Predicción de la diabetes

Daniela Cuesta - Paola Peralta Flores

Nota En este documento se agregaron comentarios los cuales estan en viñetas antes de cada codigo. En ciertos codigos se realizaban los mismos pasos por lo que evitamos la redundancia.

Intro

Este sería un ejemplo de examen. El siguiente conjunto de datos, consiste en predecir a pacientes basandonos en datos clínicos, si puede padecer diabetes o no.

Antes de cualquier método de clasificación, regresión o lo que sea, necesitamos explorar los datos.

Esto supone exámenes estadísticos inferenciales univariantes, bivariantes y multivariantes.

Pima Indians Diabetes Database

This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective of the dataset is to diagnostically predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset. Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian heritage.

Cargamos librerias

- Se cargan las librerias que se van a utilizar.

```
library(ggplot2)
library(dplyr)
```

```
Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union
```

```r
library(caret)
```

```
Loading required package: lattice
```

```r
library(e1071)
library(ggstatsplot)
```

```
You can cite this package as:
     Patil, I. (2021). Visualizations with statistical details: The 'ggstatsplot' approach.
     Journal of Open Source Software, 6(61), 3167, doi:10.21105/joss.03167
```

```r
library(ggside)
```

```
Registered S3 method overwritten by 'ggside':
  method from
  +.gg   ggplot2
```

Cargamos los datos

- La función **`read.csv()`** es utilizada para leer un archivo CSV (valores separados por comas) y almacenarlo en una variable. En este caso, el archivo "diabetes.csv" se ha leído y los datos se han almacenado en la variable **`datos`**.
- La función **`head()`** se utiliza para mostrar las primeras filas de un conjunto de datos. En este caso, **`head(datos)`** muestra las primeras filas del conjunto de datos **`datos`**.

```r
datos <- read.csv("./datos/diabetes.csv")
head(datos)
```

```
  Pregnancies Glucose BloodPressure SkinThickness Insulin  BMI
1           6     148            72            35       0 33.6
2           1      85            66            29       0 26.6
3           8     183            64             0       0 23.3
4           1      89            66            23      94 28.1
5           0     137            40            35     168 43.1
6           5     116            74             0       0 25.6
  DiabetesPedigreeFunction Age Outcome
1                    0.627  50       1
2                    0.351  31       0
3                    0.672  32       1
4                    0.167  21       0
5                    2.288  33       1
6                    0.201  30       0
```

Si echamos una búsqueda rápida en google, observamos que el pedigree, es eso, la historia familiar de diabetes. Por lo tanto, aquí podríamso hacer varias cosas ! Entre ellas, regresar los datos a dicha función, o clasificar según esta variable, considerarla o no considerarla.

Para empezar vamos a considerarla para ver la clasificación del modelo knn y bayes.

Miramos las clases de los datos

- La función **`str()`** se utiliza para mostrar la estructura interna de un objeto, lo que nos proporciona información sobre el tipo de datos y la estructura de cada columna en el conjunto de datos, se obtendrá una descripción detallada de la estructura de las columnas. Esto incluiría información como el nombre de cada columna, el tipo de datos de cada columna y una muestra de los valores presentes en cada columna.

```r
str(datos)
```

```
'data.frame':   768 obs. of  9 variables:
 $ Pregnancies             : int  6 1 8 1 0 5 3 10 2 8 ...
 $ Glucose                 : int  148 85 183 89 137 116 78 115 197 125 ...
 $ BloodPressure           : int  72 66 64 66 40 74 50 0 70 96 ...
 $ SkinThickness           : int  35 29 0 23 35 0 32 0 45 0 ...
 $ Insulin                 : int  0 0 0 94 168 0 88 0 543 0 ...
```

```
$ BMI                      : num  33.6 26.6 23.3 28.1 43.1 25.6 31 35.3 30.5 0 ...
$ DiabetesPedigreeFunction: num  0.627 0.351 0.672 0.167 2.288 ...
$ Age                      : int  50 31 32 21 33 30 26 29 53 54 ...
$ Outcome                  : int  1 0 1 0 1 0 1 0 1 1 ...
```

La única variable que debemos de cambiar es `Outcome` a factor. Donde 1 es diebetes, y 0 es no diabetes.

- En esta línea de código, se está realizando lo siguiente:

  1. **`datos$Outcome`**: Accede a la columna llamada "Outcome" dentro de "datos". La notación **`$`** se utiliza para acceder a una columna específica dentro de datos.
  2. **`as.factor()`**: Esta función se utiliza para convertir la columna en un factor. Un factor es una forma de representar variables categóricas, donde cada nivel de la variable se trata como una categoría distinta.
  3. El resultado de la función **`as.factor()`** se asigna nuevamente a la columna "Outcome" en el marco de datos "datos", utilizando el operador de asignación **`<-`**.

```
datos$Outcome  <- as.factor(datos$Outcome)
```

Análisis estadístico preliminar

- La línea de código "`dim(datos)`" es "dimensiones de datos".

  En este caso, "dim(datos)" se utiliza para obtener las dimensiones del marco de datos llamado "datos". Proporciona información sobre la cantidad de filas y columnas que tiene el marco de datos. El primer elemento del vector resultante será el número de filas y el segundo elemento será el número de columnas.

```
dim(datos)
```

```
[1] 768   9
```

Tenemos 768 filas y 9 columnas. Analicemos primero dos a dos las variables una por una

Histogramas

- La línea **`l.plots <- vector("list", length = ncol(datos)-1)`** crea una lista vacía llamada "l.plots" con una longitud igual al número de columnas de "datos" menos 1.
- La línea **`n1 <- ncol(datos) - 1`** calcula el valor de "n1" como el número de columnas de "datos" menos 1.
- Bucle "for" que itera sobre valores desde 1 hasta "n1". Dentro del bucle, se llevan a cabo las siguientes operaciones para cada valor "j":

