otros grupos de pacientes en "diabéticos" y "no diabéticos".

### 2.2.5 Sintesis de variables categóricas

```
-   Diabetes: $SI/NO$. Supondremos diabetes tipo 2 si el porcentaje de glucosa o azucar
en sangre (glyhb) supera el 6.5% . Para simplificar no crearemos el grupo de prediabetes
(glyhb entre 5.7 y 6.4%)

-   Hipertenso: $SI/NO$. Supondremos hypertensión si la presión sistólica es mayor de 140
y la diastólica es mayor de 90
```

```
# suponemos que es diabético si glyhb >= 6.5
diabetes_df$diabetes <- cut(diabetes_df$glyhb,
                            breaks = c(0,  6.49, Inf),
                            labels = c("NO","SI"),
                            right = TRUE) # right=TRUE => intervalo es (a, b]

diabetes_df$hipertenso <- as.factor(ifelse(
```

```
                                    diabetes_df$bp.1s >= 140 &
                                    diabetes_df$bp.1d >= 90,
                                    "SI",
                                    "NO")
                                  )
summary(diabetes_df[,c("diabetes","hipertenso")])
```

```
##  diabetes   hipertenso
##  NO  :316   NO  :293
##  SI  : 63   SI  : 94
##  NA's: 12   NA's:  4
```

# 3 Sección 3. Análisis exploratorio de los datos

## 3.1 3.1 Análisis descriptivo

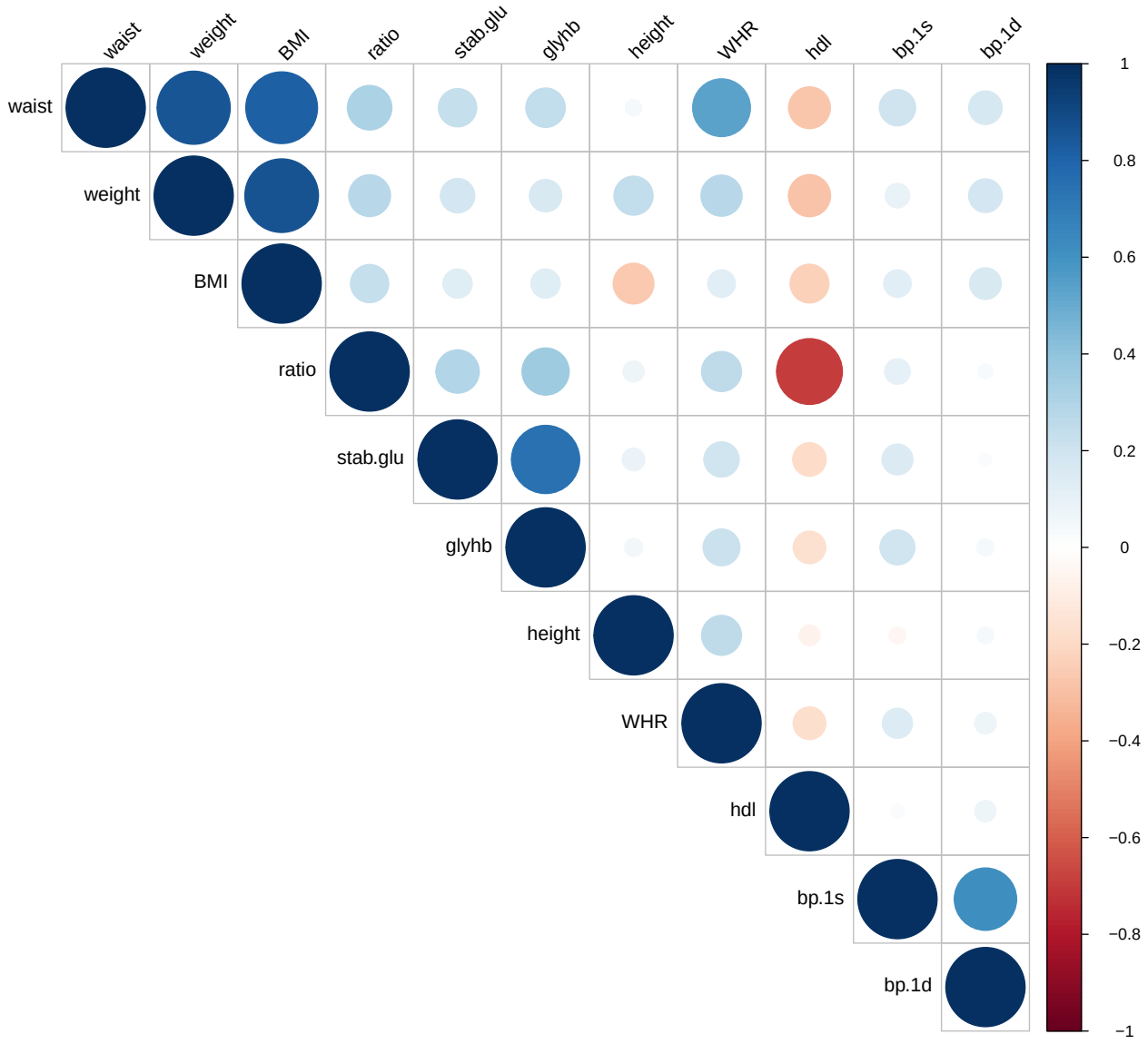
```
# quitamos las columnas que no nos interesan
diabetes_reducido <- diabetes_df %>%
  select(-c(id, bp.2s, bp.2d))

# ahora eliminamos los NAs
diabetes_df_limpio <- na.omit(diabetes_reducido)

# vemos los resultados
head(diabetes_df_limpio)
```

```
## # A tibble: 6 x 20
##    chol stab.glu   hdl ratio glyhb location age   gender height weight
##   <int>    <int> <int> <dbl> <dbl> <fct>    <fct> <fct>   <dbl>  <dbl>
## 1   203       82    56  3.60  4.31 Bucking~ [40,~ female   1.57   54.9
## 2   165       97    24  6.90  4.44 Bucking~ [20,~ female   1.63   98.9
## 3   228       92    37  6.20  4.64 Bucking~ [50,~ female   1.55  116.
## 4    78       93    12  6.5   4.63 Bucking~ [60,~ male     1.70   54.0
## 5   249       90    28  8.90  7.72 Bucking~ [60,~ male     1.73   83.0
## 6   248       94    69  3.60  4.81 Bucking~ [30,~ male     1.80   86.2
## # i 10 more variables: frame <fct>, bp.1s <int>, bp.1d <int>,
## #   waist <dbl>, hip <dbl>, time.ppn <int>, BMI <dbl>, WHR <dbl>,
## #   diabetes <fct>, hipertenso <fct>
```

Para investigar posibles correlaciones entre las variables que podamos usar para guiar nuestro análisis, vamos a obtener la matriz de correlaciones y a representarla de gráficaemnte. Hemos escogido la herramienta corrplot() del paquete homónimo, ya que permite visualizar las relaciones entre variables de una forma muy sencilla.

```
# seleccionamos las columnas numéricas
columnas_numericas <- diabetes_df_limpio[,sapply(diabetes_df_limpio, function(x)
  ↪  is.numeric(x))]
```

```r
# creamos la matriz de correlación de las variables numéricas (quitando ID)
matriz_correlacion <- cor(columnas_numericas[, -c(1, 11, 12)], method = "pearson", use =
  ↪  "pairwise.complete.obs")

# representamos los datos de manera más amigable empleando la función corrplot()
corrplot(matriz_correlacion,
             method = "circle",      # Usamos objetos circulares
             type = "upper",         # Mostramos la esquina superior de la matriz
             tl.col = "black",       # Etiquetas con texto en negro
             tl.srt = 45,            # Rotamos el texto de las etiquetas
             addCoef.col =  NULL,    # Añadimos coeficientes si el método es 'number' o el
               ↪  tipo es 'full'
             number.cex = 0.7,       # Tamaño de los números si se muestran
             order = "hclust",       # Reordenamos las variables por clustering jerárquico
               ↪  para ver patrones
             na.label = "NA",        # Etiquetamos los NAs como tales
             na.label.col = "grey"
            )
```



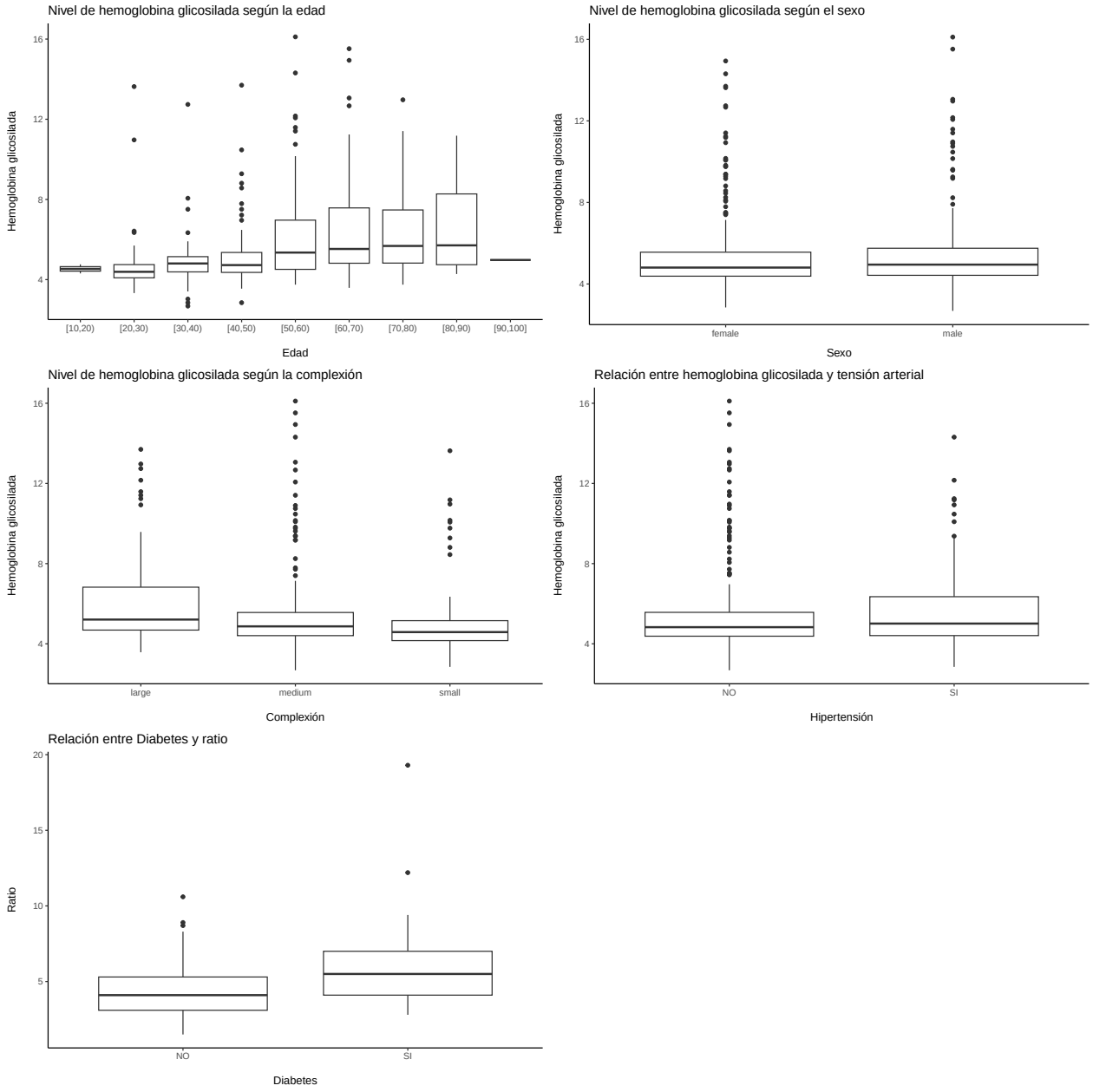

Ahora, vamos a representar distintos gráficos que nos ayuden a evaluar visualmente estas relaciones entre variables. Hemos seleccionado el gráfico de cajas y bigotes (boxplot) para estudiar las relaciones entre variables categóricas y numéricas. Hemos empleado gráficos de dispersión para enfrentar variables numéricas entre sí. Nos serviremos de las herramientas del paquete "ggplot2" para hacer gráficos de manera flexible. Con la función "plot_grid()" hemos conseguido agrupar varios de estos gráficos en el mismo lienzo.

1) Boxplots

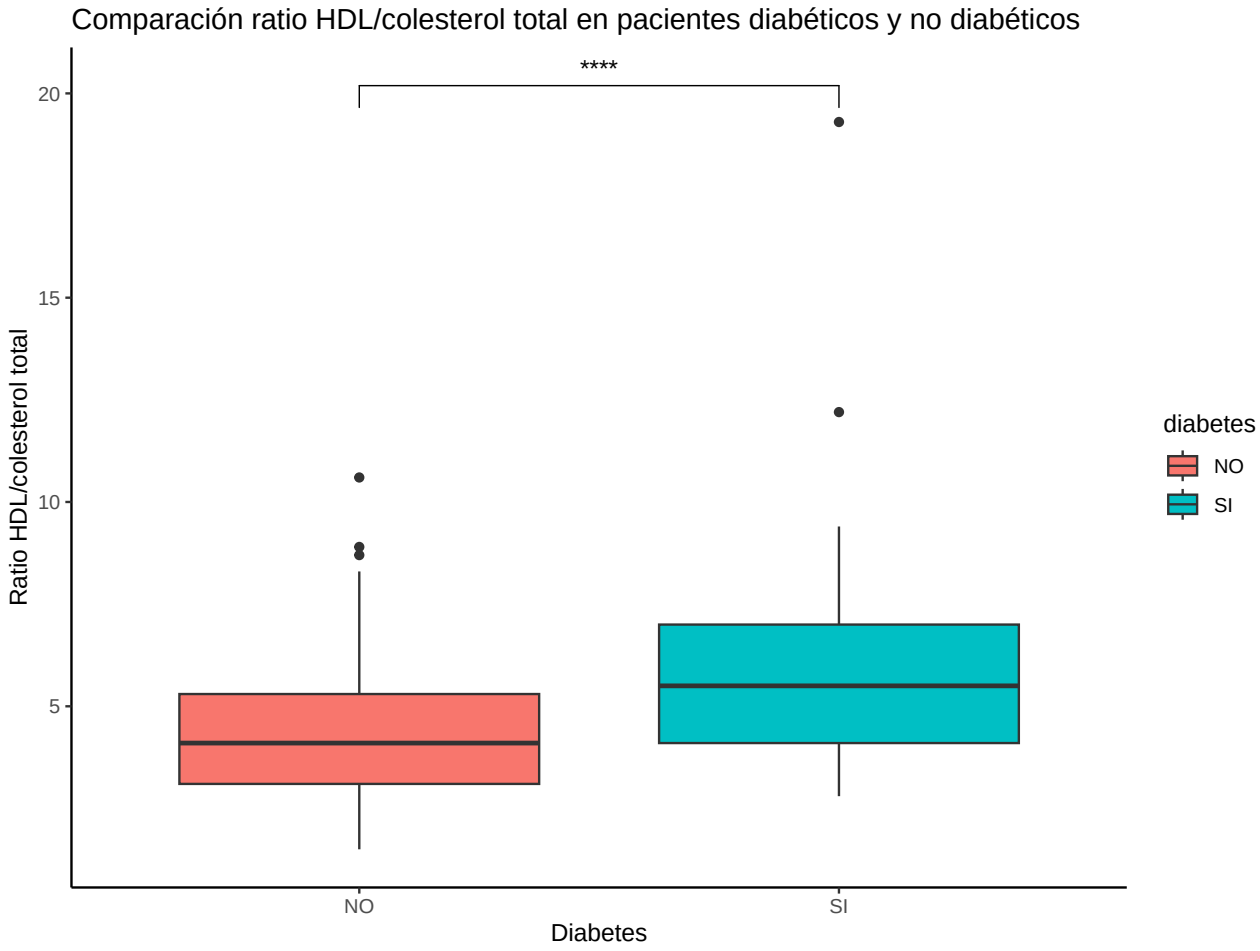
```
# representamos el % de hemoglobina glicosilada (y) frente a la edad (x):
grafico_1 <- ggplot(diabetes_df_limpio, aes (x = age, y = glyhb)) +
  geom_boxplot() +      # Definimos tipo de gráfico
  xlab("Edad") +      # Etiqueta eje X
  ylab ("Hemoglobina glicosilada") +     # Etiqueta eje Y
  ggtitle ("Nivel de hemoglobina glicosilada según la edad") +     # Título gráfico
 theme_classic() +
  theme(
    axis.title.x = element_text(margin = margin(t = 15)),
```

```
    axis.title.y = element_text(margin = margin(r = 15))
  )

# representamos el % de hemoglobina glicosilada (y) frente al sexo (x):
grafico_2 <- ggplot(diabetes_df_limpio, aes (x = gender, y = glyhb)) +
  geom_boxplot() +     # Definimos tipo de gráfico
  xlab("Sexo") +     # Etiqueta eje X
  ylab ("Hemoglobina glicosilada") +    # Etiqueta eje Y
  ggtitle ("Nivel de hemoglobina glicosilada según el sexo") +    # Título gráfico
 theme_classic() +
  theme(
    axis.title.x = element_text(margin = margin(t = 10)),
    axis.title.y = element_text(margin = margin(r = 10))
  )

# representamos el % de hemoglobina glicosilada (y) frente a la constitución (x):
grafico_3 <- ggplot(diabetes_df_limpio, aes (x = frame, y = glyhb)) +
  geom_boxplot() +     # Definimos tipo de gráfico
  xlab("Complexión") +     # Etiqueta eje X
  ylab ("Hemoglobina glicosilada") +    # Etiqueta eje Y
  ggtitle ("Nivel de hemoglobina glicosilada según la complexión") +    # Título gráfico
 theme_classic() +
  theme(
    axis.title.x = element_text(margin = margin(t = 15)),
    axis.title.y = element_text(margin = margin(r = 15))
  )

# representamos el % de hemoglobina glicosilada (y) frente a la presencia/ausencia de
  ↪  hipertensión (x):
grafico_4 <- ggplot(diabetes_df_limpio, aes (x = hipertenso, y = glyhb)) +
  geom_boxplot() +     # Definimos tipo de gráfico
  xlab("Hipertensión") +     # Etiqueta eje X
  ylab ("Hemoglobina glicosilada") +    # Etiqueta eje Y
  ggtitle ("Relación entre hemoglobina glicosilada y tensión arterial") +    # Título
    ↪  gráfico
 theme_classic() +
  theme(
    axis.title.x = element_text(margin = margin(t = 15)),
    axis.title.y = element_text(margin = margin(r = 15))
  )

# representamos la relación HDL/colesterol total (y) frente a la presencia/ausencia de
  ↪  diabetes (x):
grafico_5 <- ggplot(diabetes_df_limpio, aes (x = diabetes, y = ratio)) +
  geom_boxplot() +     # Definimos tipo de gráfico
  xlab("Diabetes") +     # Etiqueta eje X
  ylab ("Ratio") +    # Etiqueta eje Y
  ggtitle ("Relación entre Diabetes y ratio") +    # Título gráfico
 theme_classic() +
  theme(
    axis.title.x = element_text(margin = margin(t = 15)),
    axis.title.y = element_text(margin = margin(r = 15))
  )
```

```
# agrupamos los gráficos en la misma figura
plot_grid(grafico_1, grafico_2, grafico_3, grafico_4, grafico_5,  ncol = 2, nrow =3)
```




Como podría haber una diferencia significativa entre los 2 grupos en el último gráfico, vamos a estudiarlo más detalladamente, añadiendo un test de Wilcoxon-Silk (no paramétrico) para comparar ambos grupos de valores. Mostraremos el resultado en el propio gráfico, empleando la función stat_compare_means(), que nos permite generar los asteriscos en función del nivel de significación.

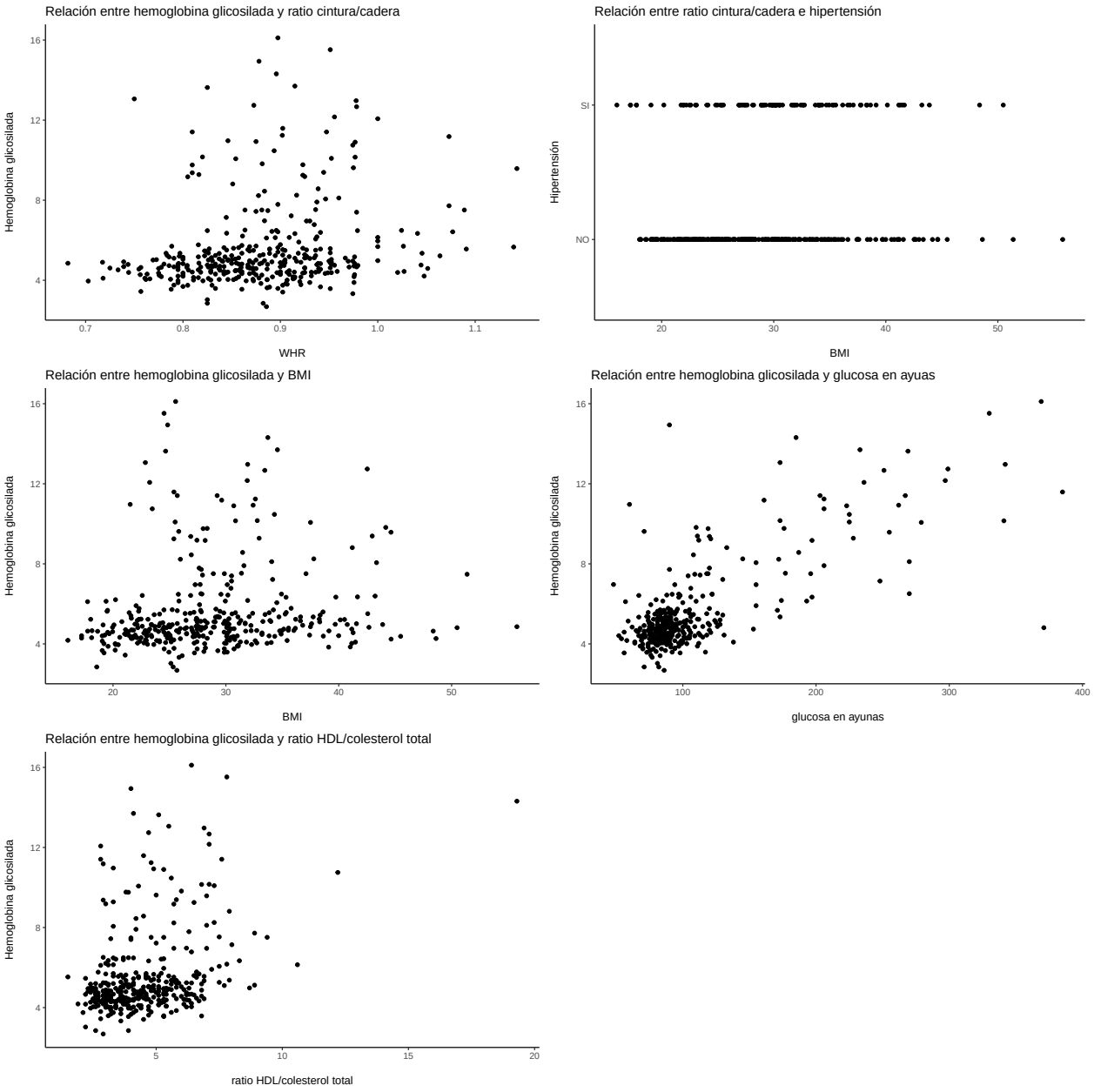



Como podemos observar, sí hay diferencias significativas entre ambos grupos para la variable "ratio HDL/colesterol total". El valor de p es menor de 0.0001, ya que el test de Wilcoxon-Silk arroja 4 asteriscos. Esto sugiere que esta variable podría ser útil en los pasos posteriores de nuestro estudio, donde buscaremos discriminar entre pacientes diabéticos y no diabéticos.

2) Gráficos de dispersión

```
# representamos el % de hemoglobina glicosilada (y) frente a la relación cintura/cadera
  ↪  (x):
grafico_7 <- ggplot(diabetes_df_limpio, aes (x = WHR, y = glyhb)) +
  geom_point() +     # Definimos tipo de gráfico
  xlab("WHR") +     # Etiqueta eje X
  ylab ("Hemoglobina glicosilada") +    # Etiqueta eje Y
  ggtitle ("Relación entre hemoglobina glicosilada y ratio cintura/cadera") +    # Título
    ↪  gráfico
 theme_classic() +
  theme(
    axis.title.x = element_text(margin = margin(t = 15)),
    axis.title.y = element_text(margin = margin(r = 15))
  )

# representamos el BMI (y) frente a la presencia/ausencia de hipertensión (x):
grafico_8 <- ggplot(diabetes_df_limpio, aes (x = BMI, y = hipertenso)) +
```

```r
  geom_point() +     # Definimos tipo de gráfico
  xlab("BMI") +     # Etiqueta eje X
  ylab ("Hipertensión") +    # Etiqueta eje Y
  ggtitle ("Relación entre ratio cintura/cadera e hipertensión") +    # Título gráfico
 theme_classic() +
  theme(
    axis.title.x = element_text(margin = margin(t = 15)),
    axis.title.y = element_text(margin = margin(r = 15))
  )

# representamos el % de hemoglobina glicosilada (y) frente al BMI (x):
grafico_9 <- ggplot(diabetes_df_limpio, aes (x = BMI, y = glyhb)) +
  geom_point() +     # Definimos tipo de gráfico
  xlab("BMI") +     # Etiqueta eje X
  ylab ("Hemoglobina glicosilada") +    # Etiqueta eje Y
  ggtitle ("Relación entre hemoglobina glicosilada y BMI") +    # Título gráfico
 theme_classic() +
  theme(
    axis.title.x = element_text(margin = margin(t = 15)),
    axis.title.y = element_text(margin = margin(r = 15))
  )

# representamos el % de hemoglobina glicosilada (y) frente al nivel de glucosa en ayunas
  ↪  (x):
grafico_10 <- ggplot(diabetes_df_limpio, aes (x = stab.glu, y = glyhb)) +
  geom_point() +     # Definimos tipo de gráfico
  xlab("glucosa en ayunas") +     # Etiqueta eje X
  ylab ("Hemoglobina glicosilada") +    # Etiqueta eje Y
  ggtitle ("Relación entre hemoglobina glicosilada y glucosa en ayuas") +    # Título
    ↪  gráfico
 theme_classic() +
  theme(
    axis.title.x = element_text(margin = margin(t = 15)),
    axis.title.y = element_text(margin = margin(r = 15))
  )

# representamos el % de hemoglobina glicosilada (y) frente a la relación HDL/colesterol
  ↪  total (x):
grafico_11 <- ggplot(diabetes_df_limpio, aes (x = ratio, y = glyhb)) +
  geom_point() +     # Definimos tipo de gráfico
  xlab("ratio HDL/colesterol total") +     # Etiqueta eje X
  ylab ("Hemoglobina glicosilada") +    # Etiqueta eje Y
  ggtitle ("Relación entre hemoglobina glicosilada y ratio HDL/colesterol total") +    #
    ↪  Título gráfico
 theme_classic() +
  theme(
    axis.title.x = element_text(margin = margin(t = 15)),
    axis.title.y = element_text(margin = margin(r = 15))
  )

# agrupamos los gráficos en la misma figura
plot_grid(grafico_7, grafico_8, grafico_9, grafico_10, grafico_11, ncol = 2, nrow = 3)
```




Las meras relaciones entre variables numéricas parecen, en principio, menos informativas en este caso.

## 3.2 3.2 Estudio probabilístico

Aunque no es el objetivo central de nuestro análisis, vamos a plantear una serie de supuestos probabilísticos para explorar cómo estas herramientas nos pueden permitir llevar a cabo predicciones basadas en nuestros datos.

Lo primero es conocer la frecuencia en nuestros datos de las dos condiciones que queremos estudiar: la diabetes y la hipertensión.

```
# tabla de frecuencias de diabetes
frec<- table(diabetes_df_limpio$diabetes)/length(diabetes_df_limpio$diabetes)
```

```r
print(frec) #tabla de frecuencias
```

```
##
##        NO        SI
## 0.8333333 0.1666667
```

```r
# tabla de frecuencias de hipertensión
frec<- table(diabetes_df_limpio$hipertenso)/length(diabetes_df_limpio$hipertenso)
print(frec)
```

```
##
##        NO        SI
## 0.7459016 0.2540984
```

Si asumimos que éstas son las probabilidades para la ocurrencia de estos fenómenos (aunque esto es simplificar mucho la cuestión) podemos realizar algunas simulaciones probabilísticas de interés. Por ejemplo, si tuviéramos 1.000 pacientes, ¿cuál sería la probabilidad de que al menos 280 fueran diabéticos? ¿Y de que al menos 600 fueran hipertensos? Otro ejemplo: probabilidad de que al menos 300 tengan ambas patologías. Por último, la probabilidad de que menos del 30% sean diabéticos.

```r
  # Probabilidad de al menos 280 diabéticos
n = 1000 # número de pacientes
p1 = 0.25 # probabilidad de que el paciente sea diabético
val1 = 279 # valor exacto
paste("Probabilidad de que haya 280 o más diabéticos:", 1 - pbinom(val1, n, p1))
```

```
## [1] "Probabilidad de que haya 280 o más diabéticos: 0.0164366603642154"
```

```r
# Probabilidad de que menos de 200 sean diabéticos
paste("Probabilidad de que menos de 200 sean diabéticos:", pbinom (199, 1000, 0.25))
```

```
## [1] "Probabilidad de que menos de 200 sean diabéticos: 8.02932873998618e-05"
```

```r
  # Probabilidad de al menos 600 hipertensos
n = 1000 # número de pacientes
p2 = 0.61 # probabilidad de que el paciente sea hipertenso
val2 = 599 # valor exacto
paste("Probabilidad de que haya 600 o más hipertensos:", 1 - pbinom(val2, n, p2))
```

```
## [1] "Probabilidad de que haya 600 o más hipertensos: 0.752381700781862"
```

```r
  # Probabilidad de al menos 300 hipertensos diabéticos
paste("Probabilidad de que haya al menos 300 hipertensos diabéticos:", (1 - pbinom(299,
  ↪  n, p1)) * (1 - pbinom(299, n, p2)))
```

```
## [1] "Probabilidad de que haya al menos 300 hipertensos diabéticos: 0.0001935903219491"
```

## 3.3 Muestreo

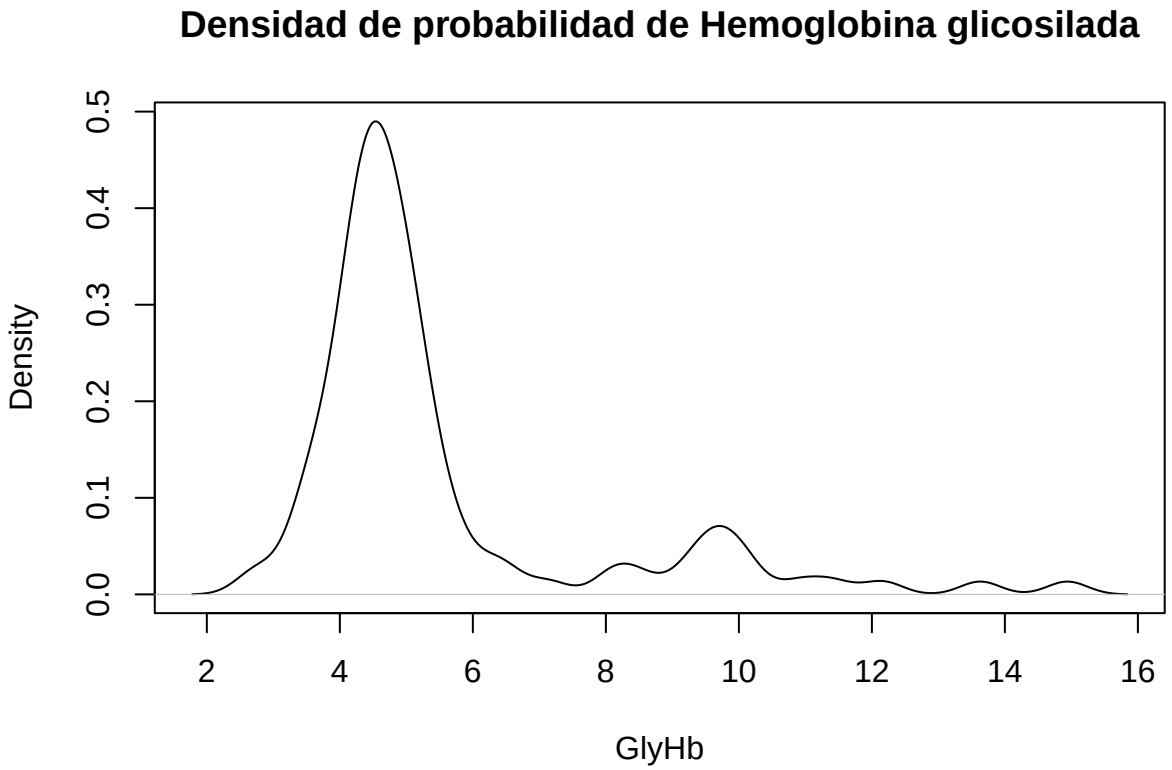
Vamos a sacar una muestra al azar de nuestro grupo de pacientes y a comprobar si una variable cualquiera, como la hemoglobina glicosilada, se distribuye de manera normal en ellos. Primero se hace el muestreo. Luego, verificamos si la distribución es normal, tanto visualmente como con un test estadístico (Shapiro-Wilk).

```
# Sacamos una muestra de nuestros pacientes y estudiamos una variable
set.seed(123) # Semilla para reproducibilidad
n <- 100  # Número de observaciones extraídas
muestra_diabetes <- sample(1:nrow(diabetes_df_limpio), n, replace = FALSE)  # n números
  ↪  al azar entre 1 y número total de filas
observaciones_muestreadas <- diabetes_df_limpio[muestra_diabetes, ]  # índices del daset
  ↪  seleccionados
head(observaciones_muestreadas)  # Con una pequeña muestra de los datos verificamos el
  ↪  resultado
```

```
## # A tibble: 6 x 20
##    chol stab.glu   hdl ratio glyhb location age   gender height weight
##   <int>    <int> <int> <dbl> <dbl> <fct>    <fct> <fct>   <dbl>  <dbl>
## 1   283      145    39  7.30  8.25 Bucking~ [60,~ female   1.55   90.7
## 2   213       83    47  4.5   3.41 Louisa   [30,~ female   1.65   71.2
## 3   155       58    69  2.20  4.17 Bucking~ [20,~ male     1.85   78.9
## 4   261      101    83  3.10  5.12 Louisa   [50,~ female   1.63   89.8
## 5   318      270   108  2.90  6.51 Louisa   [60,~ female   1.65   75.7
## 6   191      155    58  3.30  8.06 Bucking~ [30,~ female   1.57  108.
## # i 10 more variables: frame <fct>, bp.1s <int>, bp.1d <int>,
## #   waist <dbl>, hip <dbl>, time.ppn <int>, BMI <dbl>, WHR <dbl>,
## #   diabetes <fct>, hipertenso <fct>
```

Para analizar visualmente la distribución de la probabilidad de la variable, empleamos una gráfica de densidad para sus distintos valores.

```
# Comprobamos visualmente la normalidad con una gráfica de la densidad de probabilidad
plot(density(observaciones_muestreadas$glyhb),
     main = "Densidad de probabilidad de Hemoglobina glicosilada",
     xlab = "GlyHb")
```




La comprobación estadística se puede hacer con el test de normalidad de Shapiro-Wilk

```
# Comprobación estadística de normalidad con test Shapiro-Wilk
shapiro.test(observaciones_muestreadas$glyhb)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  observaciones_muestreadas$glyhb
## W = 0.73562, p-value = 3.968e-12
```

El valor de p es mucho menor de 0,0001, lo que indica que el riesgo de asumir erróneamente que los datos no tienen una distribución normal es muy bajo. Es decir, podemos asumir no-normalidad con poco riesgo de errar.

# 4 Sección 4. Modelos de aprendizaje automático

## 4.1 Comprobaciones resultados diabetes vs otras variables

La hemoglobina glicosilada nos permite dividir el grupo de pacientes en 2 subgrupos: diabetes vs no diabetes, con un punto de corte clínicamente validado y, por tanto, aplicable a nuestro análisis. A partir de esta clasificación, podemos analizar otras variables y evaluar si presentan diferencias significativas entre los grupos o si su distribución es similar.

Antes de comparar las variables de ambos grupos, debemos saber si la distribución de valores es normal o no. Ello determinará el tipo de test que podemos usar para analizarlas. Para comprobar la normalidad, usamos,

como antes, el test de SHapiro-Wilk. Empleando un bucle for doble, podemos construir una herramienta para explorar la normalidad de múltiples variables en los dos grupos (diabetes vs no diabetes).

```
# lista de variables numéricas a explorar
variables <- c("ratio", "chol", "hdl", "stab.glu", "WHR", "BMI", "bp.1s", "bp.1d")

# inicializamos una lista vacía para guardar los resultados
shapiro_resultados <- list()

# construimos un bucle que seleccione por grupo y variable
for (grupo in c("SI", "NO")) {
  for (var in variables) {
    datos <- diabetes_df_limpio[[var]][diabetes_df_limpio$diabetes == grupo]
    resultado <- shapiro.test(datos)

    # guardamos resultado de una manera identificable
    casos_estudiados <- paste("Shapiro", var, grupo, sep = "__")
    shapiro_resultados[[casos_estudiados]] <- resultado$p.value
  }
}

# mostramos los valores p para cada variable y grupo
print(shapiro_resultados)
```

```
## $Shapiro__ratio__SI
## [1] 5.417877e-08
##
## $Shapiro__chol__SI
## [1] 0.0001964091
##
## $Shapiro__hdl__SI
## [1] 9.122775e-05
##
## $Shapiro__stab.glu__SI
## [1] 0.04932389
##
## $Shapiro__WHR__SI
## [1] 0.0394465
##
## $Shapiro__BMI__SI
## [1] 0.0002877157
##
## $Shapiro__bp.1s__SI
## [1] 0.8201029
##
## $Shapiro__bp.1d__SI
## [1] 0.7569674
##
## $Shapiro__ratio__NO
## [1] 1.258764e-08
##
## $Shapiro__chol__NO
## [1] 0.0001234885
##
```

```
## $Shapiro__hdl__NO
## [1] 2.202469e-11
##
## $Shapiro__stab.glu__NO
## [1] 1.502709e-24
##
## $Shapiro__WHR__NO
## [1] 0.0696977
##
## $Shapiro__BMI__NO
## [1] 9.345255e-08
##
## $Shapiro__bp.1s__NO
## [1] 2.482507e-12
##
## $Shapiro__bp.1d__NO
## [1] 0.05350972
```

La mayoría de las variables tienen un valor p menor de 0.05 en ambos grupos, aunque algunas tienen un valor p mayor de 0.05 en uno de los grupos o en ambos. El valor p mayor de 0.05 indica que probablemente su distribución es normal. Sin embargo, como la mayoría no lo son, vamos a asumir que ninguna lo es.