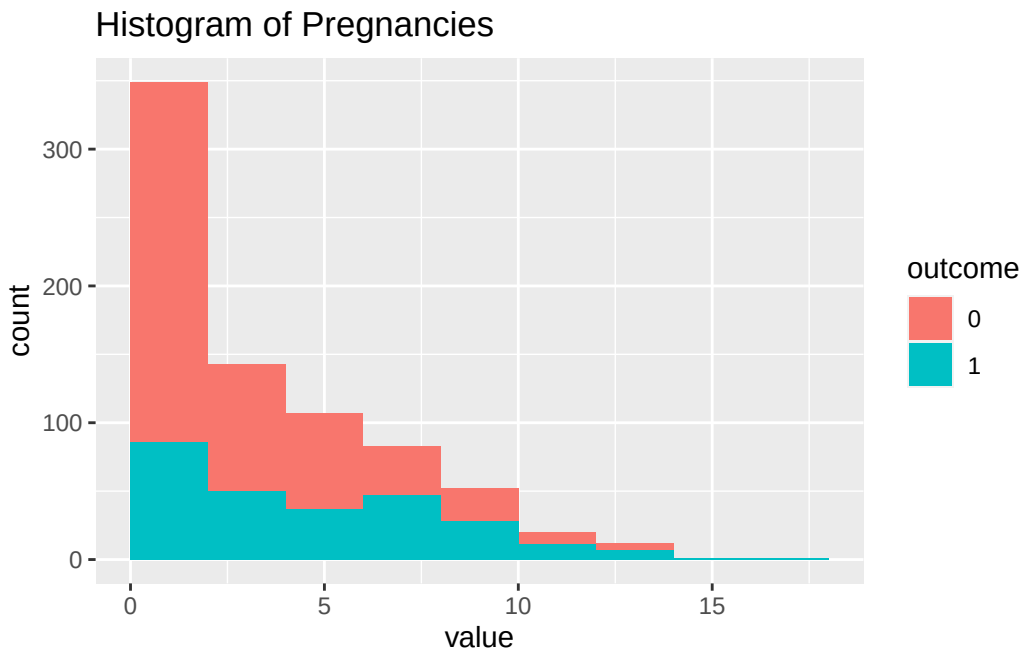
  1. **`h <- hist(datos[,j], plot = F)`**: Se calcula un histograma de la columna "j" del marco de datos "datos".
  2. **`datos.tmp <- data.frame(value = datos[,j], outcome = datos$Outcome)`**: Se crea un nuevo marco de datos llamado "datos.tmp". Este marco de datos contiene dos columnas: "value", que almacena los valores de la columna "j" de "datos", y "outcome", que contiene los valores de la columna "Outcome" de "datos".
  3. **`p1 <- ggplot(datos.tmp, aes(value, fill = outcome)) + geom_histogram(breaks = h$breaks) + ggtitle(paste("Histogram of", colnames(datos)[j]))`**: Se utiliza la biblioteca "ggplot2" para crear un objeto de trazado "ggplot".Se configura para usar el marco de datos "datos.tmp" y asignar los valores de "value" a lo largo del eje x, y se utiliza la variable "outcome" para colorear los histogramas según la categoría de "Outcome".

- Genera una serie de histogramas utilizando para cada columna en el marco de datos "datos". Cada histograma muestra la distribución de valores de la columna respectiva, coloreados según la categoría de "Outcome".
- Después de completar el bucle **`for`**, la lista **`l.plots`** contendrá los objetos de trazado generados para cada columna del marco de datos. Cada elemento de la lista será un objeto de trazado **`ggplot`** que representa el histograma de una columna en particular, coloreado según la categoría de "Outcome".

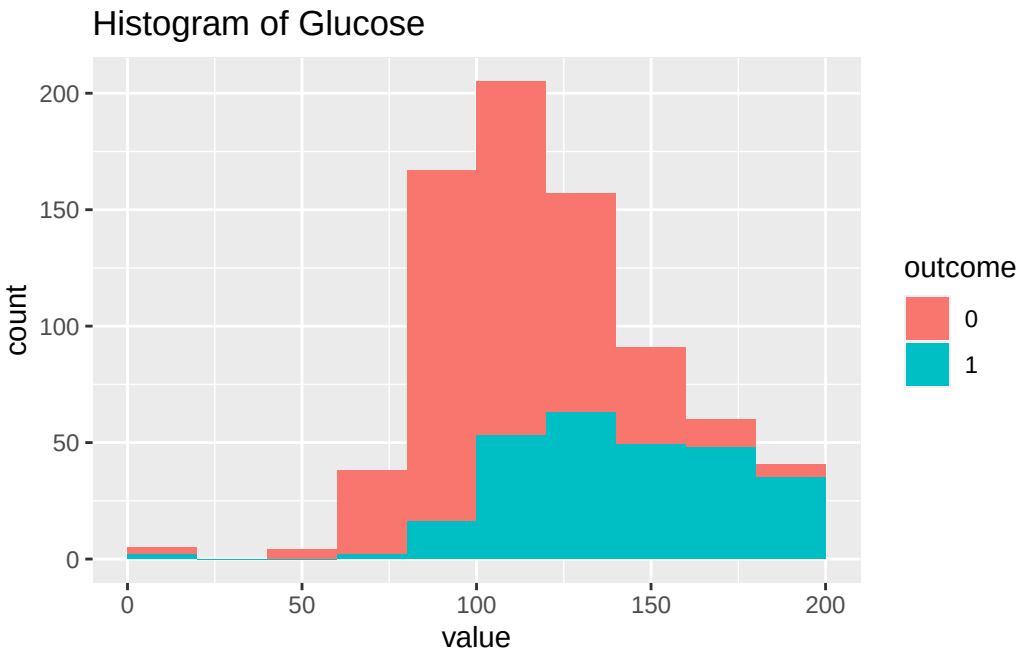
```
l.plots <- vector("list",length = ncol(datos)-1)
n1 <- ncol(datos) -1
for(j in 1:n1){

  h <-hist(datos[,j],plot = F)
  datos.tmp <- data.frame(value=datos[,j],outcome=datos$Outcome)
  p1 <- ggplot(datos.tmp,aes(value,fill=outcome))+geom_histogram(breaks=h$breaks) + ggtitl

  l.plots[[j]] <- p1
}
```

```
l.plots
```

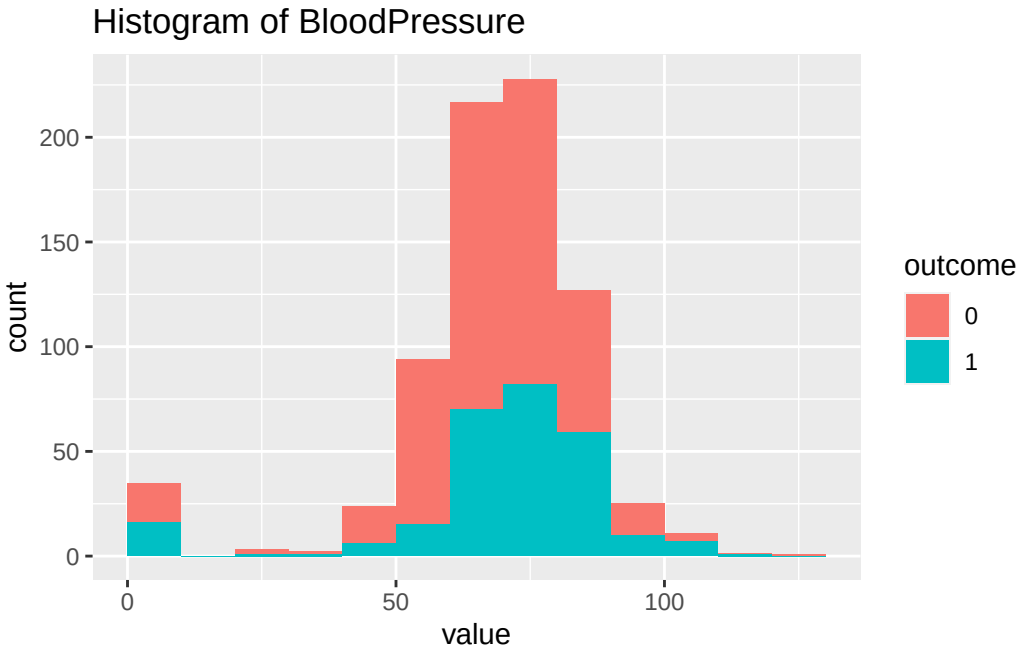
```
[[1]]
```


Histogram of Pregnancies
count
300
200
100
0
0
5
10
15
value
outcome
0
1

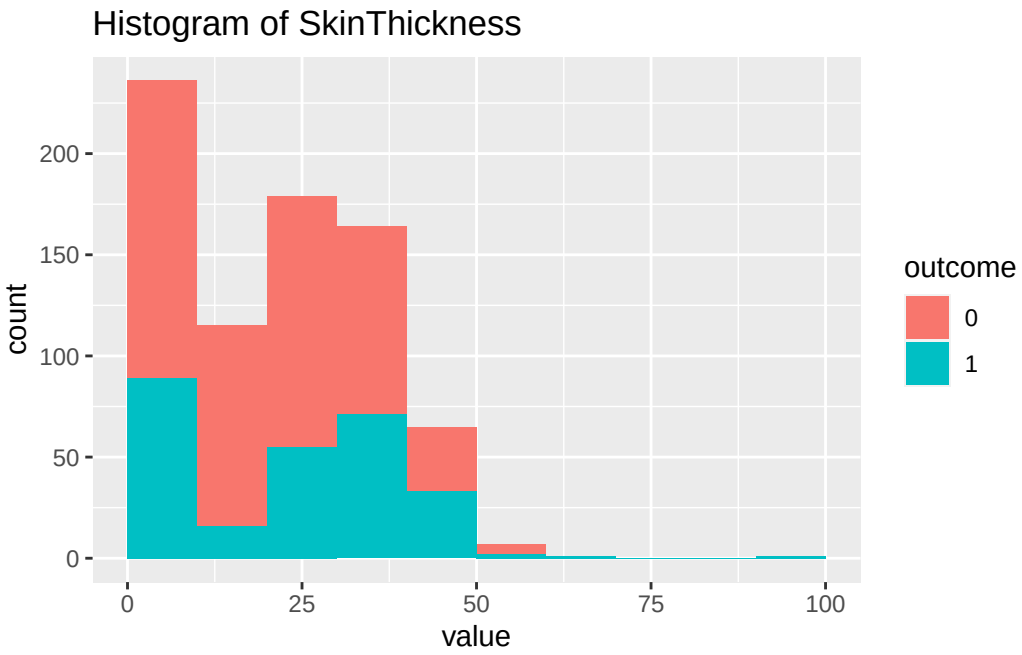
```
[[2]]
```


Histogram of Glucose
count
value
outcome
0
1
0
50
100
150
200

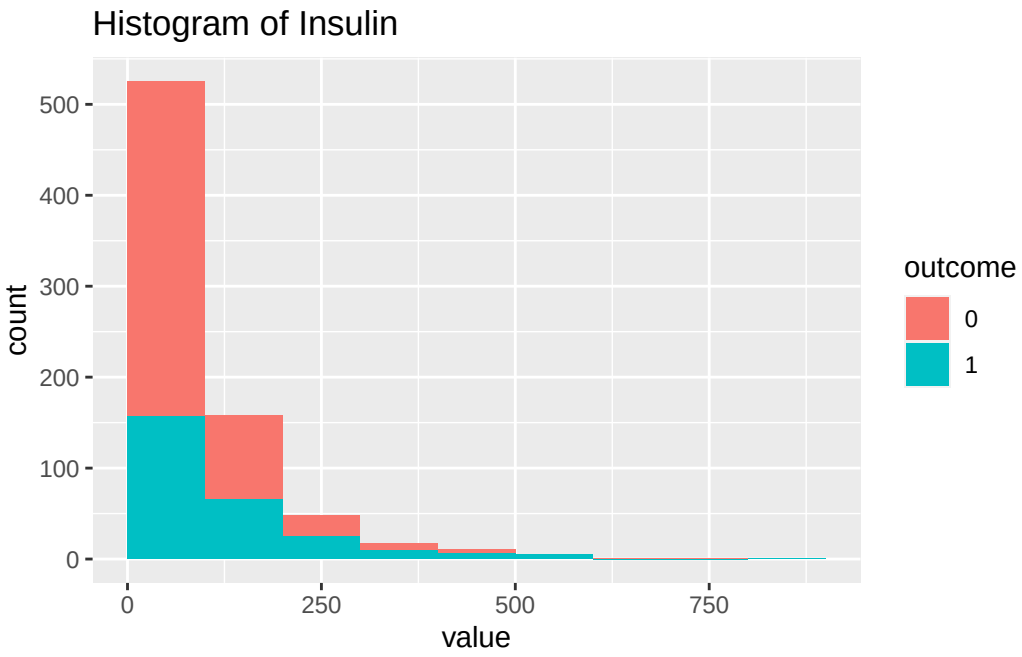
```
[[3]]
```


Histogram of BloodPressure
count
value
outcome
0
1
0
50
100
150
200

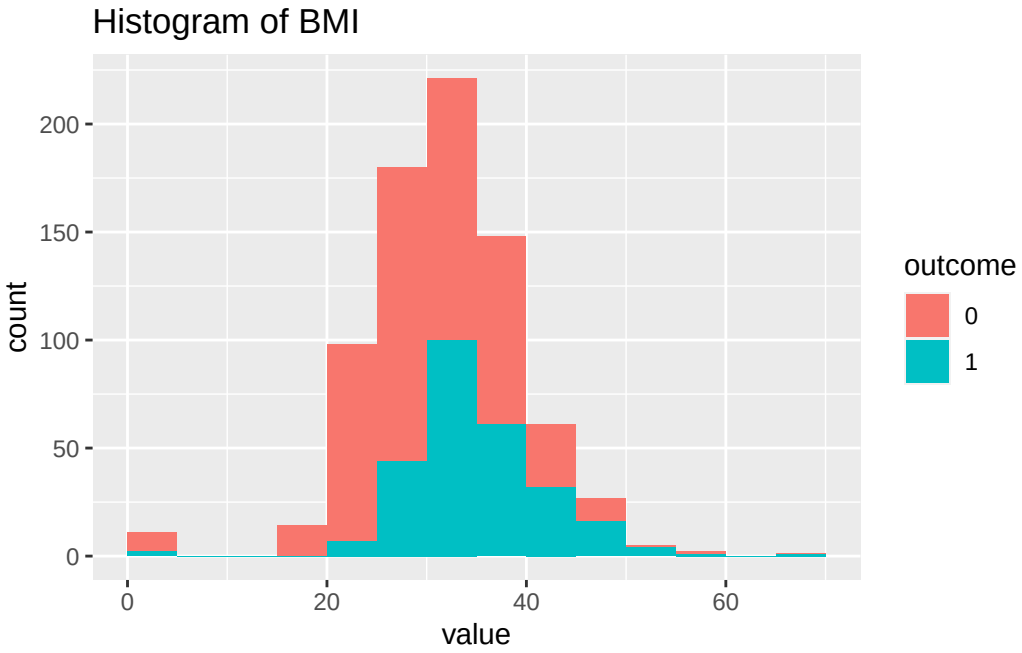
```
[[4]]
```


Histogram of SkinThickness
count
200
150
100
50
0
0
25
50
75
100
value
outcome
0
1