Lo siguiente que haremos es evaluar si hay diferencias significativas para estas variables entre los dos grupos de pacientes. Emplearemos un test no paramétrico para todas (aunque algunas puedan distribuirse de manera normal), para no complicarnos. Al tener sólo 2 grupos, empleamos Wilcoxon-Silk, que es bueno para casos binomiales.

Para hacer esta comparación de manera rápida, vamos a automatizarla para todas las variables a la vez. Para ello, emplearemos la función sapply() para aplicar a cada variable una función anónima que nos ejecute el test de Wilcoxon-Silk variable a variable.

```
# lista de variables numéricas a explorar
variables <- c("ratio", "chol", "hdl", "stab.glu", "WHR", "BMI", "bp.1s", "bp.1d")

# aplicamos el test WS a cada variable con sapply()
resultados <- sapply(variables, function(var) {
  test <- wilcox.test(as.formula(paste(var, "~ diabetes")), data = diabetes_df_limpio)
  test$p.value
})

# mostramos el valor p para cada variable
print(resultados)
```

```
##        ratio         chol          hdl     stab.glu          WHR
## 3.164580e-07 1.121634e-04 3.399722e-04 1.533248e-23 1.642647e-04
##          BMI        bp.1s        bp.1d
## 1.189749e-03 7.285677e-07 2.512521e-01
```

Dado que todas las variables estudiadas, excepto la tensión arterial diastólica, presentan un valor p menor de 0.05, podemos aceptar la hipótesis de que su distribución es diferente para ambos grupos. Por ello, las tendremos en cuenta en los análisis inferenciales y clasificatorios que realizaremos a continuación.

## 4.2 Regresión lineal simple

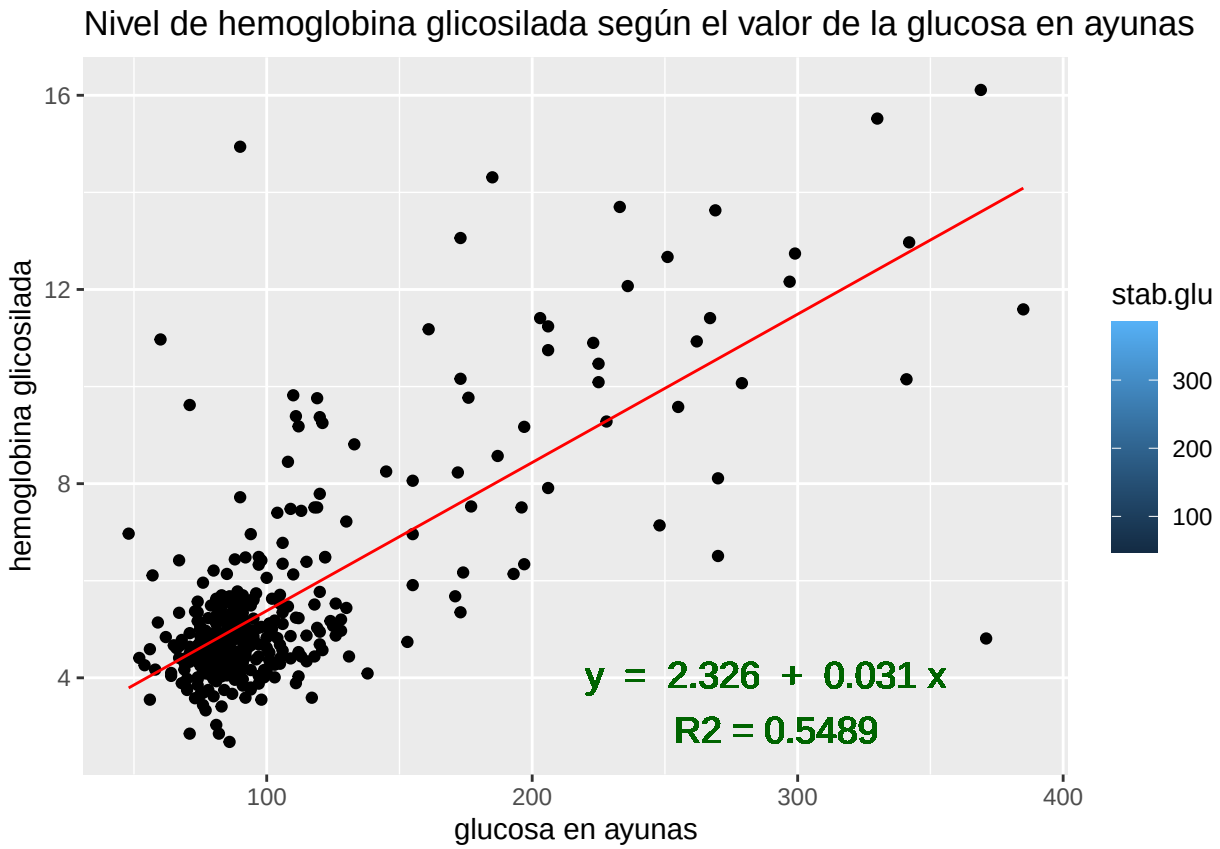
Veamos una representación simple de la matriz de correlaciones:

```
# matriz de correlaciones de columnas numéricas quitando el ID
cor(columnas_numericas[,-1])
```

```
##             stab.glu         hdl       ratio       glyhb       height
## stab.glu  1.00000000 -0.18010488  0.29889570  0.74090490  0.082475702
## hdl      -0.18010488  1.00000000 -0.69023141 -0.16949641 -0.068591817
## ratio     0.29889570 -0.69023141  1.00000000  0.35465342  0.070898165
## glyhb     0.74090490 -0.16949641  0.35465342  1.00000000  0.052250721
## height    0.08247570 -0.06859182  0.07089817  0.05225072  1.000000000
## weight    0.18880052 -0.28298268  0.27889889  0.16776851  0.243295558
## bp.1s     0.15142542  0.02950891  0.10534657  0.19442279 -0.044411815
## bp.1d     0.02569721  0.07224515  0.03484142  0.04786459  0.043452076
## waist     0.23369209 -0.27830010  0.31549761  0.24768684  0.041807866
## hip       0.14483314 -0.22221661  0.20789160  0.15167273 -0.117181984
## time.ppn -0.04845774  0.07993884 -0.05382831  0.03704938 -0.006180895
## BMI       0.13628491 -0.23982601  0.23304631  0.13581359 -0.263980548
## WHR       0.19806921 -0.17034661  0.25587953  0.21569488  0.254434157
##               weight       bp.1s       bp.1d       waist         hip
## stab.glu  0.18880052  0.15142542  0.02569721  0.23369209  0.14483314
## hdl      -0.28298268  0.02950891  0.07224515 -0.27830010 -0.22221661
## ratio     0.27889889  0.10534657  0.03484142  0.31549761  0.20789160
## glyhb     0.16776851  0.19442279  0.04786459  0.24768684  0.15167273
## height    0.24329556 -0.04441181  0.04345208  0.04180787 -0.11718198
## weight    1.00000000  0.09624288  0.18050511  0.85192261  0.82984527
## bp.1s     0.09624288  1.00000000  0.61984558  0.20976399  0.15142640
## bp.1d     0.18050511  0.61984558  1.00000000  0.17899079  0.16282460
## waist     0.85192261  0.20976399  0.17899079  1.00000000  0.83233707
## hip       0.82984527  0.15142640  0.16282460  0.83233707  1.00000000
## time.ppn -0.06221671 -0.07490369 -0.06376264 -0.06586124 -0.09251954
## BMI       0.86399427  0.12110290  0.16004513  0.81953829  0.89039547
## WHR       0.27009942  0.14472557  0.07487932  0.53244464 -0.02197517
##              time.ppn         BMI          WHR
## stab.glu -0.048457737  0.1362849  0.198069209
## hdl       0.079938843 -0.2398260 -0.170346609
## ratio    -0.053828314  0.2330463  0.255879530
## glyhb     0.037049379  0.1358136  0.215694876
## height   -0.006180895 -0.2639805  0.254434157
## weight   -0.062216714  0.8639943  0.270099417
## bp.1s    -0.074903689  0.1211029  0.144725567
## bp.1d    -0.063762636  0.1600451  0.074879321
## waist    -0.065861241  0.8195383  0.532444642
## hip      -0.092519540  0.8903955 -0.021975165
## time.ppn  1.000000000 -0.0604064  0.005364018
## BMI      -0.060406401  1.0000000  0.121734943
## WHR       0.005364018  0.1217349  1.000000000
```

Parece que la variable que mejor correlaciona con la hemoglobina glicosilada es la glucosa en ayunas. Vamos a explorar con un gráfico de dispersión y una regresión lineal esta relación:

```
## `geom_smooth()` using formula = 'y ~ x'
```




Debido a la dispersión de los valores, el coeficiente de determinación (R2) no es muy alto. La glucosa en ayunas, por sí sola, no parece muy útil para predecir la hemoglobina glicosilada.

### 4.3 Regresión lineal múltiple para ver determinación de la diabetes

Intentaremos realizar un modelo mejor con regresión lineal múltiple. Vamos a probar a explicar la variabilidad de la hemoglobina glicosilada con un conjunto de las otras variables que nos parece representativo del total de parámetros:

```
# modelo de regresión múltiple
regmult_diabetes <- lm(glyhb ~ stab.glu + ratio + bp.1s + bp.1d + BMI + WHR, data =
  ↪  diabetes_df_limpio)

summary(regmult_diabetes)
```

```
##
## Call:
## lm(formula = glyhb ~ stab.glu + ratio + bp.1s + bp.1d + BMI +
##     WHR, data = diabetes_df_limpio)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
```

```
## -7.9418 -0.7114 -0.1757  0.5085  9.9169
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.1015792  1.0201569   0.100  0.92074
## stab.glu     0.0280598  0.0015157  18.513  < 2e-16 ***
## ratio        0.1686630  0.0476185   3.542  0.00045 ***
## bp.1s        0.0090838  0.0043225   2.102  0.03629 *
## bp.1d       -0.0057995  0.0072436  -0.801  0.42387
## BMI          0.0006825  0.0120320   0.057  0.95479
## WHR          1.0718824  1.0846829   0.988  0.32372
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.465 on 359 degrees of freedom
## Multiple R-squared:  0.5759, Adjusted R-squared:  0.5688
## F-statistic: 81.25 on 6 and 359 DF,  p-value: < 2.2e-16
```

El coeficiente de determinación (R2) sigue sin ser demasiado bueno. Por suerte, existe una función en R, step(), que nos permite inferir cuál podría ser la combinación óptima de predictores para un modelo. Vamos a emplearla para ver si podemos mejorar nuestra regresión lineal:

```
  # Con la función step() buscamos los mejores predictores para nuestra variable
step(object=regmult_diabetes,direction ="both", trace=1)
```

```
## Start:  AIC=286.48
## glyhb ~ stab.glu + ratio + bp.1s + bp.1d + BMI + WHR
##
##            Df Sum of Sq     RSS    AIC
## - BMI       1      0.01  770.56 284.48
## - bp.1d     1      1.38  771.93 285.13
## - WHR       1      2.10  772.65 285.47
## <none>                    770.55 286.48
## - bp.1s     1      9.48  780.03 288.95
## - ratio     1     26.93  797.48 297.05
## - stab.glu  1    735.65 1506.20 529.79
##
## Step:  AIC=284.48
## glyhb ~ stab.glu + ratio + bp.1s + bp.1d + WHR
##
##            Df Sum of Sq     RSS    AIC
## - bp.1d     1      1.37  771.93 283.13
## - WHR       1      2.11  772.67 283.48
## <none>                    770.56 284.48
## + BMI       1      0.01  770.55 286.48
## - bp.1s     1      9.48  780.03 286.95
## - ratio     1     28.09  798.64 295.58
## - stab.glu  1    739.00 1509.56 528.60
##
## Step:  AIC=283.13
## glyhb ~ stab.glu + ratio + bp.1s + WHR
##
##            Df Sum of Sq     RSS    AIC
```

```
## - WHR       1      2.11  774.04 282.13
## <none>                   771.93 283.13
## + bp.1d     1      1.37  770.56 284.48
## + BMI       1      0.00  771.93 285.13
## - bp.1s     1      8.99  780.92 285.37
## - ratio     1     28.27  800.20 294.29
## - stab.glu  1    748.81 1520.73 529.30
##
## Step:  AIC=282.13
## glyhb ~ stab.glu + ratio + bp.1s
##
##            Df Sum of Sq     RSS    AIC
## <none>                   774.04 282.13
## + WHR       1      2.11  771.93 283.13
## + bp.1d     1      1.37  772.67 283.48
## + BMI       1      0.00  774.04 284.13
## - bp.1s     1     10.07  784.11 284.86
## - ratio     1     32.89  806.94 295.36
## - stab.glu  1    768.97 1543.01 532.62
```

```
##
## Call:
## lm(formula = glyhb ~ stab.glu + ratio + bp.1s, data = diabetes_df_limpio)
##
## Coefficients:
## (Intercept)     stab.glu        ratio        bp.1s
##    0.749856     0.028338     0.179190     0.007297
```

A la luz de los resultados obtenidos, el mejor conjunto de predictores sería: glucosa en ayunas + ratio HDL/colesterol total + tensión arterial sistólica. Vamos a probar el modelo con esos predictores:

```
  # Salida de la función step(): mejor combinación de variables independientes
regmult_diabetes_mejorado <- lm(formula = glyhb ~ stab.glu + ratio + bp.1s, data =
  ↪  diabetes_df_limpio)

summary(regmult_diabetes_mejorado)
```

```
##
## Call:
## lm(formula = glyhb ~ stab.glu + ratio + bp.1s, data = diabetes_df_limpio)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -8.0158 -0.7325 -0.1840  0.4813  9.9744
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 0.749856   0.487002   1.540 0.124498
## stab.glu    0.028338   0.001494  18.964  < 2e-16 ***
## ratio       0.179190   0.045686   3.922 0.000105 ***
## bp.1s       0.007297   0.003362   2.170 0.030642 *
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.462 on 362 degrees of freedom
## Multiple R-squared:  0.574,  Adjusted R-squared:  0.5704
## F-statistic: 162.6 on 3 and 362 DF,  p-value: < 2.2e-16
```

Este modelo no parece significativamente mejor que el anterior para explicar la hemoglobina glicosilada. Por ello, vamos a emplear otro tipo de herramientas para intentar comprender la variabilidad de este paráemtro y poder implementar modelos predictivos mejores.

Empezaremos por un Análisis de Componentes Principales (PCA).

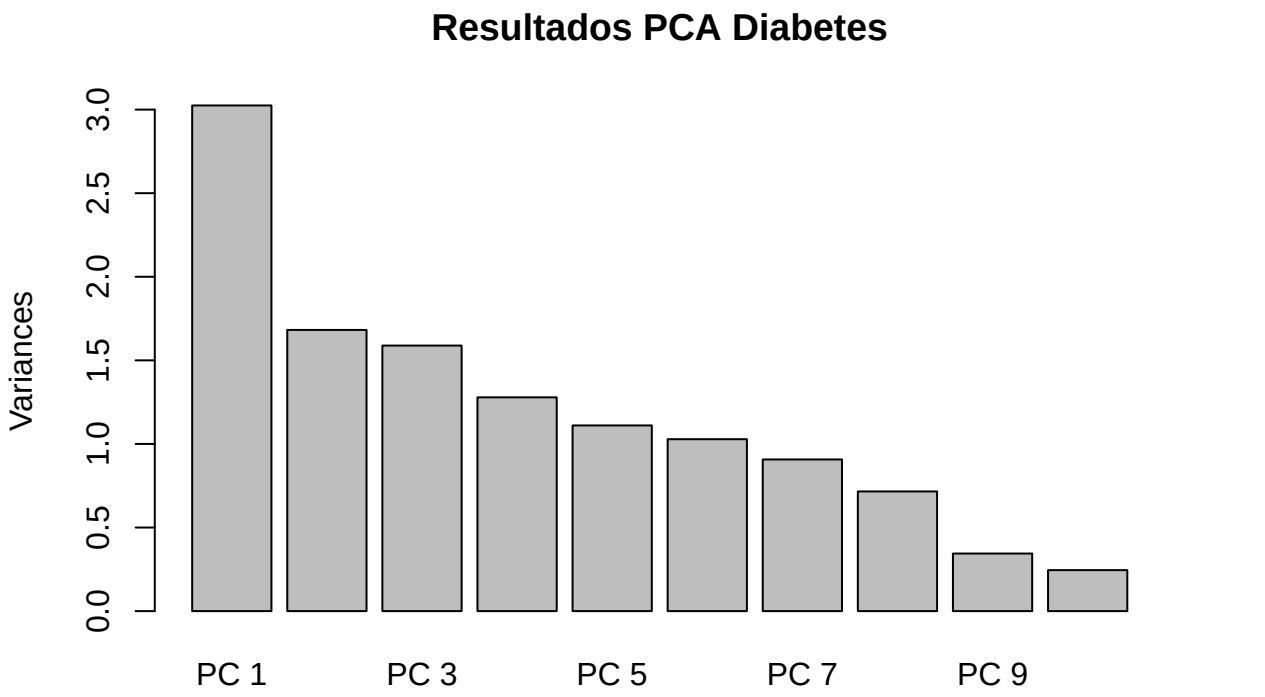
## 4.4 Prueba de un Análisis de Componentes Principales

Vamos a ver si podemos separar los dos grupos de pacientes (diabetes vs no-diabetes) en base a los componentes principales 1 y 2, que explican la mayor parte de la varianza de los datos. Para ello, seleccionamos las variables a emplear, agrupamos los pacientes en base a la presencia/ausencia de diabetes, realizamos el PCA y representamos sus resultados con autoplot().

```
# Seleccionamos solo las columnas numéricas relevantes para el PCA
variables_pca <- c("chol", "stab.glu", "hdl", "ratio", "glyhb", "height",
              "weight", "bp.1s", "bp.1d", "time.ppn", "BMI", "WHR")

# Quitamos filas con NA si hay
datos_completos <- diabetes_df_limpio %>%
  select(all_of(variables_pca), diabetes) %>%
  na.omit()

# Quitamos la columna diabetes porque no es numérica
datos_pca <- datos_completos %>%
  select(-diabetes)

# Guardamos el factor diabetes separadamente
clase_diabetes <- datos_completos$diabetes

# Realizamos el PCA
pca_resultado <- prcomp(datos_pca, scale. = TRUE)

# Analizamos el resultado del PCA
summary(pca_resultado)
```