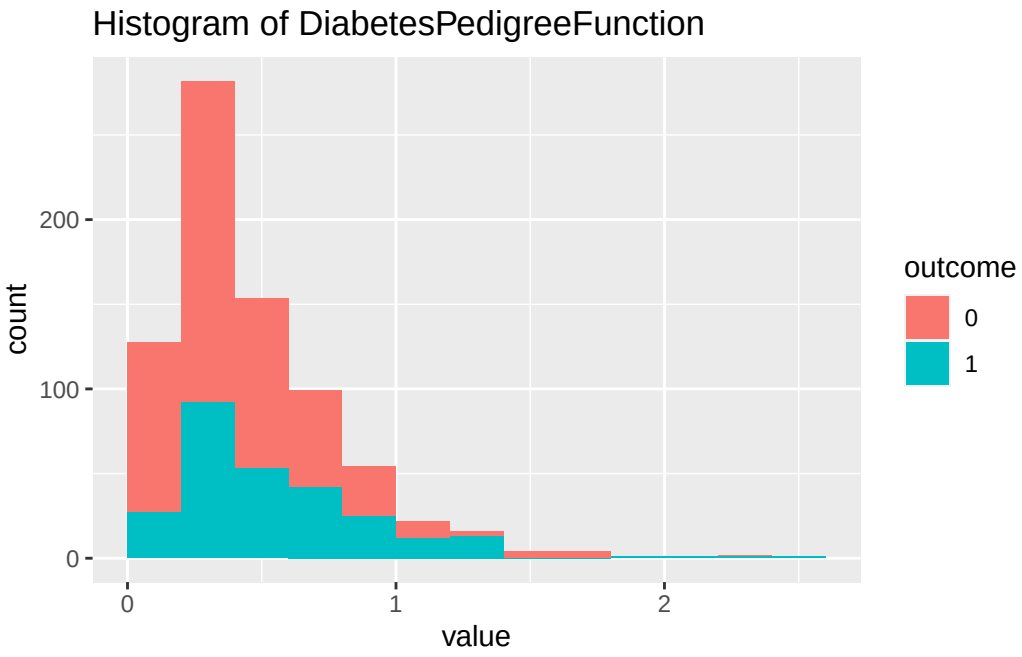
```
[[5]]
```


Histogram of Insulin
count
value
outcome
0
1
0
100
200
300
400
500
0
250
500
750

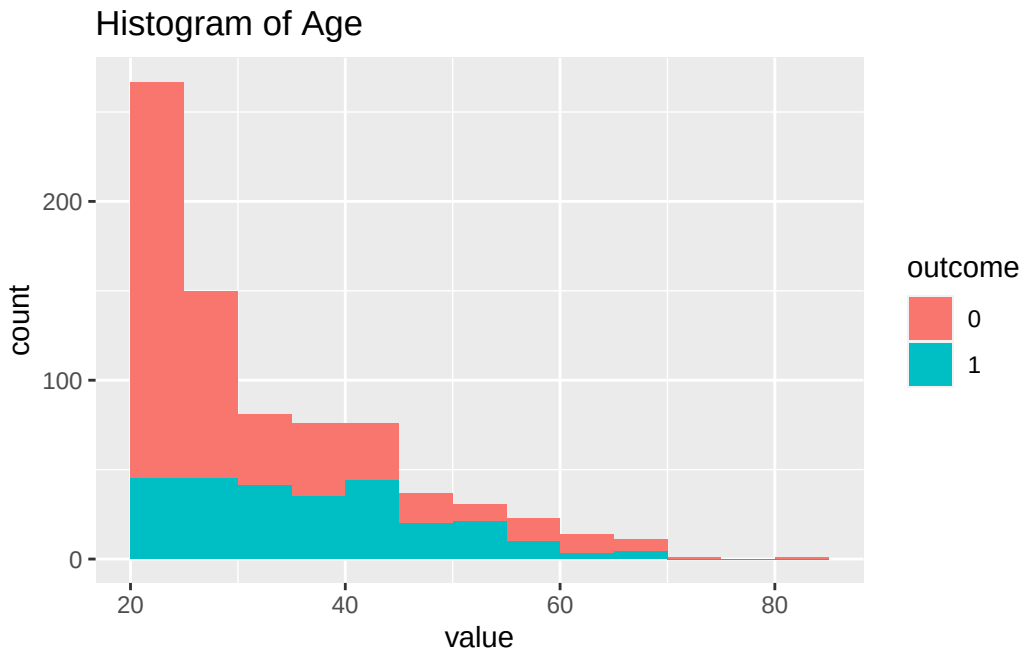
```
[[6]]
```


Histogram of BMI
count
value
outcome
0
1
0
50
100
150
200
0
20
40
60

[[7]]


Histogram of DiabetesPedigreeFunction
count
200
100
0
0
1
2
value
outcome
0
1

[[8]]

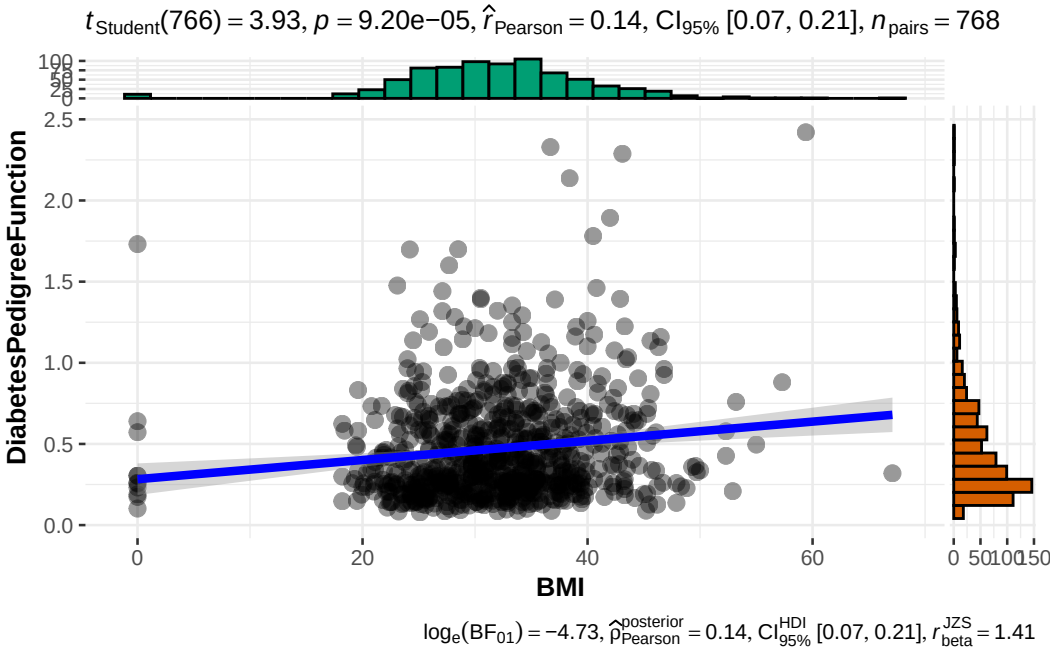


En lo particular la variable del pedigree se me hace importante, entonces vamos a realizar gráficos de dispersión

En realidad, una buena práctica es correlacionar todas contra todas...

```
ggscatterstats(datos,BMI,DiabetesPedigreeFunction)
```

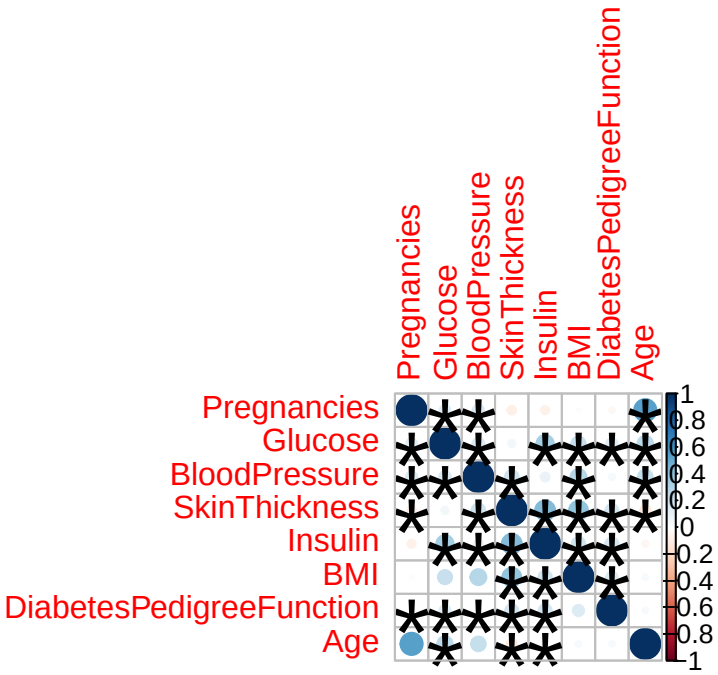
```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Sin embargo, esto puede ser un proceso tedioso… imaginad hacer 16 gráficas ! podemos condersarlo todo.

- Se realiza el análisis de correlación utilizando **`corr.test()`** de la biblioteca **`psych`**. Los resultados se almacenan en el objeto **`obj.cor`**.
- Se crea una matriz de valores (**`p.values`**) utilizando los valores de p originales y ajustados (**`obj.cor$p`** y **`obj.cor$p.adj`**, respectivamente).
- Luego, se configura la diagonal de **`p.values`** en 1 para asegurar que los valores de p en la diagonal principal no se muestren en el gráfico.
- Finalmente, se utiliza **`corrplot::corrplot()`** para crear el gráfico de correlación. un nivel de significancia de 0.05 (**`sig.level = 0.05`**) y se etiquetan las correlaciones insignificantes (**`insig = "label_sig"`**).

```
obj.cor <- psych::corr.test(datos[,1:n1])
p.values <- obj.cor$p
p.values[upper.tri(p.values)] <- obj.cor$p.adj
p.values[lower.tri(p.values)] <- obj.cor$p.adj
diag(p.values) <- 1
corrplot::corrplot(corr = obj.cor$r,p.mat = p.values,sig.level = 0.05,insig = "label_sig")
```



Ahora podemos proceder a hacer algo similar, con una serie de comparaciones dos a dos sobre las medias o medianas, sobre cada variable y la variable de interés.

Primero debemos aplicar una regresión linear con variable dependiente cada variable numérica y por la categórica. Es decir un t.test pero con el fin de ver los residuos, para ver la normalidad de éstos.

- El código realiza un cálculo de pruebas de normalidad para los residuos de modelos de regresión lineal aplicados a las columnas del marco de datos "datos" en función de la variable "Outcome".

    1. **`apply(datos[, 1:n1], 2, ...)`** aplica una función a cada columna del subconjunto de "datos" definido como **`datos[, 1:n1]`**.
    2. La función **`function(x) summary(lm(x ~ datos$Outcome))$residuals`** se aplica a cada columna. Realiza un ajuste de regresión lineal de la columna contra "Outcome" utilizando **`lm()`**. Luego, extrae los residuos del resumen del modelo utilizando **`summary()$residuals`**.
    3. **`apply(..., 2, shapiro.test)`** aplica la prueba de normalidad de Shapiro-Wilk (**`shapiro.test`**) a los residuos obtenidos en el paso anterior.
    4. El resultado de las pruebas de normalidad se asigna a la variable **`p.norm`**.

- Al ejecutar **`p.norm`**, obtendrás un vector de valores p que indica la significancia de la prueba de normalidad de Shapiro-Wilk para los residuos de cada columna en relación con

"Outcome". **Valores p pequeños (menores que el nivel de significancia deseado) sugieren que los residuos no siguen una distribución normal.**

```
p.norm <- apply(apply(datos[,1:n1],
                2,
                function(x) summary(lm(x~datos$Outcome))$residuals),
          2,
          shapiro.test)

p.norm
```

```
$Pregnancies

    Shapiro-Wilk normality test

data:  newX[, i]
W = 0.9389, p-value < 2.2e-16


$Glucose

    Shapiro-Wilk normality test

data:  newX[, i]
W = 0.97511, p-value = 3.726e-10


$BloodPressure

    Shapiro-Wilk normality test

data:  newX[, i]
W = 0.81468, p-value < 2.2e-16


$SkinThickness

    Shapiro-Wilk normality test

data:  newX[, i]
W = 0.92004, p-value < 2.2e-16
```

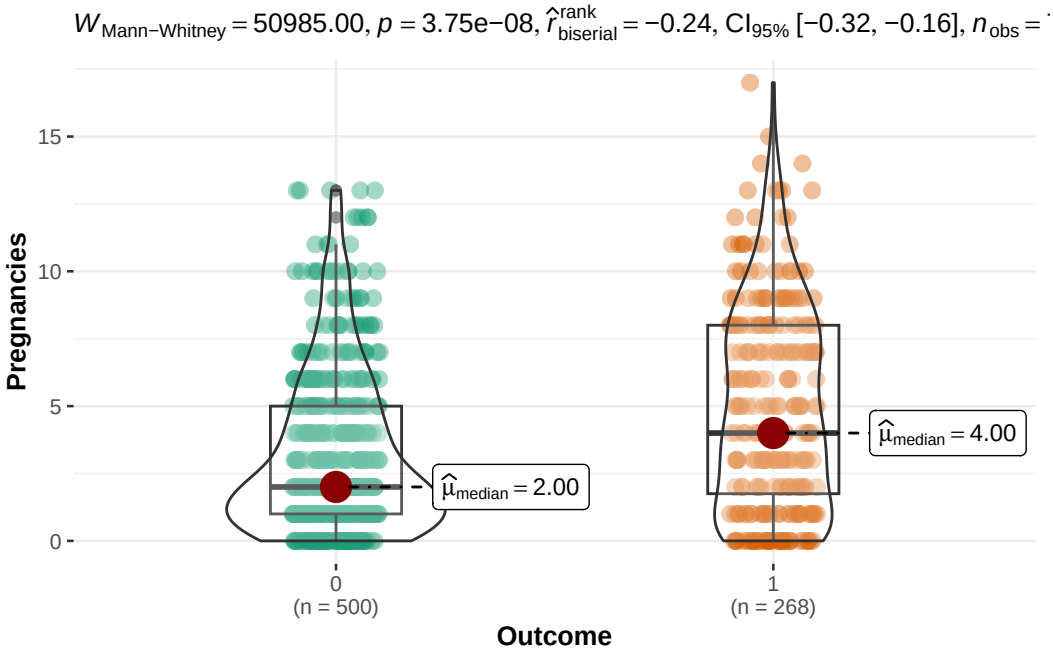
```
$Insulin