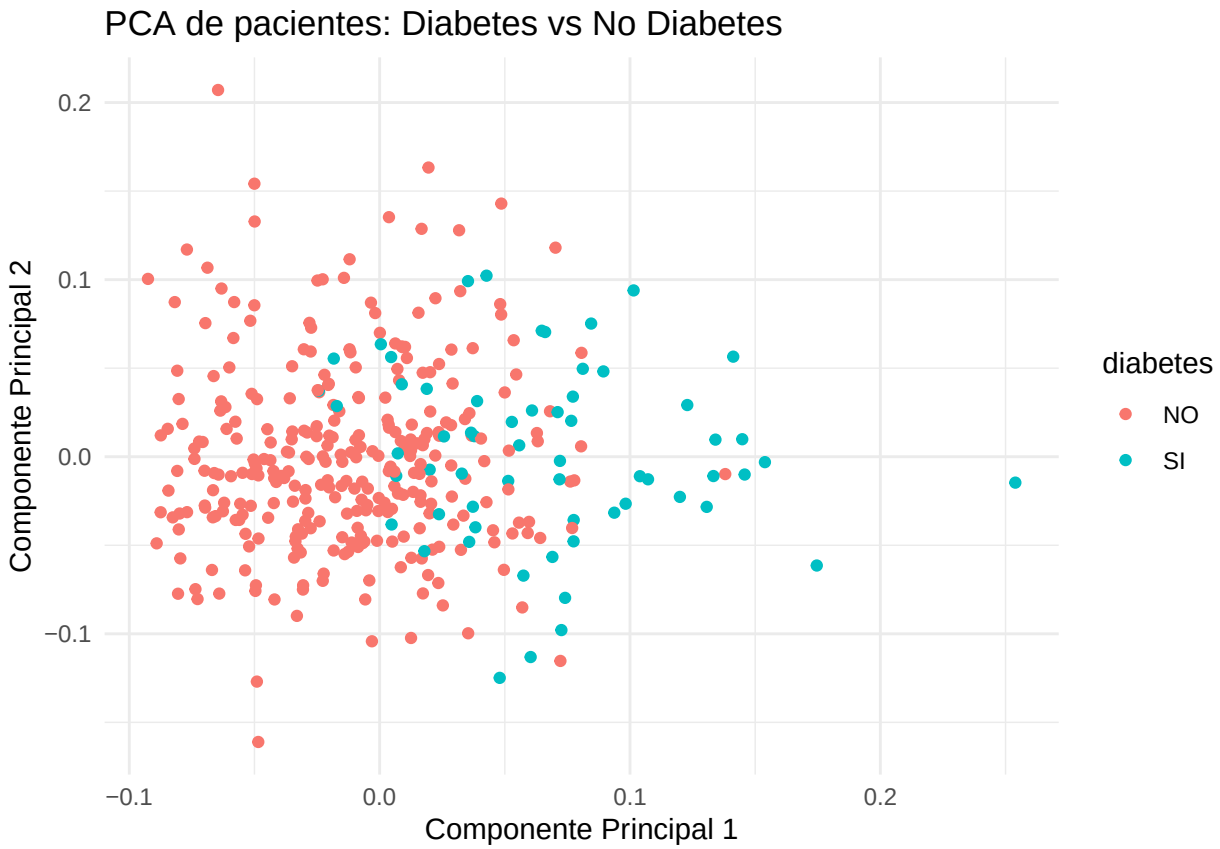
```
## Importance of components:
##                           PC1    PC2    PC3    PC4     PC5    PC6
## Standard deviation     1.7392 1.2969 1.2603 1.1308 1.05391 1.0141
## Proportion of Variance 0.2521 0.1402 0.1324 0.1066 0.09256 0.0857
## Cumulative Proportion  0.2521 0.3922 0.5246 0.6311 0.72371 0.8094
##                           PC7     PC8    PC9    PC10    PC11    PC12
## Standard deviation     0.9525 0.84613 0.5868 0.49510 0.26106 0.07920
## Proportion of Variance 0.0756 0.05966 0.0287 0.02043 0.00568 0.00052
## Cumulative Proportion  0.8850 0.94467 0.9734 0.99380 0.99948 1.00000
```

```r
# Graficamos el resultado
barplot(pca_resultado$sdev[1:10]**2,
        names.arg = paste0("PC ", 1:10),
        col = "grey",
        main = "Resultados PCA Diabetes",
        ylab = "Variances")
```




Para ver si el PCA permite segregar los pacientes con y sin diabetes, vamos a representar gráficamente ambos grupos en función de los primeros PC (1 y 2), que explican la mayoría de la varianza.

```r
# Visualizamos el resultado enfrentando PC1 y PC2 y usando "diabetes" como clasificador
autoplot(pca_resultado, data = datos_completos, colour = "diabetes") +
  labs(title = "PCA de pacientes: Diabetes vs No Diabetes",
       x = "Componente Principal 1",
       y = "Componente Principal 2") +
  theme_minimal()
```




No parece que el análisis de PC permita separar claramente los grupos, lo que sugiere que las variables estudiadas no son excesivamente dependientes de la condición de "diabético". Necesitaremos, pues, otro tipo de herramienta para intentar hacer esta discriminación.

## 4.5 Regresión logística

Vamos a utilizar la regresión logística para llevar a cabo uno de los objetivos principales de este estudio: construir un modelo predictivo que estime la probabilidad de que un paciente sea diabético, basándonos en las variables medidas. Para ello, primero dividiremos nuestros datos en dos conjuntos: uno de entrenamiento (70% observaciones) y otro de prueba (resto de observaciones).

Entrenaremos el modelo con el conjunto de entrenamiento, permitiéndole aprender la relación entre las variables y el diagnóstico de diabetes. Luego, evaluaremos su capacidad predictiva aplicándolo al conjunto de prueba, que contiene datos inéditos. Así podremos medir su eficacia para clasificar correctamente a nuevos pacientes.

Es fundamental que ambos conjuntos, el de entrenamiento y el de prueba, sean representativos de la población total, y que mantengan una distribución homogénea de las clases (diabéticos y no diabéticos), para garantizar la validez del modelo y su aplicación general.

```
#fijamos la semilla
set.seed(666)

# extraemos los indices de las muestras que conformarán el dataset de training
# para ello usamos la funcion createDataPartition de la libreria caret
# https://www.rdocumentation.org/packages/caret/versions/7.0-1/topics/createDataPartition
```

```r
train_index <- createDataPartition(
                    diabetes_df_limpio$diabetes,
                    p = 0.7,
                    list = FALSE,
                    times = 1
)

# definimos el conjunto de entrenamiento
training_set <- diabetes_df_limpio[train_index,]

# definimos el conjunto de prueba (total - conjunto entrenamiento)
test_set <- diabetes_df_limpio[-train_index,]

# validar que el conjunto de entrenamiento está equilibrado:
# vemos si hay una distribución razonable de diabéticos y no diabéticos en cada
# conjunto.
print("training set")
```

```
## [1] "training set"
```

```r
print(summary(training_set[,c("diabetes","hipertenso","frame")]))
```

```
##  diabetes hipertenso    frame
##  NO:214   NO:186     large : 64
##  SI: 43   SI: 71     medium:123
##                      small : 70
```

```r
print("test set")
```

```
## [1] "test set"
```

```r
print(summary(test_set[,c("diabetes","hipertenso","frame")]))
```

```
##  diabetes hipertenso    frame
##  NO:91    NO:87      large :32
##  SI:18    SI:22      medium:49
##                      small :28
```

```r
# creamos el modelo con las variables que decidamos
predictores <- c("hipertenso","frame","BMI","WHR","ratio","stab.glu","chol",
  ↪  "hdl","bp.1s","bp.1d")

formula_str <- paste("diabetes","~", paste(predictores, collapse=" + "))
formula <- as.formula(formula_str)

cat("La formula que utilizaremos es: \n", formula_str)
```

```
## La formula que utilizaremos es:
##  diabetes ~ hipertenso + frame + BMI + WHR + ratio + stab.glu + chol + hdl + bp.1s + bp.1d
```

```r
modelo_logistico <- glm(formula, data=training_set, family = "binomial")
print(summary(modelo_logistico))
```

```
##
## Call:
## glm(formula = formula, family = "binomial", data = training_set)
##
## Coefficients:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -15.750877   5.063518  -3.111  0.00187 **
## hipertensoSI  -0.017503   0.933566  -0.019  0.98504
## framemedium    0.061511   0.631945   0.097  0.92246
## framesmall     0.499552   0.827297   0.604  0.54595
## BMI            0.019002   0.041350   0.460  0.64585
## WHR            5.081299   4.083959   1.244  0.21342
## ratio         -0.275697   0.361083  -0.764  0.44515
## stab.glu       0.042653   0.007662   5.567  2.6e-08 ***
## chol           0.021894   0.011387   1.923  0.05451 .
## hdl           -0.042435   0.040794  -1.040  0.29824
## bp.1s          0.022521   0.012628   1.783  0.07451 .
## bp.1d         -0.008576   0.031321  -0.274  0.78422
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 232.12  on 256  degrees of freedom
## Residual deviance: 107.26  on 245  degrees of freedom
## AIC: 131.26
##
## Number of Fisher Scoring iterations: 6
```

Aplicamos el modelo a los datos de prueba para ver qué tal predice el resultado. Hay que fijar el umbral de probabilidad para que el modelo clasifique en uno u otro grupo. En principio lo vamos a fijar en 0.5, que es un umbral de clasificación estándar que da igual peso a ambas clases. Si la probabilidad predicha de pertenecer a un grupo es mayor de 0.5, asignará el paciente a ese grupo.

Luego, realizamos una matriz de confusión nos permitirá valorar el nivel de acierto: número de verdaderos y falsos positivos, y de verdaderos negativos y falsos negativos. Con esto, obtenemos la precisión, sensibilidad y especificidad del modelo.

```r
prob_test <- predict(modelo_logistico, newdata = test_set, type="response")

# definimos los dos grupos posibles
clase_positiva <- levels(training_set$diabetes)[2] # diabéticos
clase_negativa <- levels(training_set$diabetes)[1] # no diabéticos

# definimos el umbral de clasificación para hacer las predicciones
clase_predicha <- factor(
                    ifelse(prob_test > 0.5 , clase_positiva, clase_negativa),
                    levels = levels(test_set$diabetes))

# generamos la matriz d econfusión
```

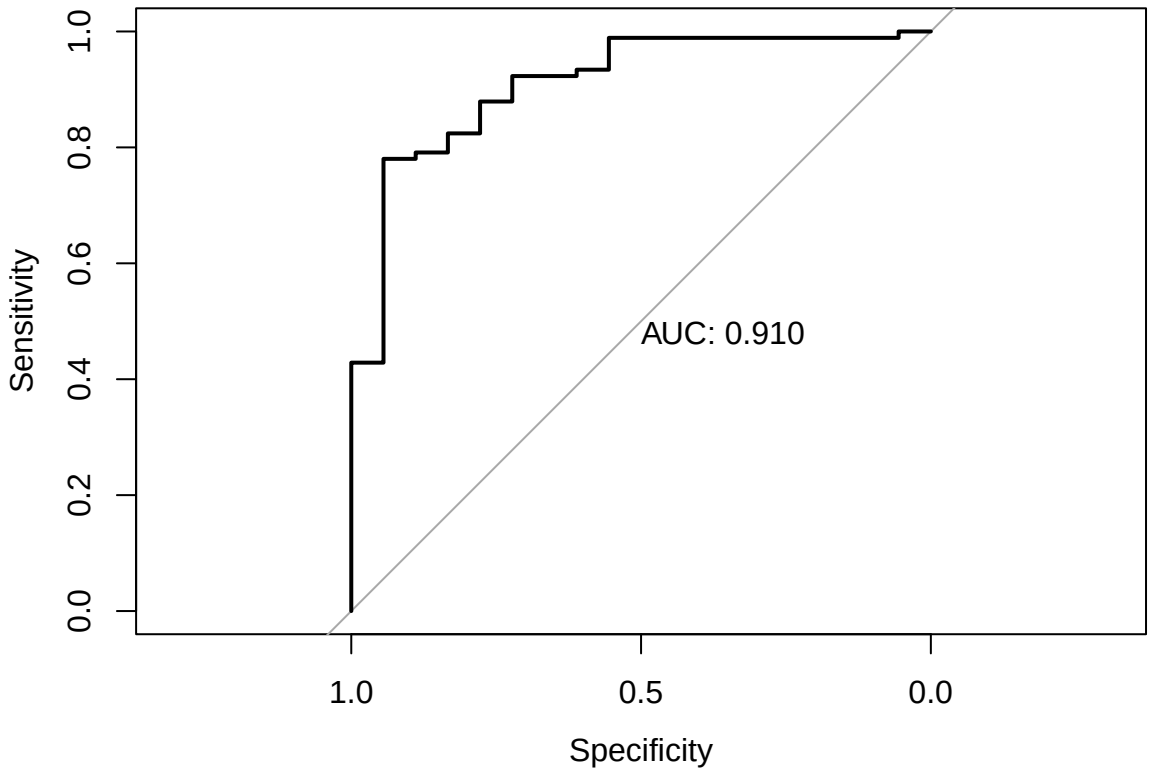
```r
matriz_confusion <- confusionMatrix(data = clase_predicha,
                                    reference = test_set$diabetes,
                                    positive = clase_positiva)
print(matriz_confusion)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction NO SI
##         NO 90 10
##         SI  1  8
##
##                Accuracy : 0.8991
##                  95% CI : (0.8266, 0.9485)
##     No Information Rate : 0.8349
##     P-Value [Acc > NIR] : 0.04074
##
##                   Kappa : 0.5422
##
##  Mcnemar's Test P-Value : 0.01586
##
##             Sensitivity : 0.44444
##             Specificity : 0.98901
##          Pos Pred Value : 0.88889
##          Neg Pred Value : 0.90000
##              Prevalence : 0.16514
##          Detection Rate : 0.07339
##    Detection Prevalence : 0.08257
##       Balanced Accuracy : 0.71673
##
##        'Positive' Class : SI
##
```

La sensibilidad es baja (50%) y el p-value es menor de 0.05, lo que indica que no estamos discriminando las clases con mucha eficacia. Sólo la mitad de los diabéticos han sido clasificados como diabéticos, de ahí ese valor de sensibilidad. La especificidad es muy buena, lo que indica que pocos no diabéticos son clasificados como diabéticos.

Vamos a emplear una curva ROC como método complementario para evaluar la precisión predictiva del modelo. Tenemos que fijarnos en el área bajo la curva (AUC). Si la AUC es cercana a 1, indicará que el modelo es bueno para discriminar entre diabéticos y no diabéticos.

```r
# hacemos el test roc
roc_test <- roc(response = test_set$diabetes,
                predictor = prob_test,
                levels = rev(levels(test_set$diabetes)), direction = ">")

# representamos la curva
plot(roc_test, print.auc = TRUE)
```




El AUC es superior a 0.921, lo cual sugiere que el modelo es muy buen predictor. Es probable que, si usamos otro umbral (distinto de 0.5), la clasificación sea mejor. Vamos a intentar aumentar la sensibilidad del modelo, aunque sea a costa de reducir un poco la especificidad. Preferimos clasificar bien a los diabéticos de entrada, dado que tiene mayor riesgo no detectarlos cuando lo son (error beta) que estimarlos diabéticos sin serlo (error alfa).

Como todavía podríamos mejorar el modelo, usando la función step(), vamos a intentarlo antes de modificar el umbral. Esta función nos indica aproximadamente qué variables explican mejor la variabilidad de los datos, lo que permite descartar las demás y hacer más robusto (en principio) el modelo.

```
# Buscamos el ajuste óptimo con el método stepwise
modelo_step <- step(modelo_logistico, direction = "both", trace = FALSE)
summary(modelo_step)
```

```
##
## Call:
## glm(formula = diabetes ~ stab.glu + chol + bp.1s, family = "binomial",
##     data = training_set)
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -12.359485   2.224392  -5.556 2.75e-08 ***
## stab.glu      0.044350   0.007097   6.249 4.12e-10 ***
## chol          0.013570   0.006550   2.072   0.0383 *
## bp.1s         0.017722   0.009040   1.960   0.0500 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
```

```
##
##     Null deviance: 232.12  on 256  degrees of freedom
## Residual deviance: 111.06  on 253  degrees of freedom
## AIC: 119.06
##
## Number of Fisher Scoring iterations: 6
```

La salida de la función nos sugiere hacer un modelo basado sólo en la glucosa en ayunas (stab.glu), el colesterol total (chol) y la tensión arterial sistólica (bp.1s). Vamos a probar este enfoque:

```
# creamos el modelo sólo con las variables que step() nos dice
predictores <- c("stab.glu","chol", "bp.1s")

formula_str <- paste("diabetes","~", paste(predictores, collapse=" + "))
formula <- as.formula(formula_str)

cat("La formula que utilizaremos es: \n", formula_str)
```

```
## La formula que utilizaremos es:
##  diabetes ~ stab.glu + chol + bp.1s
```

```
modelo_logistico <- glm(formula, data=training_set, family = "binomial")
print(summary(modelo_logistico))
```

```
##
## Call:
## glm(formula = formula, family = "binomial", data = training_set)
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -12.359485   2.224392  -5.556 2.75e-08 ***
## stab.glu      0.044350   0.007097   6.249 4.12e-10 ***
## chol          0.013570   0.006550   2.072   0.0383 *
## bp.1s         0.017722   0.009040   1.960   0.0500 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 232.12  on 256  degrees of freedom
## Residual deviance: 111.06  on 253  degrees of freedom
## AIC: 119.06
##
## Number of Fisher Scoring iterations: 6
```

Hacemos la matriz de confusión:

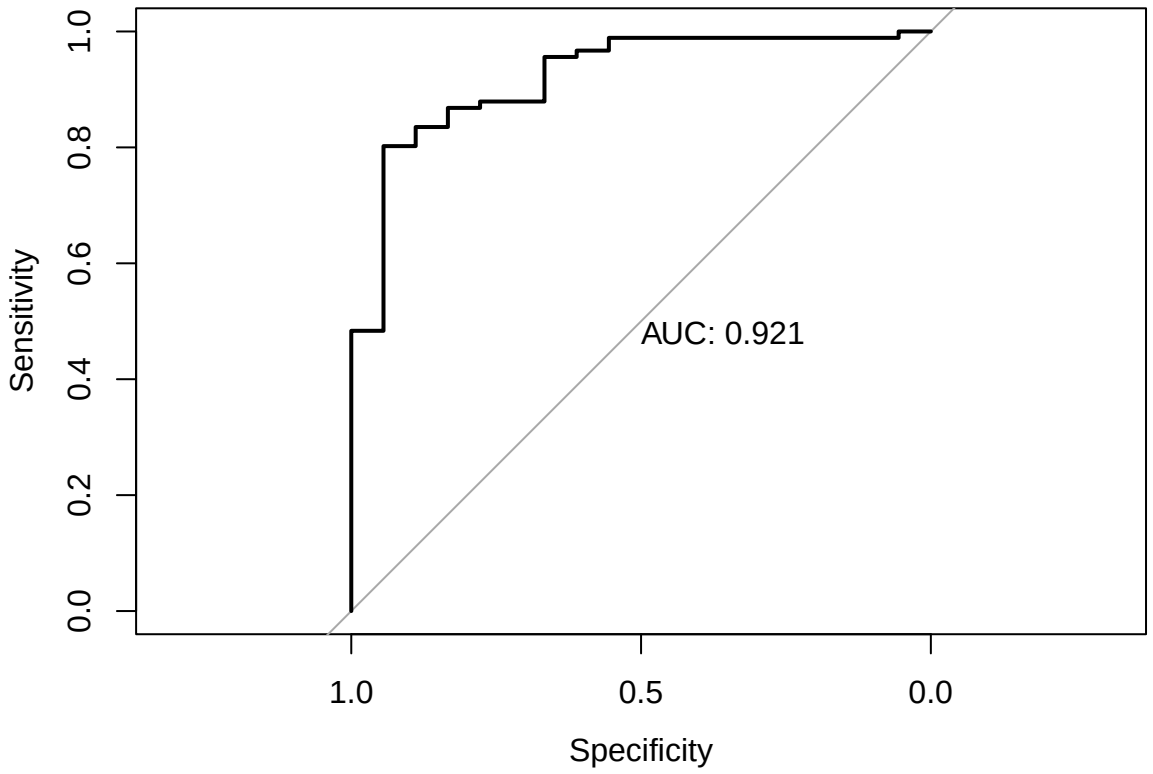
```
prob_test <- predict(modelo_logistico, newdata = test_set, type="response")

clase_positiva <- levels(training_set$diabetes)[2] # diabéticos
clase_negativa <- levels(training_set$diabetes)[1] # no diabéticos
```

```
clase_predicha <- factor(
                    ifelse(prob_test > 0.5 , clase_positiva, clase_negativa),
                    levels = levels(test_set$diabetes))

matriz_confusion <- confusionMatrix(data = clase_predicha,
                                    reference = test_set$diabetes,
                                    positive = clase_positiva)
print(matriz_confusion)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction NO SI
##         NO 90  9
##         SI  1  9
##
##                Accuracy : 0.9083
##                  95% CI : (0.8377, 0.9551)
##     No Information Rate : 0.8349
##     P-Value [Acc > NIR] : 0.02085
##
##                   Kappa : 0.5951
##
##  Mcnemar's Test P-Value : 0.02686
##
##             Sensitivity : 0.50000
##             Specificity : 0.98901
##          Pos Pred Value : 0.90000
##          Neg Pred Value : 0.90909
##              Prevalence : 0.16514
##          Detection Rate : 0.08257
##    Detection Prevalence : 0.09174
##       Balanced Accuracy : 0.74451
##
##        'Positive' Class : SI
##
```

Vemos que la sensibilidad, la precisión y la especificidad no han variado significativamente. Vamos a ver qué pasa con la curva ROC:

```
# hacemos el test roc
roc_test <- roc(response = test_set$diabetes,
                predictor = prob_test,
                levels = rev(levels(test_set$diabetes)), direction = ">")

# representamos la curva
plot(roc_test, print.auc = TRUE)
```




El valor del AUC es muy similar al anterior. En consecuencia, parece que no podemos mejorar nuestro modelo empleando las variables que tenemos. Lo único que podemos hacer es modificar el umbral clasificatorio para mejorar la sensibilidad del modelo. Vamos a ser muy estrictos: escogeremos un umbral de 0.1 para potenciar la clasificación correcta de diabéticos a costa de la de no diabéticos.

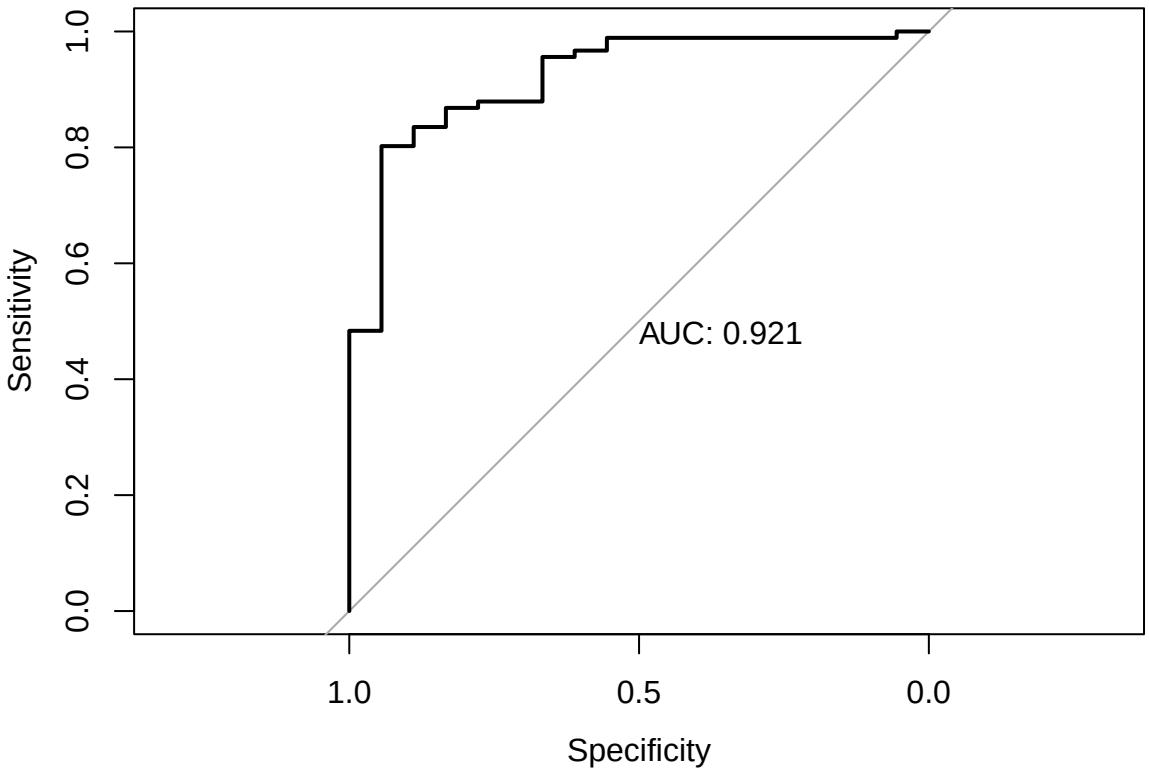
```
prob_test <- predict(modelo_logistico, newdata = test_set, type="response")

# definimos los dos grupos posibles
clase_positiva <- levels(training_set$diabetes)[2] # diabéticos
clase_negativa <- levels(training_set$diabetes)[1] # no diabéticos

# cambiamos el umbral de clasificación
clase_predicha <- factor(
                    ifelse(prob_test > 0.1 , clase_positiva, clase_negativa),
                    levels = levels(test_set$diabetes))

# generamos la matriz d econfusión
matriz_confusion <- confusionMatrix(data = clase_predicha,
                                    reference = test_set$diabetes,
                                    positive = clase_positiva)
print(matriz_confusion)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction NO SI
##         NO 75  2
##         SI 16 16
##
```

```
##                Accuracy : 0.8349
##                  95% CI : (0.7516, 0.8991)
##     No Information Rate : 0.8349
##     P-Value [Acc > NIR] : 0.562499
##
##                   Kappa : 0.5435
##
##  Mcnemar's Test P-Value : 0.002183
##
##             Sensitivity : 0.8889
##             Specificity : 0.8242
##          Pos Pred Value : 0.5000
##          Neg Pred Value : 0.9740
##              Prevalence : 0.1651
##          Detection Rate : 0.1468
##    Detection Prevalence : 0.2936
##       Balanced Accuracy : 0.8565
##
##        'Positive' Class : SI
##
```

Comprobamos que el AUC del modelo no varía al cambiar el umbral:

```r
# hacemos el test roc
roc_test <- roc(response = test_set$diabetes,
                predictor = prob_test,
                levels = rev(levels(test_set$diabetes)), direction = ">")

# representamos la curva
plot(roc_test, print.auc = TRUE)
```




Hemos mejorado la capacidad del modelo para clasificar diabéticos, que es nuestro principal interés. Un mejor modelo, que maximizara la sensibilidad y la especificidad, requeriría emplear otras variables o contar con un mayor número de observaciones.

# 5 Sección 5. Visualización

Para facilitar la visualización de los datos, así como de las herramientas estadísticas empleadas, hemos programado una aplicación Shiny. En ella, mostramos los principales análisis realziados en este estudio: resumen de datos, estadística descriptiva y estadística predictiva. Podemos visualizar las características de los datos y realizar diferentes representaciones gráficas escogiendo las variables. En la parte correspondiente al modelo de regresión logística, podemos seleccionar el tamaño de los conjuntos de entrenamiento y de prueba y las variables empleadas para la predicción, así como representar la curva ROC y ver su principal parámetro: el área bajo la curva (AUC), indicador de la capacidad predictiva del modelo.

## 5.1 Estructura aplicación

Se ha planteado la aplicación mediante los siguientes elementos:

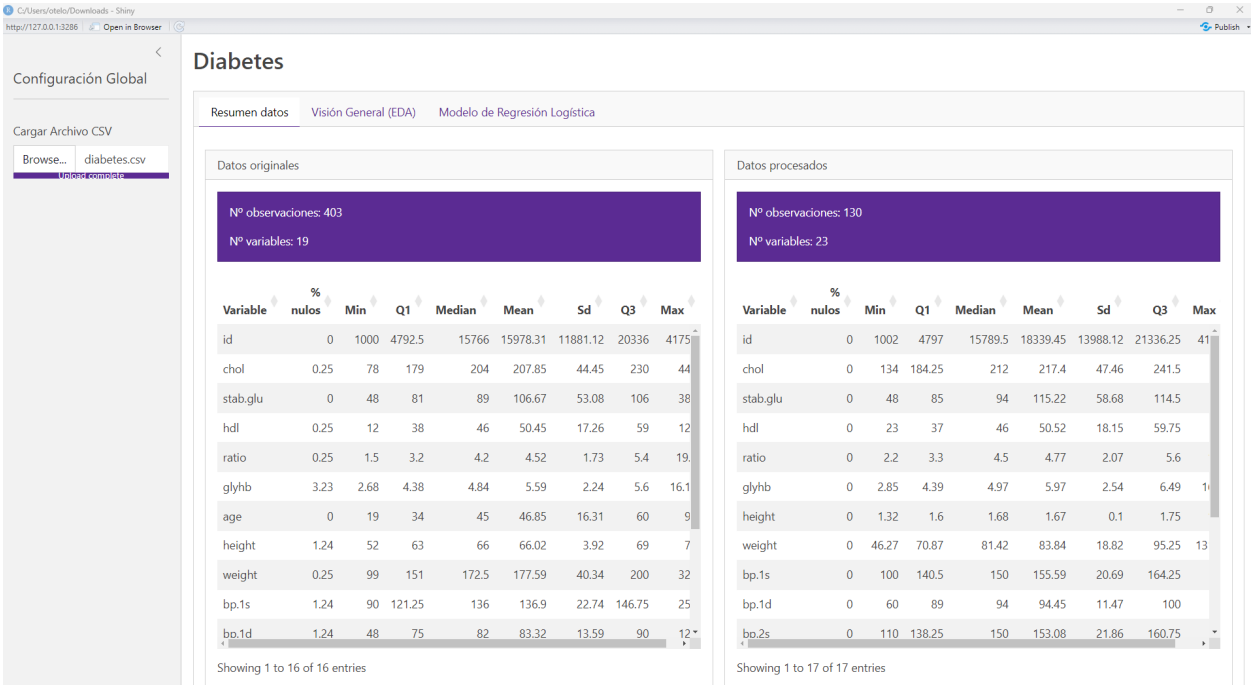
- Panel lateral izquierdo donde se podrá cargar el archivo que se quiere analizar.
- Pestaña de resumen de datos. Donde podremos ver un resumen estadístico de los datos cargados inicialmente y compararlos con los procesados finalmente ( que son los que se utilizarán en el resto de las secciones)
- Pestaña de visión general y análisis exploratorio de datos (EDA). Donde tendremos disponemos de multitud de gráficos configurables para analizar los datos
- Pestaña para el modelo de regresión logística. Donde tendremos un modelo de regresión logística para

la predicción del campo “diabetes”

En las siguientes imágenes podemos ver como ha quedado el diseño.

### 5.1.1 Pestaña de resumen de datos

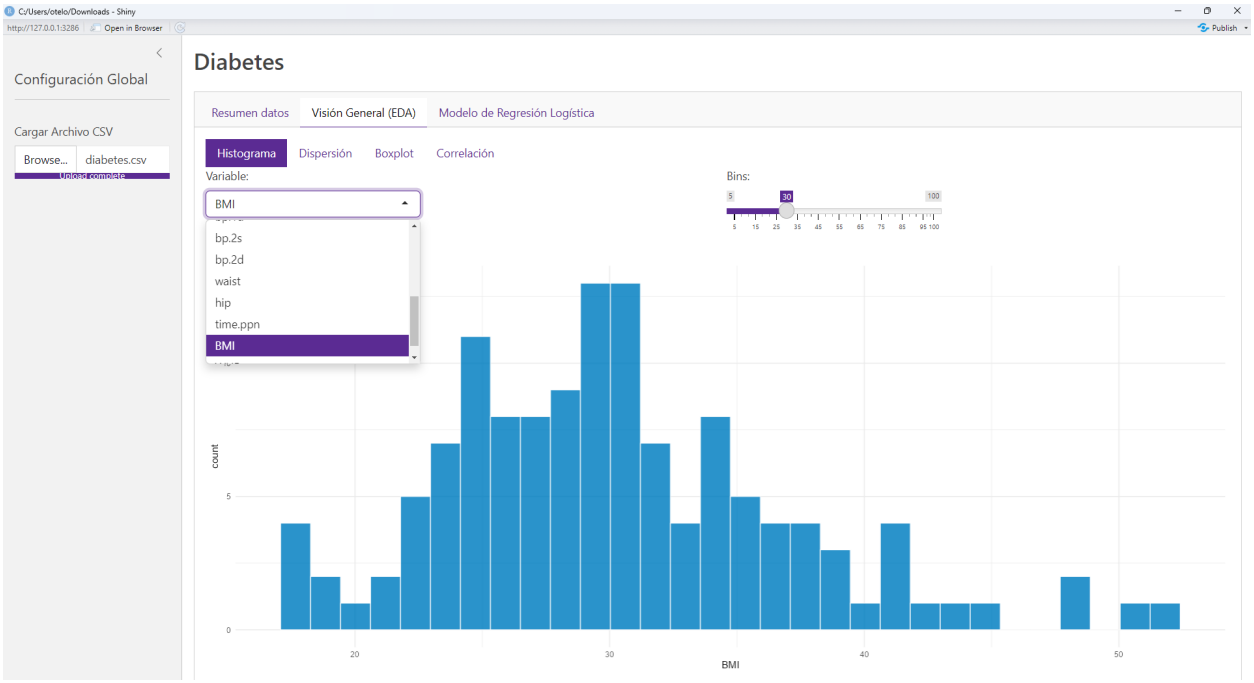
Aquí hacemos un resumen estadístico (así como del porcentaje de nulos) de las variables numericas, por un lado de los datos brutos cargados desde el fichero y a la derecha como han quedado estos datos después de hacer el procesamiento (añadir variables, transformaciones y eliminacion de filas con valores nulos).



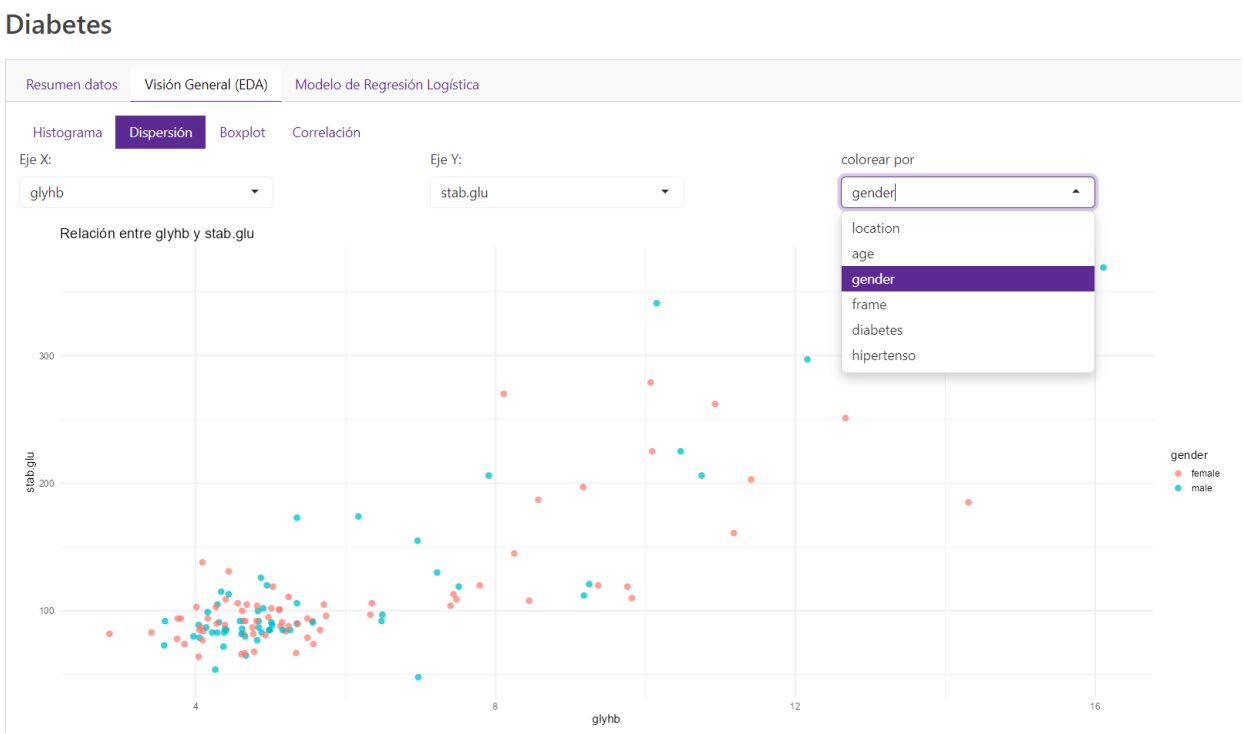

### 5.1.2 Pestaña de visión general y análisis exploratorio de datos (EDA)

En esta sección podemos analizar los datos de forma visual con histogramas, gráficos de dispersión, gráficos de caja y la correlación entre las variables numéricas. En cada gráfico podemos seleccionar qué variables queremos mostrar en cada eje.

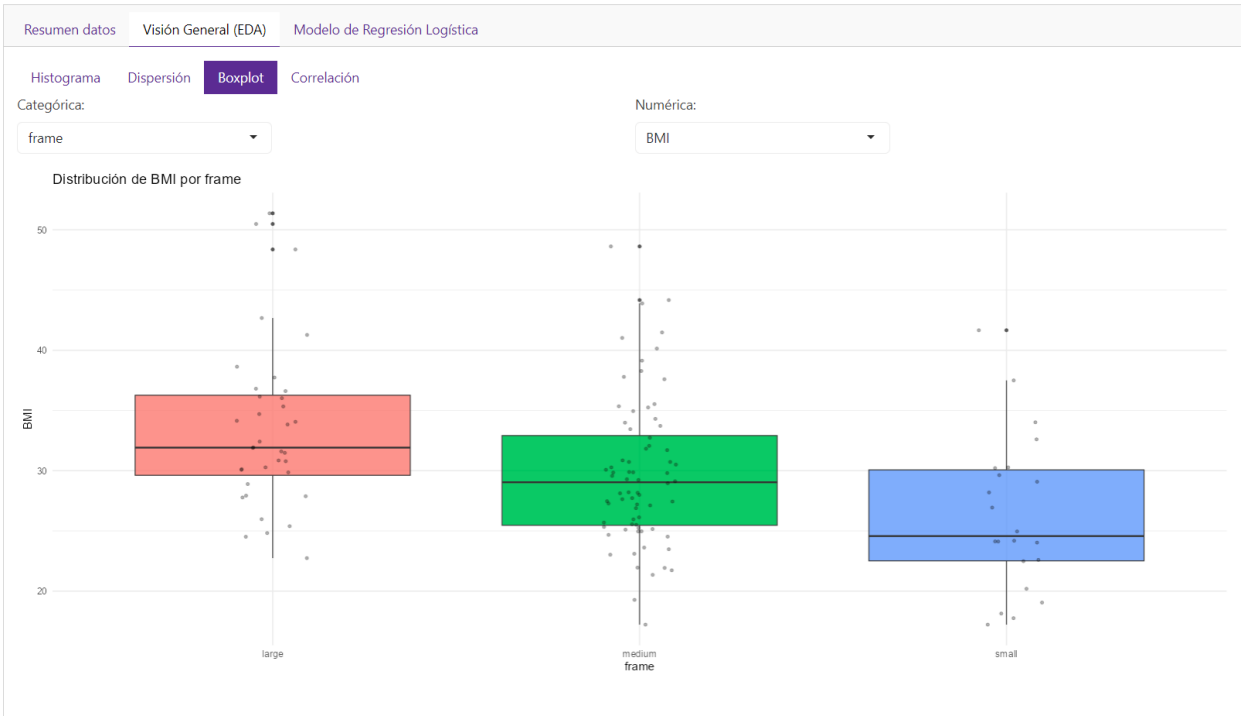
**5.1.2.1 Histograma** Podemos seleccionar tanto la variable cuya frecuencia queremos ver como el número de barras del histograma



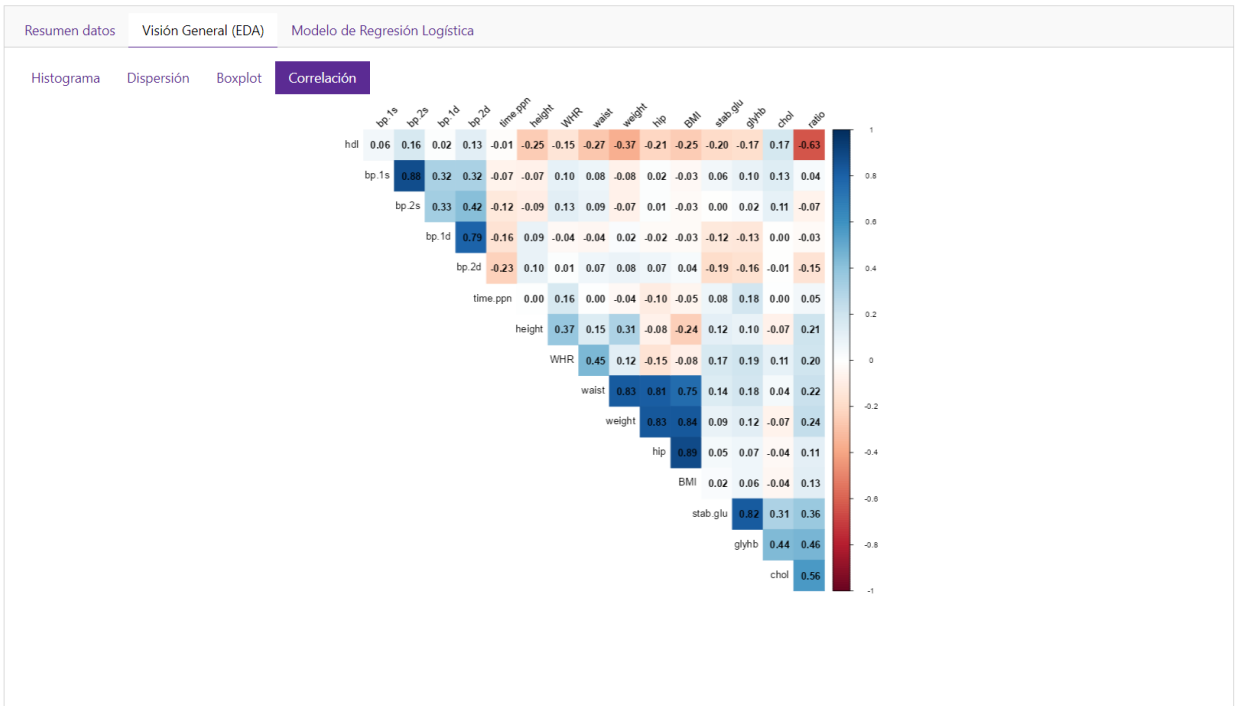

**5.1.2.2 Dispersión** Aparte de poder seleccionar que variable mostramos en cada eje, se ha añadido el poder segmentar por colores según las variables categóricas. De esta manera se puede intentar identificar de forma visual la existencia de algún tipo de cluster.




**5.1.2.3 Boxplot** Jugando con las variables categóricas y numéricas podemos hacer cualquier combinación.




**5.1.2.4 Correlación** Donde se puede ver de forma muy visual la correlación existente entre cada una de las variables numéricas

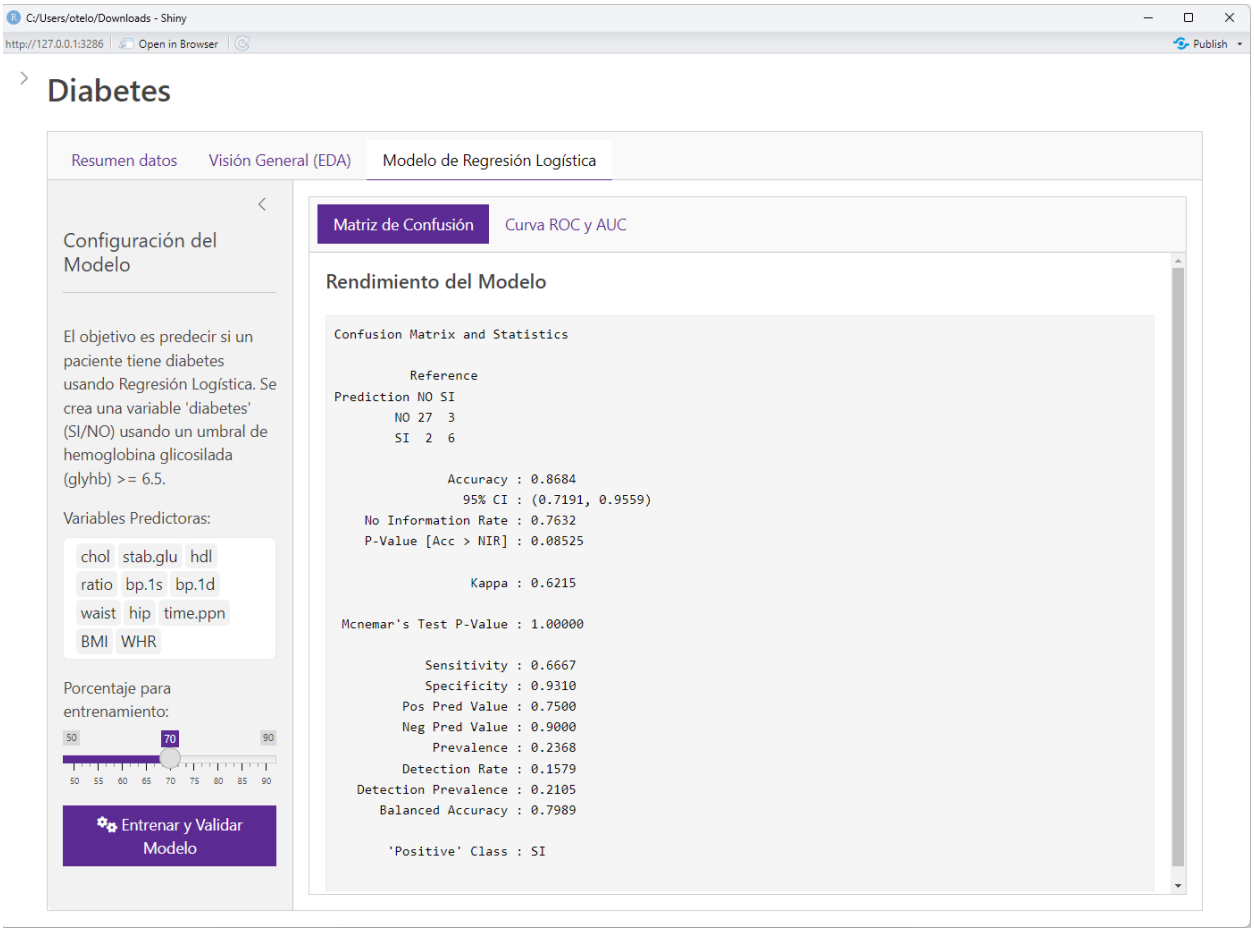


### 5.1.3 Pestaña para el modelo de regresión logística

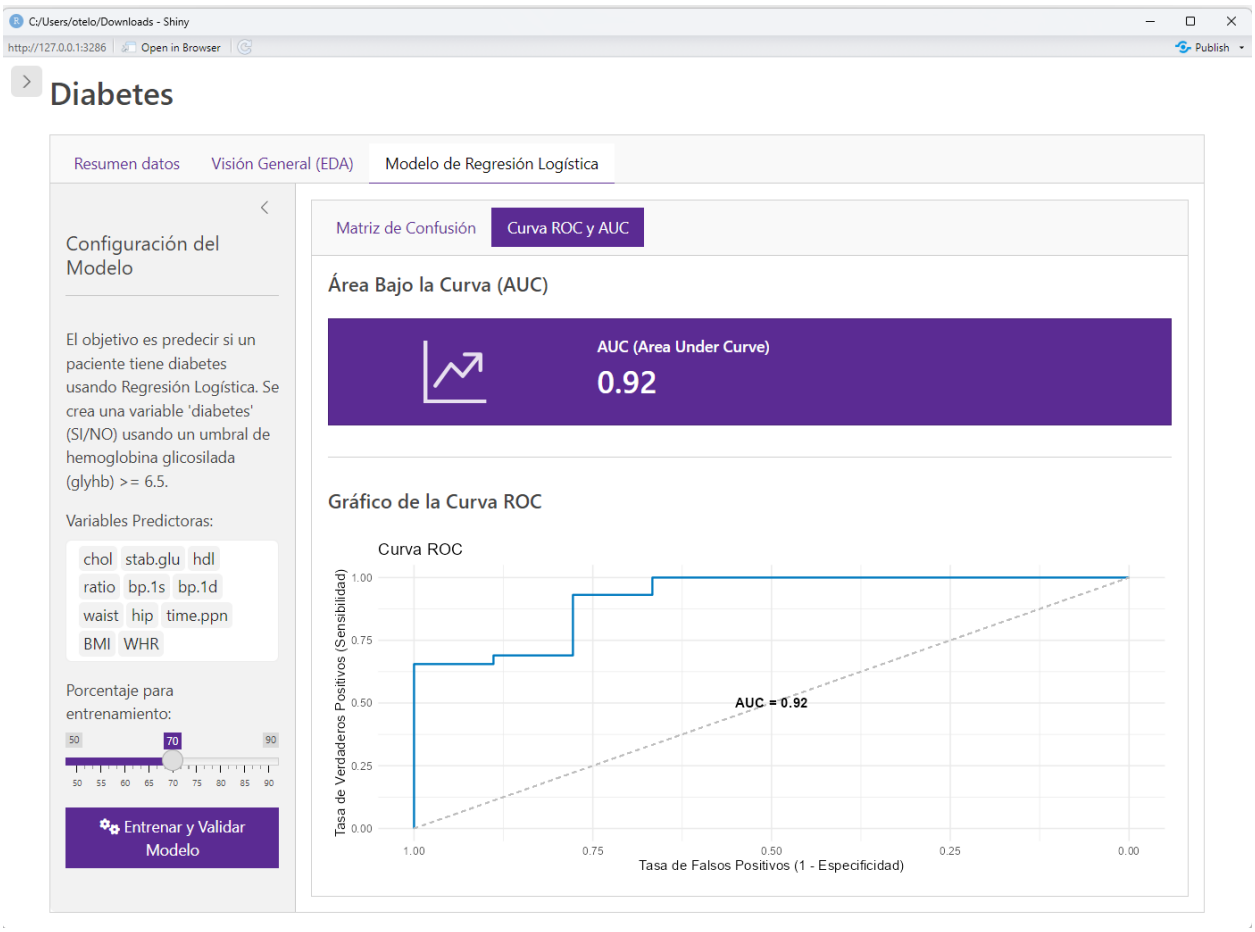
Aquí podemos jugar con un modelo de regresión logística para predecir el campo “diabetes” que hemos creado de forma sintética.

Se pueden seleccionar que variables queremos usar como predictores así como variar el tamaño del conjunto de entrenamiento y test.

En la primera sección podremos ver el rendimiento del modelo y su matriz de confusión




En la segunda sección se muestra la curva ROC de este modelo y su área bajo la curva (AUC)




## 5.2 Explicación del código

### 5.2.1 Importación de librerias

Utilizamos las siguientes cuatro librerias:

- shiny. Para la aplicación
- DT. Para la tabla dinámica
- ggplot2. Para realizar los gráficos
- bslib. Libreria con unos estilos y layouts mas vistosos

### 5.2.2 Definición interfaz gráfica

Aquí describimos como será el layout de la applicación y que objetos se mostrarán en cada sitio. Como hemos comentado tendremos un panel lateral para la carga del archivo y tres pestañas donde mostrar los datos y gráficos.

**5.2.2.1 Panel principal** Usando la libreria bslib, cargamos los estilos del tema *pulse* que hemos obtenido de https://bootswatch.com/pulse/

y jugamos con los estilos con la información obtenida de https://rstudio.github.io/bslib/articles/theming/index.html

Creamos la barra lateral donde incrustamos el selector de archivos y creamos las pestañas centrales con *navset_card_tab* de bslib que podemos encontrar en https://rstudio.github.io/bslib/reference/navset.html

```
ui <- page_sidebar(

# cargamos el tema "pulse" de bootswatch
theme = bs_theme(version = 5, bootswatch = "pulse"),

titlePanel(textOutput("titulo_archivo")),

sidebar = sidebar(
    title = "Configuración Global",
    fileInput("file",
              "Cargar Archivo CSV",
              accept = c("text/csv", ".csv")) ),

navset_card_tab( id = "main_tabs",

    nav_panel("Resumen datos", ...
```

#### 5.2.2.2 Pestaña de resumen de datos

Usando *layout_columns* y *card* ponemos las dos tarjetas en dos columnas con la tabla de resumen de datos originales y procesados respectivamente.

Destacar la utilización de *uiOutput* (en el cliente) y *renderUI* (en el servidor) para generar en componente visual dinámicamente desde el servidor.

```
    nav_panel("Resumen datos",

              layout_columns(
                col_widths = c(6, 6),
                card(
                  full_screen = TRUE,
                  card_header("Datos originales"),
                  card_body(
                    layout_columns(
                      col_widths = c(12, 12),
                      row_heights = c(1,5),
                      fill = FALSE,
                      card(
                        # Le damos estilo con clases de Bootstrap para que se vea como un value_box
                        class = "bg-purple text-white",
                        card_body(
                          uiOutput("resumen_datos_originales")
                        )
                      ),
                      DTOutput("tabla")
                    )
                  )
                ),
                card(
```

```
                full_screen = TRUE,
                card_header("Datos procesados"),
                card_body(
                  layout_columns(
                    col_widths = c(12, 12),
                    row_heights = c(1,5),
                    fill = FALSE,
                    card(
                      # Le damos estilo con clases de Bootstrap para
                      # que se vea como un value_box
                      class = "bg-purple text-white",
                      card_body(
                        uiOutput("resumen_datos_procesados")
                      )
                    ),
                    DTOutput("tablaProcesada")
                  )
                )
              )
            )
    ),
```

**5.2.2.3 Pestaña de visión general y análisis exploratorio de datos (EDA)** En este panel se encuentran todos los gráficos que hemos dividido en secciones pero en lugar de pestañas lo hacemos con botones (usando *navset_pill* ). Todas las secciones tienen la misma estructura:

- uiOutput. Para generar dinámicamente desde el servidor los selectores de variables
- plotOutput. Para presentar el gráfico

```
nav_panel("Visión General (EDA)",
              navset_pill(
                nav_panel("Histograma",
                          uiOutput("hist_selector_ui"),
                          plotOutput("histograma", height = "600px")
                         ),
                nav_panel("Dispersión",
                          uiOutput("scatter_selector_ui"),
                          plotOutput("scatter_plot", height = "600px")
                         ),
                nav_panel("Boxplot",
                          uiOutput("box_selector_ui"),
                          plotOutput("boxplot", height = "600px")
                         ),
                nav_panel("Correlación",
                          plotOutput("corr_plot", height = "600px")
                )
              )
    ),
```

**5.2.2.4 Pestaña para el modelo de regresión logística** Esta sección consta de un panel lateral izquierdo donde se interactua con los parametros del modelo de regresión y un panel central donde se visualizan los resultados.