    Shapiro-Wilk normality test

data:  newX[, i]
W = 0.77776, p-value < 2.2e-16


$BMI

    Shapiro-Wilk normality test

data:  newX[, i]
W = 0.94359, p-value < 2.2e-16


$DiabetesPedigreeFunction

    Shapiro-Wilk normality test

data:  newX[, i]
W = 0.84939, p-value < 2.2e-16


$Age

    Shapiro-Wilk normality test

data:  newX[, i]
W = 0.88114, p-value < 2.2e-16
```

Todas las variables son no normales, tal como vemos en los histogramas.

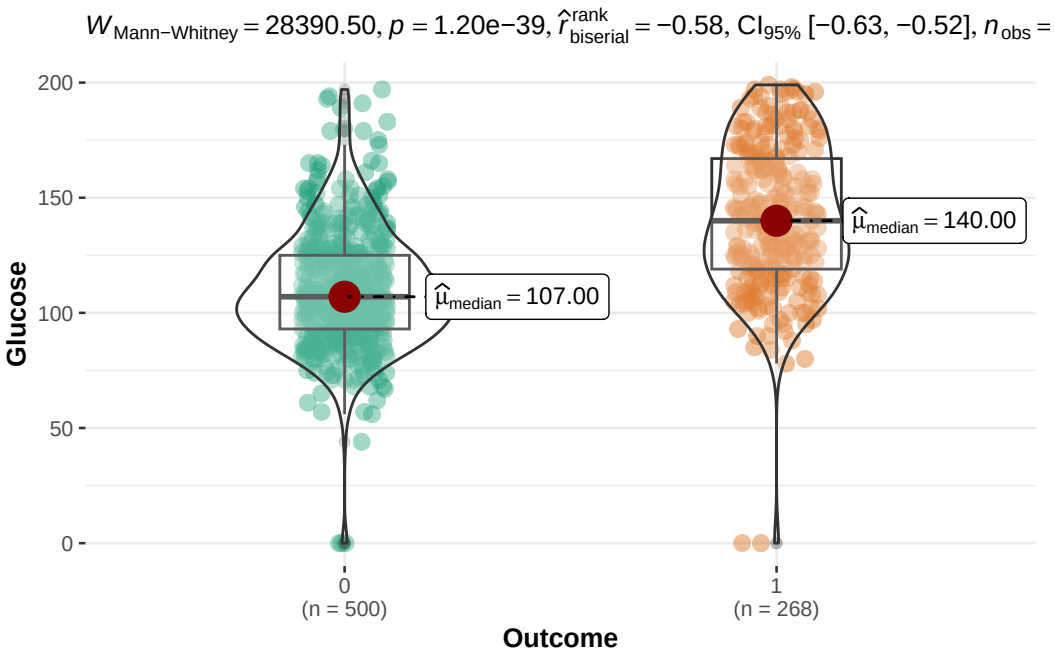
- El código **`ggbetweenstats(datos, Outcome, Pregnancies, type = "nonparametric")`** utiliza una función llamada **`ggbetweenstats`** para realizar un análisis estadístico comparando la variable "Pregnancies" entre diferentes niveles de la variable "Outcome" en el marco de datos "datos". El argumento **`type = "nonparametric"`** indica que se utilizará un enfoque no paramétrico en el análisis.

  Lo mismo es para todas las líneas de código que están a continuación:

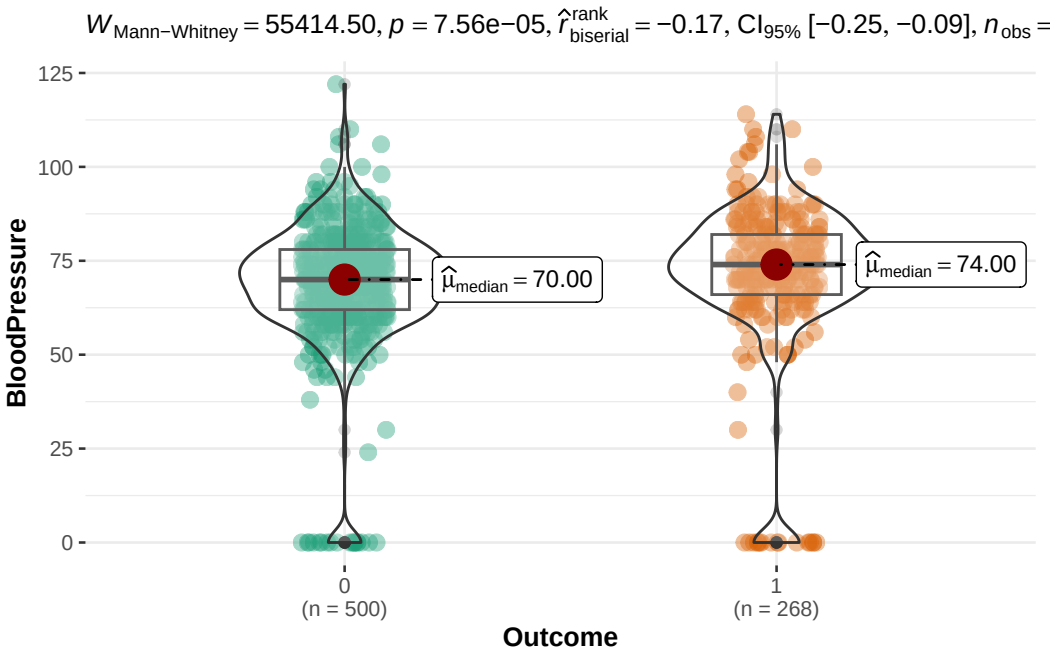
```
ggbetweenstats(datos,Outcome,Pregnancies,type = "nonparametric")
```



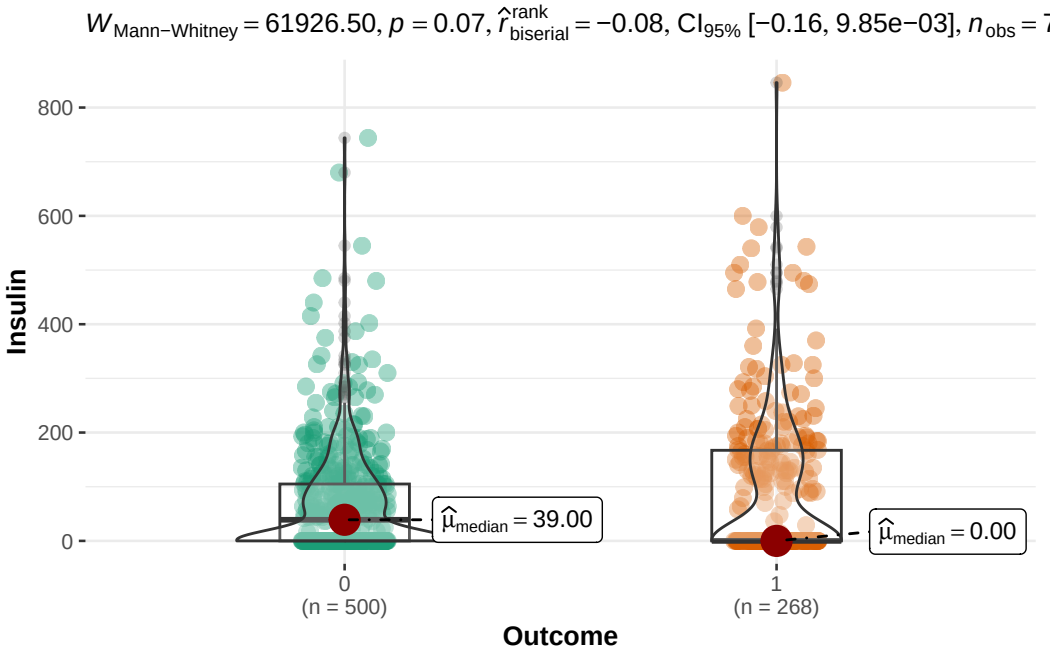

```
ggbetweenstats(datos,Outcome,Glucose,type = "nonparametric")
```



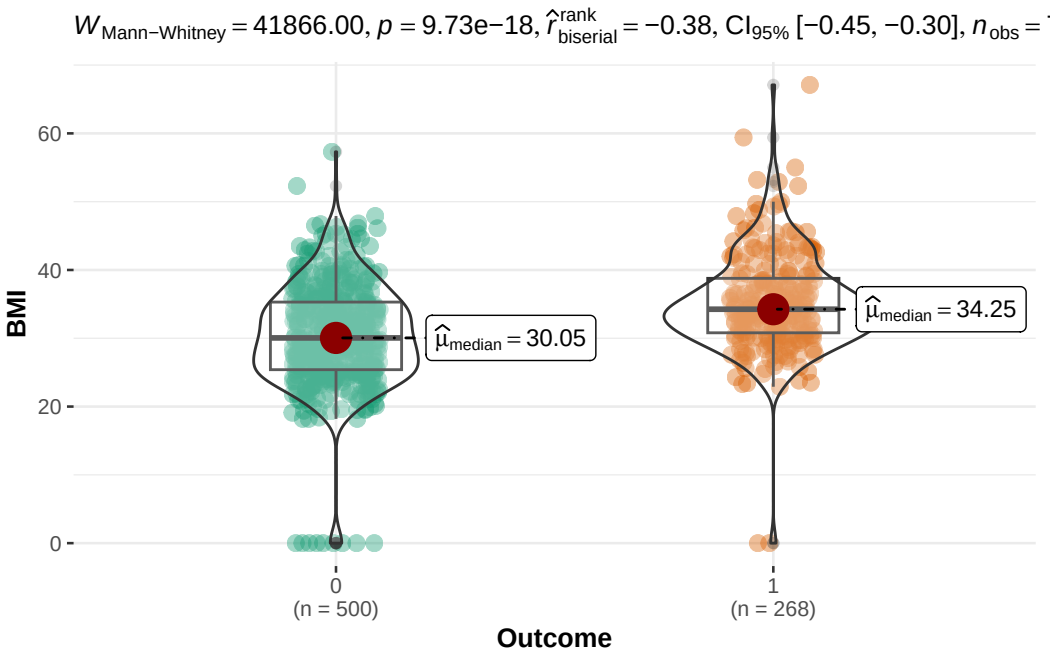

```
ggbetweenstats(datos,Outcome,BloodPressure,type = "nonparametric")
```



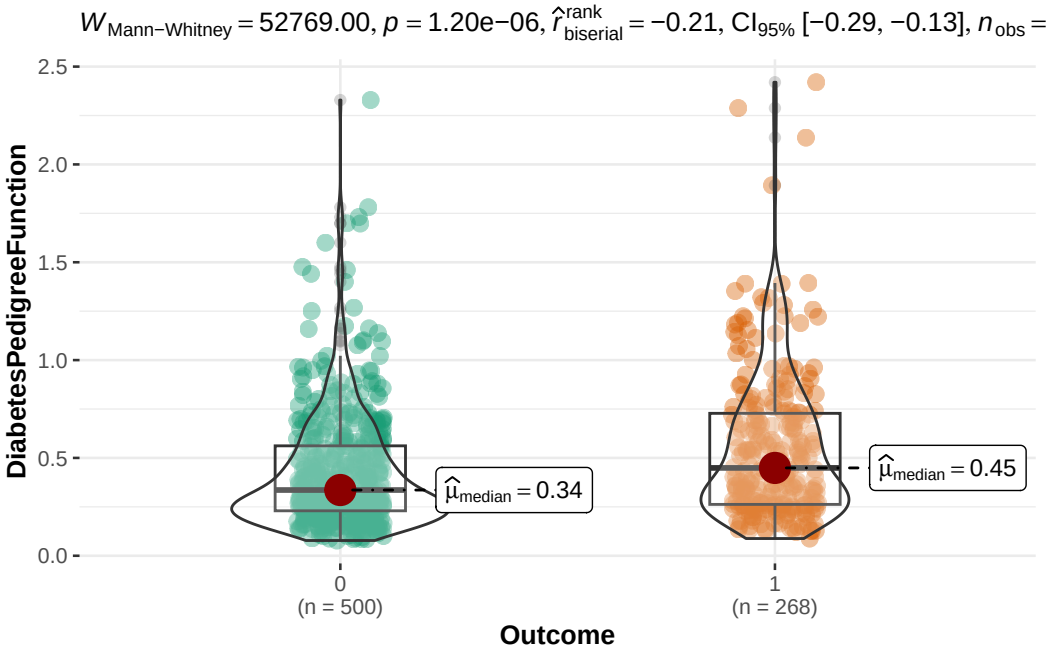
```
ggbetweenstats(datos,Outcome,Insulin,type = "nonparametric")
```



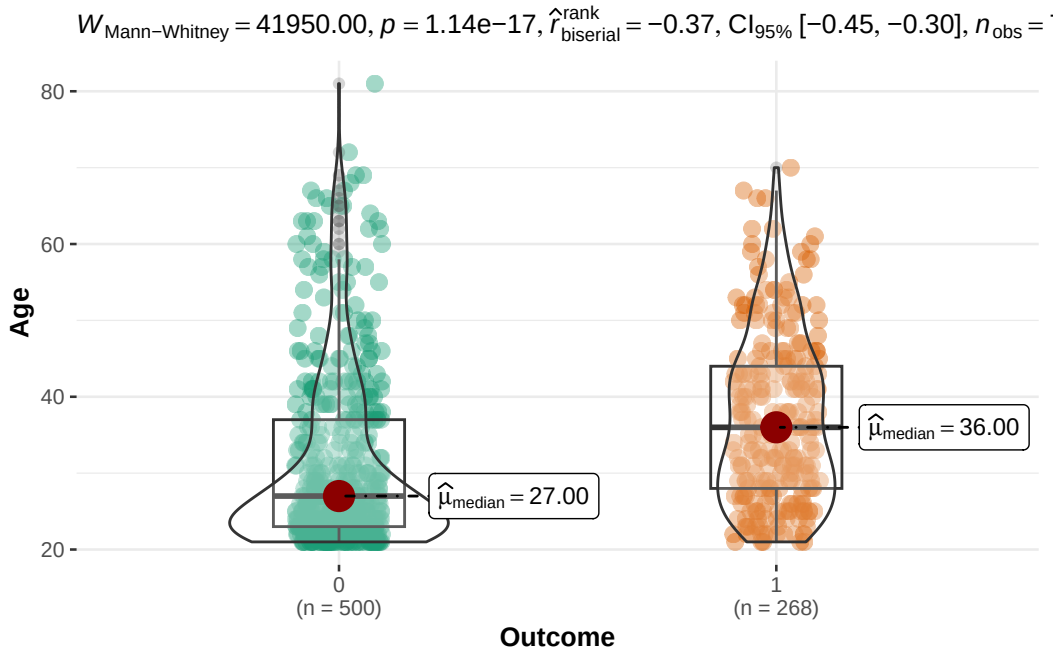