Los únicos elementos nuevos que no habiamos usado antes son:

- actionButton : Botón para lanzar la ejecución del modelo
- value_box : Tarjeta para mostrar un valor añadiendo un icono

```
nav_panel("Modelo de Regresión Logística",
      layout_sidebar(
        sidebar = sidebar(
          title = "Configuración del Modelo",
          p("El objetivo es predecir si un paciente tiene diabetes
             usando Regresión Logística. Se crea una variable 'diabetes' (SI/NO)
             usando un umbral de hemoglobina glicosilada (glyhb) >= 6.5."),

          uiOutput("model_predictors_ui"),

          sliderInput("train_split",
                      "Porcentaje para entrenamiento:",
                      min = 50, max = 90, value = 70, step = 5),

          actionButton("train_model_btn",
                       "Entrenar y Validar Modelo",
                       class = "btn-primary",
                       icon = icon("cogs"))
        ),

              navset_card_pill(
                nav_panel("Matriz de Confusión",
                          card_body(
                            h5("Rendimiento del Modelo"),
                            verbatimTextOutput("confusion_matrix_print")
                          )
                ),
                nav_panel("Curva ROC y AUC",
                          card_body(
                            h5("Área Bajo la Curva (AUC)"),
                            # Usaremos un value_box para destacar el valor del AUC
                            value_box(
                              title = "AUC (Area Under Curve)",
                              value = textOutput("auc_value"),
                              showcase = bsicons::bs_icon("graph-up-arrow"),
                              theme = "primary"
                            ),
                            hr(),
                            h5("Gráfico de la Curva ROC"),
                            plotOutput("roc_curve_plot")
                          )
                )
              )
            )
    )
  )
```

### 5.2.3 Implementación del servidor

En el servidor es donde se ejecuta toda la lógica de la aplicación. Principalmente podemos destacar las siguientes secciones:

**5.2.3.1 Carga de datos** Es la parte que se encarga de leer el fichero que se ha seleccionado mediante el componete gráfico *fileInput* el cual proporciona el nombre del fichero en la variable *file* que usamos para leerlo y cambiar el título de la aplicación poniendole el nombre del ficher seleccionado.

```
data <- reactive({
  req(input$file)
  # Los datos usados dependen del archivo seleccionado
  read.csv(input$file$datapath, row.names = NULL)
})

# Definimos título dependiente del archivo
output$titulo_archivo <- renderText({
  if (is.null(input$file)) {
    # Si no se ha cargado ningún archivo
    "Cargue un archivo"
  } else {
    nombre_limpio <- tools::file_path_sans_ext(input$file$name)
    nombre_capitalizado <- paste0(toupper(substring(nombre_limpio, 1, 1)),
      substring(nombre_limpio, 2))

    paste(nombre_capitalizado)
  }
})
```

**5.2.3.2 Resumen estadisticos variables numéricas** Aquí tenemos las siguientes dos partes:

- RenderUI. Donde creamos dinánicamente el HTML con el resumen de número de observaciones y número de variables
- RenderDT: Donde generamos la tabla resumen con el apoyo de las siguientes funciones que hemos creado:
  - procesaDatosOriginales: donde le pasamos por parámetro los datos originales y devuelve un dataframe con los datos transformados (factorización, cambio de unidades, variables sintéticas y eliminación de nulos)
  - estadisticas_numericas: dado un dataframe devuelve otro con los estadísticos de las variables numéricas

```
## ----------- Datos procesados ----------------
# Resumen
output$resumen_datos_procesados <- renderUI({
  req(data())

  # procesamos los datos
  df <- procesaDatosOriginales(data())

  tagList(
    # Usamos tags$p() para crear párrafos. El texto es reactivo.
```

```
      tags$p(paste("Nº observaciones:", nrow(df))),
      tags$p(paste("Nº variables:",  ncol(df))),
    )
  })
  # Tabla estadísticos
  output$tablaProcesada <- renderDT({
    req(data())
    # procesamos los datos
    df <- data()
    datos <- procesaDatosOriginales(df)

    # creamos las estadísticas numéricas
    df <- estadisticas_numericas(datos)

    # En caso de que tengamos datos montamos la tabla
    datatable(
      df,
      options = list(
        paging = FALSE,
        scrollY = "450px",
        scrollX = TRUE,
        searching = FALSE # Habilitar búsqueda
      ),
      rownames = FALSE # No mostrar nombres de fila
    )

  })
```

**5.2.3.3 Visión general y análisis exploratorio de datos (EDA)** Aquí tenemos el código para la generación de todos los gráficos. Todos ellos tienen la misma estructura por lo que para no extenderse demasiado pondremos como ejemplo el gráfico de dispersión

La estructura que se repite tiene dos partes:

**5.2.3.3.1 Generación dinámica de los selectores de variables** Aquí procesamos los datos originales, selccionamos cuales son las variables númericas y categóricas y creamos los componentes visuales para cada caso

```
# --------- Scaterplot -----------------------------
   output$scatter_selector_ui <- renderUI({
     req(data())
     # procesamos los datos
     datos <- procesaDatosOriginales(data())

     variables_numericas <- names(datos[,sapply(datos, is.numeric)])
     variables_categoricas <- names(datos[,sapply(datos, function(x) is.factor(x) ||
        is.character(x))])

     variables_numericas <- setdiff(variables_numericas, c("id"))
     color_choices <-  c("Ninguna" = "",variables_categoricas)

     tagList(
```

```
      fluidRow(
        column(4,
               selectInput("var_scatter_x",
                           "Eje X:",
                           choices = variables_numericas,
                           selected = intersect("imc",
                                                variables_numericas)[1])
               ),
        column(4,
               selectInput("var_scatter_y",
                           "Eje Y:",
                           choices = variables_numericas,
                           selected = intersect("hemoglobina_glicosilada",
                                                variables_numericas)[1])),
        column(4,
               selectInput("color_scatter",
                           "colorear por",
                           choices = color_choices,
                           selected = "Ninguna"
               )
        )
      )
    )
  })
```

**5.2.3.3.2 Generación del gráfico** Aquí creamos el scatterplot con ggplot. Aquí solo destacar las siguientes dos cosas:

- Uso de *aes_string* para que las variables de los ejes sean dinámicas en función de lo que se haya elegido anteriormente
- Uso de *geom_point* para colorear los puntos en caso de ser necesario en función de la variable categórica escogida

```
output$scatter_plot <- renderPlot({

   req(data(), input$var_scatter_x, input$var_scatter_y);

   datos <- procesaDatosOriginales(data())

   p <- ggplot(datos,
               aes_string(x = input$var_scatter_x, y = input$var_scatter_y));

   if (input$color_scatter != "" ) {
     p <- p + geom_point(aes_string(color = input$color_scatter), alpha = 0.7, size = 3)
   }
   else {
     p <- p + geom_point(alpha = 0.7, size = 3, color = "#007bc2")
   }

   p + labs(title = paste("Relación entre",
                          input$var_scatter_x, "y",
                          input$var_scatter_y)) +
     theme_minimal(base_size = 14)
```

```
})
```

**5.2.3.4 Modelo de regresión logística** Para llevar a cabo la regresión logística diferenciamos tres partes:

**5.2.3.4.1 Selector variables predictoras** Aquí simplemente creamos de forma dinámica el selector (*selectInput*) de los predictores del modelo

```
output$model_predictors_ui <- renderUI({

  req(data())
  # procesamos los datos
  datos <- procesaDatosOriginales(data())

  variables_numericas <- names(datos[,sapply(datos, is.numeric)])
  variables_categoricas <- names(datos[
    sapply(datos,
           function(x) is.factor(x) ||
             is.character(x))
  ])


  predictores_sugeridos <- setdiff(variables_numericas,
                                   c("id", "glyhb", "height", "weight"))

  selectInput("model_predictors", "Variables Predictoras:",
              choices = c(variables_numericas, variables_categoricas),
              selected = predictores_sugeridos,
              multiple = TRUE)
})
```

**5.2.3.4.2 Ejecución del modelo** Aquí utilizamos *EventReactive* para detectar cuando se pulsa el boton para correr el modelo que es el mismo que hemos utilizado anterirormente en la práctica.

Calculamos la matriz de confusión y la curva ROC y las devolvemos

```
model_results <- eventReactive(input$train_model_btn, {
       req(req(data()), input$model_predictors)

       datos <- procesaDatosOriginales(data())

       columnas_modelo <- c("diabetes", input$model_predictors)
       df <- na.omit(datos[, columnas_modelo])

       set.seed(666)

       train_index <- createDataPartition(
         datos$diabetes,
         p = 0.7,
         list = FALSE,
         times = 1
```

```
      )


      training_set <- datos[train_index,]
      test_set <- datos[-train_index,]

     #cat(input$model_predictors)

      formula_str <- paste("diabetes","~", paste(input$model_predictors, collapse=" + "))
      formula_str <- gsub("NULL","", formula_str, ignore.case = TRUE)
      formula <- as.formula(formula_str)

      #cat("La formula que utilizaremos es: \n", formula_str)

      modelo_logistico <- glm(formula, data=training_set, family = "binomial")


      prob_test <- predict(modelo_logistico, newdata = test_set, type="response")

      clase_positiva <- levels(training_set$diabetes)[2] # diabéticos
      clase_negativa <- levels(training_set$diabetes)[1] # no diabéticos

      clase_predicha <- factor(
        ifelse(prob_test > 0.5 , clase_positiva, clase_negativa),
        levels = levels(test_set$diabetes))

      matriz_confusion <- confusionMatrix(data = clase_predicha,
                                          reference = test_set$diabetes,
                                          positive = clase_positiva)

      roc_test <- roc(response = test_set$diabetes,
                      predictor = prob_test,
                      levels = rev(levels(test_set$diabetes)))

      return(list(confusionMatrix = matriz_confusion, roc = roc_test))
  })
```

**5.2.3.4.3 Visualización de resultados** En función de los datos devueltos por el modelo, exponemos de forma reactiva los resultados

Para la matriz de confusión y el valor AUC tenemos

```
  output$confusion_matrix_print <- renderPrint({
     req(model_results())
      model_results()$confusionMatrix
    })

   output$auc_value <- renderText({
     req(model_results())
     # Extraemos el AUC del objeto roc y lo formateamos
     auc_val <- auc(model_results()$roc)
     paste0(round(auc_val, 3))
```

```
  })
```

Para crear la curva ROC nos valemos de la función *ggroc* cuya documentación podemos ver en https://www.rdocumentation.org/packages/pROC/versions/1.18.5/topics/ggroc.roc

```
 output$roc_curve_plot <- renderPlot({
     req(model_results())
     roc_obj <- model_results()$roc

     # Graficar la curva ROC con ggplot2 para un mejor estilo
     ggroc(roc_obj, color = "#007bc2", size = 1) +
       geom_segment(aes(x = 1, xend = 0, y = 0, yend = 1),
                    color="grey",
                    linetype="dashed") +
       ggtitle("Curva ROC") +
       labs(x = "Tasa de Falsos Positivos (1 - Especificidad)",
            y = "Tasa de Verdaderos Positivos (Sensibilidad)") +
       annotate("text", x = .5, y = .5,
                label = paste("AUC =", round(auc(roc_obj), 3)),
                size = 5, fontface = "bold") +
       theme_minimal(base_size = 14)
   })
```

## 5.3 Código completo aplicación

```
library(shiny)
library(bslib)
library(ggplot2)
library(DT)

ui <- page_sidebar(
  # cargamos el tema "pulse" de bootswatch
  theme = bs_theme(version = 5, bootswatch = "pulse"),

  titlePanel(textOutput("titulo_archivo")),

  sidebar = sidebar(
    title = "Configuración Global",
    fileInput("file", "Cargar Archivo CSV",
              accept = c("text/csv", ".csv"))
  ),

  navset_card_tab(
    id = "main_tabs",

    nav_panel("Resumen datos",

              layout_columns(
                col_widths = c(6, 6),
                card(
                  full_screen = TRUE,
```

```
                card_header("Datos originales"),
                card_body(
                  layout_columns(
                    col_widths = c(12, 12),
                    row_heights = c(1,5),
                    fill = FALSE,
                    card(
                      # Le damos estilo con clases de Bootstrap para que se vea como un
                        ↪  value_box
                      class = "bg-purple text-white",
                      card_body(
                        uiOutput("resumen_datos_originales")
                      )
                    ),
                    DTOutput("tabla")
                  )
                )
              ),
              card(
                full_screen = TRUE,
                card_header("Datos procesados"),
                card_body(
                  layout_columns(
                    col_widths = c(12, 12),
                    row_heights = c(1,5),
                    fill = FALSE,
                    card(
                      # Le damos estilo con clases de Bootstrap para
                      # que se vea como un value_box
                      class = "bg-purple text-white",
                      card_body(
                        uiOutput("resumen_datos_procesados")
                      )
                    ),
                    DTOutput("tablaProcesada")
                  )
                )
              )
            )
  ),

  nav_panel("Visión General (EDA)",
            navset_pill(
              nav_panel("Histograma",
                        uiOutput("hist_selector_ui"),
                        plotOutput("histograma", height = "600px")
                       ),
              nav_panel("Dispersión",
                        uiOutput("scatter_selector_ui"),
                        plotOutput("scatter_plot", height = "600px")
                       ),
              nav_panel("Boxplot",
                        uiOutput("box_selector_ui"),
                        plotOutput("boxplot", height = "600px")
```

```
                    ),
                nav_panel("Correlación",
                          plotOutput("corr_plot", height = "600px")
                )
              )
    ),

    nav_panel("Modelo de Regresión Logística",
      layout_sidebar(
        sidebar = sidebar(
          title = "Configuración del Modelo",
          p("El objetivo es predecir si un paciente tiene diabetes
             usando Regresión Logística. Se crea una variable 'diabetes' (SI/NO)
             usando un umbral de hemoglobina glicosilada (glyhb) >= 6.5."),

          uiOutput("model_predictors_ui"),

          sliderInput("train_split",
                      "Porcentaje para entrenamiento:",
                      min = 50, max = 90, value = 70, step = 5),

          actionButton("train_model_btn",
                       "Entrenar y Validar Modelo",
                       class = "btn-primary",
                       icon = icon("cogs"))
        ),

                navset_card_pill(
                  nav_panel("Matriz de Confusión",
                            card_body(
                              h5("Rendimiento del Modelo"),
                              verbatimTextOutput("confusion_matrix_print")
                            )
                  ),
                  nav_panel("Curva ROC y AUC",
                            card_body(
                              h5("Área Bajo la Curva (AUC)"),
                              # Usaremos un value_box para destacar el valor del AUC
                              value_box(
                                title = "AUC (Area Under Curve)",
                                value = textOutput("auc_value"),
                                showcase = bsicons::bs_icon("graph-up-arrow"),
                                theme = "primary"
                              ),
                              hr(),
                              h5("Gráfico de la Curva ROC"),
                              plotOutput("roc_curve_plot")
                            )
                  )
                )
              )
    )
  )
)
```

```r
server <- function(input, output) {

  data <- reactive({
    req(input$file)
    # Los datos usados dependen del archivo seleccionado
    read.csv(input$file$datapath, row.names = NULL)
  })

  # Definimos título dependiente del archivo
  output$titulo_archivo <- renderText({
    if (is.null(input$file)) {
      # Si no se ha cargado ningún archivo
      "Cargue un archivo"
    } else {
      nombre_limpio <- tools::file_path_sans_ext(input$file$name)
      nombre_capitalizado <- paste0(toupper(substring(nombre_limpio, 1, 1)),
    ↪  substring(nombre_limpio, 2))

      paste(nombre_capitalizado)
    }
  })


  ## ------------------ Datos originales ------------------
  # Resumen
  output$resumen_datos_originales <- renderUI({
    req(data())
    tagList(
      # Usamos tags$p() para crear párrafos. El texto es reactivo.
      tags$p(paste("Nº observaciones:", nrow(data()))),
      tags$p(paste("Nº variables:",  ncol(data()))),
    )
  })
  # Tabla estadísticos
  output$tabla <- renderDT({
    req(data())

    df <- estadisticas_numericas(data())

    datatable(
      df,
      options = list(
        paging = FALSE,
        scrollY = "450px",
        scrollX = TRUE,
        searching = FALSE # Deshabilitar búsqueda
      ),
      rownames = FALSE # No mostrar nombres de fila
    )
  })

  ## ----------- Datos procesados ----------------
  # Resumen
```

```
output$resumen_datos_procesados <- renderUI({
  req(data())

  # procesamos los datos
  df <- procesaDatosOriginales(data())

  tagList(
    # Usamos tags$p() para crear párrafos. El texto es reactivo.
    tags$p(paste("Nº observaciones:", nrow(df))),
    tags$p(paste("Nº variables:",  ncol(df))),
  )
})
# Tabla estadísticos
output$tablaProcesada <- renderDT({
  req(data())
  # procesamos los datos
  df <- data()
  datos <- procesaDatosOriginales(df)

  # creamos las estadísticas numéricas
  df <- estadisticas_numericas(datos)

  # En caso de que tengamos datos montamos la tabla
  datatable(
    df,
    options = list(
      paging = FALSE,
      scrollY = "450px",
      scrollX = TRUE,
      searching = FALSE # Habilitar búsqueda
    ),
    rownames = FALSE # No mostrar nombres de fila
  )

})

# ----------- Histograma -------------------------------
output$hist_selector_ui <- renderUI({

  req(data())
  # procesamos los datos
  df <- data()
  datos <- procesaDatosOriginales(df)

  variables_numericas <- names(datos[,sapply(datos, is.numeric)])
  variables_numericas <- setdiff(variables_numericas, c("id"))

  tagList(
    fluidRow(
       column(6,
        selectInput("var_hist",
                    "Variable:",
                    choices = variables_numericas,
                    selected = intersect("BMI", variables_numericas)[1])
```

```
        ),
        column(6,
         sliderInput("bins_hist",
                 "Bins:",
                 min = 5,
                 max = 100,
                  value = 30)
        )
     )
   )
})

output$histograma <- renderPlot({

   req(data(), input$var_hist)
   # procesamos los datos
   df <- data()
   datos <- procesaDatosOriginales(df)

   ggplot(datos,
          aes_string(x = input$var_hist)) +
     geom_histogram(bins = input$bins_hist,
                    fill = "#007bc2",
                    color = "white",
                    alpha = 0.8) +
     labs(title = paste("Distribución de", input$var_hist)) +
     theme_minimal(base_size = 14)
 })


# --------- Scaterplot -----------------------------
 output$scatter_selector_ui <- renderUI({
   req(data())
   # procesamos los datos
   datos <- procesaDatosOriginales(data())

   variables_numericas <- names(datos[,sapply(datos, is.numeric)])
   variables_categoricas <- names(datos[,sapply(datos, function(x) is.factor(x) ||
↪   is.character(x))])

   variables_numericas <- setdiff(variables_numericas, c("id"))
   color_choices <-  c("Ninguna" = "",variables_categoricas)

   tagList(
     fluidRow(
       column(4,
              selectInput("var_scatter_x",
                          "Eje X:",
                          choices = variables_numericas,
                          selected = intersect("imc",
                                               variables_numericas)[1])
              ),
       column(4,
              selectInput("var_scatter_y",
```

```
                        "Eje Y:",
                        choices = variables_numericas,
                        selected = intersect("hemoglobina_glicosilada",
                                             variables_numericas)[1])),
        column(4,
               selectInput("color_scatter",
                        "colorear por",
                        choices = color_choices,
                        selected = "Ninguna"
               )
        )
      )
    )
  })

  output$scatter_plot <- renderPlot({

    req(data(), input$var_scatter_x, input$var_scatter_y);

    datos <- procesaDatosOriginales(data())

    p <- ggplot(datos,
                aes_string(x = input$var_scatter_x, y = input$var_scatter_y));

    if (input$color_scatter != "" ) {
      p <- p + geom_point(aes_string(color = input$color_scatter), alpha = 0.7, size =
↪  3)
    }
    else {
      p <- p + geom_point(alpha = 0.7, size = 3, color = "#007bc2")
    }

    p + labs(title = paste("Relación entre",
                        input$var_scatter_x, "y",
                        input$var_scatter_y)) +
      theme_minimal(base_size = 14)
  })


# ------------------ Boxplot ----------------------------------
  output$box_selector_ui <- renderUI({
    req(data())
    # procesamos los datos
    datos <- procesaDatosOriginales(data())

    variables_numericas <- names(datos[,sapply(datos, is.numeric)])
    variables_categoricas <- names(datos[
                                    sapply(datos,
                                         function(x) is.factor(x) ||
                                                     is.character(x))
                                    ])
    variables_numericas <- setdiff(variables_numericas, c("id"))

    tagList(
```

```
    fluidRow(
      column(6,
         selectInput("var_box_cat", "Categórica:",
                     choices = variables_categoricas,
                     selected = intersect("frame",
                                          variables_categoricas)[1])
      ),
      column(6,
        selectInput("var_box_num", "Numérica:",
                    choices = variables_numericas,
                    selected = intersect("BMI",
                                         variables_numericas)[1])
      )
    )
  )
})

output$boxplot <- renderPlot({

  req(data(), input$var_box_cat, input$var_box_num);

  datos <- procesaDatosOriginales(data())

  ggplot(datos,
         aes_string(x = input$var_box_cat, y = input$var_box_num)) +
    geom_boxplot(
      aes_string(fill = input$var_box_cat),
      alpha = 0.8,
      show.legend = FALSE) +
    geom_jitter(width = 0.1, alpha = 0.3) +
    labs(title = paste("Distribución de",
                       input$var_box_num, "por",
                       input$var_box_cat)) +
    theme_minimal(base_size = 14)
})


# ----------------- Correlación -----------------------------
output$corr_plot <- renderPlot({
  req(data());

  datos <- procesaDatosOriginales(data())

  variables_numericas <- names(datos[,sapply(datos, is.numeric)])
  variables_numericas <- setdiff(variables_numericas, c("id"))

  df_numeric <- na.omit(datos[, variables_numericas]);
  cor_matrix <- cor(df_numeric);
  corrplot(cor_matrix,
           method = "color",
           type = "upper",
           order = "hclust",
           addCoef.col = "black",
           tl.col = "black",
```

```
            tl.srt = 45,
            diag = FALSE
  )
})



# ------------ Regresión logística -------------------------


output$model_predictors_ui <- renderUI({

  req(data())
  # procesamos los datos
  datos <- procesaDatosOriginales(data())

  variables_numericas <- names(datos[,sapply(datos, is.numeric)])
  variables_categoricas <- names(datos[
    sapply(datos,
           function(x) is.factor(x) ||
             is.character(x))
  ])

  predictores_sugeridos <- setdiff(variables_numericas,
                                   c("id", "glyhb", "height", "weight"))

  selectInput("model_predictors", "Variables Predictoras:",
              choices = c(variables_numericas, variables_categoricas),
              selected = predictores_sugeridos,
              multiple = TRUE)
})

model_results <- eventReactive(input$train_model_btn, {
     req(req(data()), input$model_predictors)

     datos <- procesaDatosOriginales(data())

     columnas_modelo <- c("diabetes", input$model_predictors)
     df <- na.omit(datos[, columnas_modelo])

     set.seed(666)

     train_index <- createDataPartition(
       datos$diabetes,
       p = 0.7,
       list = FALSE,
       times = 1
     )


     training_set <- datos[train_index,]
     test_set <- datos[-train_index,]

    #cat(input$model_predictors)
```

```
      formula_str <- paste("diabetes","~", paste(input$model_predictors, collapse=" +
↪   "))
      formula_str <- gsub("NULL","", formula_str, ignore.case = TRUE)
      formula <- as.formula(formula_str)

      #cat("La formula que utilizaremos es: \n", formula_str)

      modelo_logistico <- glm(formula, data=training_set, family = "binomial")


      prob_test <- predict(modelo_logistico, newdata = test_set, type="response")

      clase_positiva <- levels(training_set$diabetes)[2] # diabéticos
      clase_negativa <- levels(training_set$diabetes)[1] # no diabéticos

      clase_predicha <- factor(
        ifelse(prob_test > 0.5 , clase_positiva, clase_negativa),
        levels = levels(test_set$diabetes))

      matriz_confusion <- confusionMatrix(data = clase_predicha,
                                          reference = test_set$diabetes,
                                          positive = clase_positiva)

      roc_test <- roc(response = test_set$diabetes,
                      predictor = prob_test,
                      levels = rev(levels(test_set$diabetes)))

      return(list(confusionMatrix = matriz_confusion, roc = roc_test))
})



output$confusion_matrix_print <- renderPrint({
  req(model_results())
   model_results()$confusionMatrix
 })

output$auc_value <- renderText({
  req(model_results())
  # Extraemos el AUC del objeto roc y lo formateamos
  auc_val <- auc(model_results()$roc)
  paste0(round(auc_val, 3))
})

output$roc_curve_plot <- renderPlot({
  req(model_results())
  roc_obj <- model_results()$roc

  # Graficar la curva ROC con ggplot2 para un mejor estilo
  ggroc(roc_obj, color = "#007bc2", size = 1) +
    geom_segment(aes(x = 1, xend = 0, y = 0, yend = 1),
                 color="grey",
                 linetype="dashed") +
```

```
    ggtitle("Curva ROC") +
    labs(x = "Tasa de Falsos Positivos (1 - Especificidad)",
         y = "Tasa de Verdaderos Positivos (Sensibilidad)") +
    annotate("text", x = .5, y = .5,
             label = paste("AUC =", round(auc(roc_obj), 3)),
             size = 5, fontface = "bold") +
    theme_minimal(base_size = 14)
})

# --------- Funciones de apoyo -----------------------------
procesaDatosOriginales <- function(data) {

  df_procesado <- data

  # ------- Cambio unidades de variables --------------------
  # de pulgadas a metros
  df_procesado$height <- df_procesado$height * 0.0254
  df_procesado$hip <- df_procesado$hip * 0.0254
  df_procesado$waist <- df_procesado$waist * 0.0254

  # de libras a kilos
  df_procesado$weight <- df_procesado$weight * 0.453592

  # ------- factorización variables -------------------------
  columnas_factorizables <- c("location", "gender", "frame")

  df_procesado[columnas_factorizables] <- lapply(
    df_procesado[columnas_factorizables],
    as.factor)

  # Eliminamos el nivel extra de Frame
  df_procesado <- df_procesado[df_procesado$frame != "", ]  # Filtra filas con factor
↪  diferente de ""
  df_procesado$frame <- droplevels(df_procesado$frame)    # Elimina el nivel vacío de
↪  los factores

  # Factorizamos la edad
  df_procesado$age <- cut(df_procesado$age,   # partimos la variable...
                          breaks = seq(0, 100, by = 10),          # ...en intervalos de
                            ↪  10 años
                          right = FALSE,                # intervalo cerrado a la
                            ↪  izquierda [x,y)
                          include.lowest = TRUE          # incluye el primer valor en el
                            ↪  primer grupo
  )

  # --------- Creamos nuevas variables -------------------

  # indice de masa corporal
  df_procesado$BMI <- df_procesado$weight / df_procesado$height^2

  # ratio cintura cadera (Waist Hip Ratio)
  df_procesado$WHR <- df_procesado$waist / df_procesado$hip
```

```r
  # suponemos que es diabético si glyhb >= 6.5
  df_procesado$diabetes <- cut(df_procesado$glyhb,
                               breaks = c(0,  6.49, Inf),
                               labels = c("NO","SI"),
                               right = TRUE) # right=TRUE => intervalo es (a, b]

  df_procesado$hipertenso <- as.factor(ifelse(
    df_procesado$bp.1s >= 140 &
      df_procesado$bp.1d >= 90,
    "SI",
    "NO")
  )

  # quitamos las columnas que no nos interesan
  diabetes_reducido <- df_procesado %>%
    select(-c(id, bp.2s, bp.2d))  # Elimina solo lo que realmente quieres quitar

  # ahora eliminamos los NAs
  df_procesado <- na.omit(diabetes_reducido)

  return (df_procesado)
}

estadisticos_columna <- function(col) {
  if (is.numeric(col)) {
    # Si es numérica, devolver resumen estadístico básico y el porcentaje de nulos
    # Para hacer el porcentaje basta con hacer la media del numero de nulos (nºnulos /
      ↪  nº valores de la columna)
    # Para los estadisticos eliminamos los valores faltantes con na.rm=TRUE
    return(
      c(
        Nulos = round(mean(is.na(col)) * 100, 2),
        Min = round(min(col, na.rm = TRUE), 2),
        Q1 = round(quantile(col, 0.25, na.rm = TRUE), 2),
        Median = round(median(col, na.rm = TRUE), 2),
        Mean = round(mean(col, na.rm = TRUE), 2),
        Sd = round(sd(col, na.rm = TRUE), 2),
        Q3 = round(quantile(col, 0.75, na.rm = TRUE), 2),
        Max = round(max(col, na.rm = TRUE), 2)
      )
    )
  }
}

estadisticas_numericas <- function(df) {
  # Escogemos solo las variables numéricas
  columnas_numericas <- df[, sapply(df, function(x)
    is.numeric(x))]

  # Aplicamos la funcion de estadísticos para cada variable
  estadisticos <- lapply(columnas_numericas, estadisticos_columna)

  # Convertir lista de vectores a data.frame
  tabla_resumen <- do.call(rbind, estadisticos)
```

```
    # Cramos el dataframe final añadiendo el nombre de las variables como primera columna
    tabla_resumen <- data.frame(Variable = rownames(tabla_resumen),
                                tabla_resumen,
                                row.names = NULL)
    names(tabla_resumen) <- c("Variable","%
      ↪  nulos","Min","Q1","Median","Mean","Sd","Q3","Max")
    return(tabla_resumen)

  }

}

# Creamos la aplicación Shiny
shinyApp(ui = ui, server = server)
```

# 6 Sección 6. Conclusiones

## 6.1 Valoración final

En este estudio hemos realizado un análisis de las variables cuantitativas y cualitativas de un grupo amplio de pacientes (403), de los cuales un 15,6% eran diabéticos y un 23,3 % eran hipertensos. El diagnóstico de diabetes se establece habitualmente en la práctica clínica cuando la hemoglobina glicosilada del paciente iguala o supera el 6,5%. Nuestro propósito principal era comprobar si alguna de las otras variables medidas en estos pacientes presentaba una correlación sólida con respecto a la hemoglobina glicosilada, y, por tanto, si podía servir como marcador biologico para establecer el riesgo de diabetes antes de debutar con la enfermedad.

Nuestro análisis estadístico descriptivo reveló cierto nivel de correlación entre los niveles de hemoglobina glicosilada y los de la glucosa en ayunas. Por ello, decidimos realizar un análisis de regresión lineal simple con el propósito de comprobar si un modelo basado en esa correlación podría ayudarnos a predecir niveles altos de hemoglobina glicosilada en función del nivel de glucosa en ayunas. Como el modelo no parecía muy eficaz, realizamos una regresión lineal múltiple, que tampoco reveló una capacidad predictiva satisfactoria.

A continuación, decidimos emplear herramientas más potentes. Primero, comprobamos si un análisis de componentes principales nos permitía clasificar adecuadamente a los pacientes en diabéticos y no diabéticos, en función del resto de variables numéricas. Como esta prueba tampoco dio resultados sólidos, nos propusimos realizar una regresión logística, basada en el entrenamiento de un modelo y su posterior validación. Ajustando progresivamente los parámetros, conseguimos construir un modelo con una precisión, sensibilidad y especificidad muy buenas.

Finalmente, creamos una aplicación Shiny compleja que permite no sólo visualizar los datos de nuestro estudio, sino también hacer las representaciones gráficas correspondientes y el modelo de regresión logística de manera interactiva, ajustando las variables y otros parámetros.

Como conclusión, consideramos que hemos hecho un análisis estadístico completo y construido un modelo eficaz para clasificar a otros grupos de pacientes, distintos de los aquí analizados, en “diabéticos” o “no diabéticos”. Sin embargo, habría que explorar otro tipo de variables biológicas para ver si se pueden conseguir modelos multiparamétricos o uniparamétricos aún mejores. Tales modelos podrían ayudar a anticipar el debut de la diabetes, lo que facilitaría actuar de urgencia sobre el paciente para intentar evitarlo (por ejemplo, mejorando los hábitos alimenticios).

## 6.2 Valoración del trabajo y el informe

### 6.2.1 Valoración Enrique

El desarrollo de la actividad en colaboración con Otelo ha sido muy satisfactorio. Hemos llegado pronto a un acuerdo sobre el tipo de análisis que queríamos realizar y nos hemos puesto a trabajar en él, manteniendo reuniones periódicas de cierta extensión y trabajando también por nuestra cuenta (siguiendo las directrices fijadas en estas reuniones).

Nos hemos beneficiado mutuamente de un enfoque multidisciplinar, ya que provenimos de sectores académicos y profesionales opuestos en lo que a este Máster se refiere: Otelo es especialista en informática y programación, y yo en ciencias biológicas. Así, hemos colaborado aportando cada uno los conocimientos propios de nuestro campo y alcanzado una buena sinergia.

Hemos dedicado ambos una parte importante de nuestro tiempo a reflexionar sobre los datos y a ejecutar el análisis más preciso posible, en base a nuestros conocimientos. Nuestro objetivo ha sido siempre llegar al siguiente nivel para aprender lo máximo posible con esta asignatura. Pese a que tenemos todavía un notable espacio de mejora (dada la potencia de este lenguaje de programación), considero que el trabajo en grupo ha servido para consolidar los conocimientos de R y de estadística adquiridos durante el semestre.

Como ejemplo de algo a mejorar podría citar el uso de GitHub para coordinar el trabajo. Al no haber conseguido sincronizarlo con el R Studio, hemos tenido que ir subiendo las actualizaciones a mano cada uno por nuestra cuenta. Pero, así y todo, hemos sacado un buen partido de él.

El análisis de datos generado aporta muchísima información, ya que hemos puesto a punto un estudio descriptivo e inferencial muy completo, y hemos generado una herramienta de visualización muy atractiva y amable con el usuario. Estoy muy satisfecho con la calidad del informe final, que sirve como punto de partida para potenciar mis conocimientos de R y de estadística en el futuro.

### 6.2.2 Valoración Otelo

Por mi parte solo puedo decir cosas positivas del trabajo que hemos hecho. El trato con Enrique ha sido muy bueno desde el primer dia que nos conocimos. Ha sido fácil el llegar a acuerdos con él, desde la elección del dataset a estudiar pasando por todos los puntos del trabajo.

Hemos hecho varias sesiones de trabajo en conjunto donde debatíamos los puntos que nos habiamos comprometido a estudiar previamente, llegando a un consenso en estos puntos y proponiendo hitos para la siguiente sesión.

Aún con la diferencia de nuestros perfiles ( Enrique es bioquímico y yo ingeniero en informática) nos hemos sabido complementar bastante bien y creo que hemos aprendido el uno del otro.

A nivel técnico nos hemos organizado compartiendo el código y documentos a través de github, cosa que al principio parecia muy buena idea, pero al intentar usarlo desde Rstudio nos hemos encontrado con muchos problemas y al final hemos tenido que hacer las sincronizaciones a mano.

# 7 Sección 7. Bibliografía adicional

## 7.1 package installation

https://r-coder.com/install-r-packages/

https://forum.posit.co/t/is-it-a-good-idea-to-conditionally-load-install-libraries-at-the-beginning-of-a-

script/15334

https://stackoverflow.com/questions/4090169/elegant-way-to-check-for-missing-packages-and-install-them

## 7.2 uso de índices

https://rspatial.org/intr/4-indexing.html

## 7.3 splitting objects

https://www.geeksforgeeks.org/r-language/how-to-split-vector-and-data-frame-in-r/

## 7.4 mixing graphs

https://www.sthda.com/english/articles/24-ggpubr-publication-ready-plots/81-ggplot2-easy-way-to-mix-multiple-graphs-on-the-same-page/

## 7.5 correlation matrix

https://rpubs.com/tskam/Corrgram

https://www.geeksforgeeks.org/r-language/how-to-plot-a-correlation-matrix-into-a-graph-using-r/

## 7.6 PCA analysis

https://www.geeksforgeeks.org/r-language/principal-component-analysis-with-r-programming/

https://www.r-bloggers.com/2021/05/principal-component-analysis-pca-in-r/

https://www.sthda.com/english/articles/31-principal-component-methods-in-r-practical-guide/118-principal-component-analysis-in-r-prcomp-vs-princomp/

https://rpubs.com/cristina_gil/pca

https://www.datacamp.com/tutorial/pca-analysis-r

## 7.7 creating R functions

https://r4ds.had.co.nz/functions.html

https://www.r-bloggers.com/2022/04/how-to-create-your-own-functions-in-r/

https://www.dataquest.io/blog/write-functions-in-r/

https://www.w3schools.com/r/r_functions.asp

https://stackoverflow.com/questions/58626472/how-to-apply-the-same-function-to-several-variables-in-r

## 7.8 apply functions

https://www.geeksforgeeks.org/r-language/apply-lapply-sapply-and-tapply-in-r/

https://www.dataquest.io/blog/apply-functions-in-r-sapply-lapply-tapply/

## 7.9 Shapiro-Wilk test

https://rpubs.com/rubenortiz/192363

https://www.geeksforgeeks.org/r-language/normality-test-for-multi-grouped-data-in-r/?utm_source=chatgpt.com

## 7.10 Mann-Whitney / Wilcoxon-Silk test

https://guides.library.lincoln.ac.uk/mash/mann_whitney_R

https://www.sheffield.ac.uk/media/30589/download?attachment

https://rpubs.com/samidlimon/mannwhitney

https://rpubs.com/luis_fernandez_bernal/ejercicio_mann_whitney

https://www.rdocumentation.org/packages/stats/versions/3.6.2/topics/wilcox.test

https://www.sthda.com/english/wiki/unpaired-two-samples-wilcoxon-test-in-r

https://stackoverflow.com/questions/70210990/how-to-run-paired-wilcoxon-test-for-multiple-variables-using-a-for-loop?utm_source=chatgpt.com

https://www.sthda.com/english/articles/24-ggpubr-publication-ready-plots/76-add-p-values-and-significance-levels-to-ggplots/

https://search.r-project.org/CRAN/refmans/ggpubr/html/stat_compare_means.html

## 7.11 linear regression

https://www.datacamp.com/tutorial/multiple-linear-regression-r-tutorial

https://www.geeksforgeeks.org/machine-learning/multiple-linear-regression-using-r/

## 7.12 logistic regression

https://www.datacamp.com/tutorial/logistic-regression-R

https://www.sthda.com/english/articles/36-classification-methods-essentials/151-logistic-regression-essentials-in-r/

https://www.r-bloggers.com/2015/09/how-to-perform-a-logistic-regression-in-r/

https://www.geeksforgeeks.org/logistic-regression-in-r-programming/

https://rpubs.com/stevenazza7181/825139

## 7.13 confusion matrix

https://www.digitalocean.com/community/tutorials/confusion-matrix-in-r

https://rpubs.com/ljencisoo/509744

https://www.geeksforgeeks.org/r-language/confusion-matrix-in-r/

## 7.14 curva ROC

https://www.rdocumentation.org/packages/pROC/versions/1.18.5/topics/ggroc.roc

https://rpubs.com/arquez9512/599268

https://www.geeksforgeeks.org/r-language/plotting-roc-curve-in-r-programming/

## 7.15 ifelse() function

https://www.datamentor.io/r-programming/ifelse-function

https://www.datamentor.io/r-programming/ifelse-function

## 7.16 Shiny app

https://stackoverflow.com/questions/39436713/r-shiny-reactivevalues-vs-reactive

https://www.appsilon.com/post/r-shiny-reactivity

https://debruine.github.io/shinyintro/reactives.html

https://shiny.posit.co/r/getstarted/build-an-app/reactivity-essentials/reactive-elements.html

https://mastering-shiny.org/basic-reactivity.html

https://shiny.posit.co/r/getstarted/shiny-basics/lesson6/

https://rpubs.com/JairoAyala/Shiny_parte1

https://rpubs.com/cmlopera/introShiny

https://diegokoz.github.io/intro_ds/clase_6/06_explicacion.nb.html

https://shiny.posit.co/r/getstarted/shiny-basics/lesson1/

https://www.rdocumentation.org/packages/pROC/versions/1.18.5/topics/ggroc.roc

https://rstudio.github.io/bslib/reference/navset.html

https://rstudio.github.io/bslib/articles/theming/index.html

https://bootswatch.com/pulse/