```
ggbetweenstats(datos,Outcome,BMI,type = "nonparametric")
```




```
ggbetweenstats(datos,Outcome,DiabetesPedigreeFunction,type = "nonparametric")
```



```
ggbetweenstats(datos,Outcome,Age,type = "nonparametric")
```



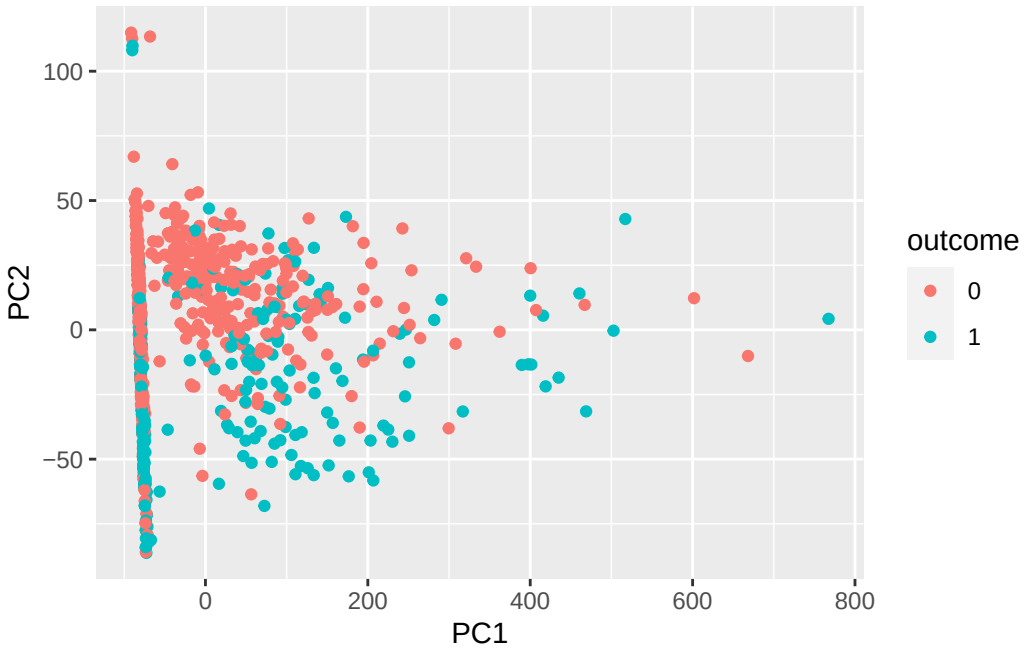
PCA

- El código **`summary(datos)`** Muestra un resumen estadístico de los datos contenidos en el marco de datos "datos". Proporciona una descripción de las variables, incluyendo medidas de tendencia central, dispersión y otros estadísticos relevantes.
- La siguiente línea Esta realiza el análisis de componentes principales utilizando la función **`prcomp()`**. Se aplica a un subconjunto de columnas de "datos"
- La siguiente línea combina las coordenadas de los componentes principales (**`pcx$x`**) con la variable "Outcome" de "datos". El resultado se almacena en el marco de datos **`plotpca`**, que ahora t**iene las coordenadas de los componentes principales y la información de "Outcome".**
- Finalmente, se crea un gráfico de dispersión de las dos primeras componentes principales (**`PC1`** y **`PC2`**) utilizando **`geom_point()`**. Los puntos se colorearán según la variable "Outcome" del marco de datos **`plotpca`**.

```
summary(datos)
```

```
 Pregnancies        Glucose      BloodPressure    SkinThickness
Min.   : 0.000   Min.   :  0.0   Min.   :  0.00   Min.   : 0.00
1st Qu.: 1.000   1st Qu.: 99.0   1st Qu.: 62.00   1st Qu.: 0.00
Median : 3.000   Median :117.0   Median : 72.00   Median :23.00
Mean   : 3.845   Mean   :120.9   Mean   : 69.11   Mean   :20.54
3rd Qu.: 6.000   3rd Qu.:140.2   3rd Qu.: 80.00   3rd Qu.:32.00
Max.   :17.000   Max.   :199.0   Max.   :122.00   Max.   :99.00
   Insulin           BMI        DiabetesPedigreeFunction      Age
Min.   :  0.0   Min.   : 0.00   Min.   :0.0780           Min.   :21.00
1st Qu.:  0.0   1st Qu.:27.30   1st Qu.:0.2437           1st Qu.:24.00
Median : 30.5   Median :32.00   Median :0.3725           Median :29.00
Mean   : 79.8   Mean   :31.99   Mean   :0.4719           Mean   :33.24
3rd Qu.:127.2   3rd Qu.:36.60   3rd Qu.:0.6262           3rd Qu.:41.00
Max.   :846.0   Max.   :67.10   Max.   :2.4200           Max.   :81.00
Outcome
0:500
1:268
```

```r
pcx <- prcomp(datos[,1:n1],scale. = F) ## escalamos por la variablidad de los datos

plotpca <- bind_cols(pcx$x,outcome=datos$Outcome)
ggplot(plotpca,aes(PC1,PC2,color=outcome))+geom_point()
```



Ahora vamos a ver si haciendo unas transformaciones esto cambia. Pero antes debemos de ver las variables sospechosas...

Pero de igual manera podemos escalar a ver si hay algun cambio...

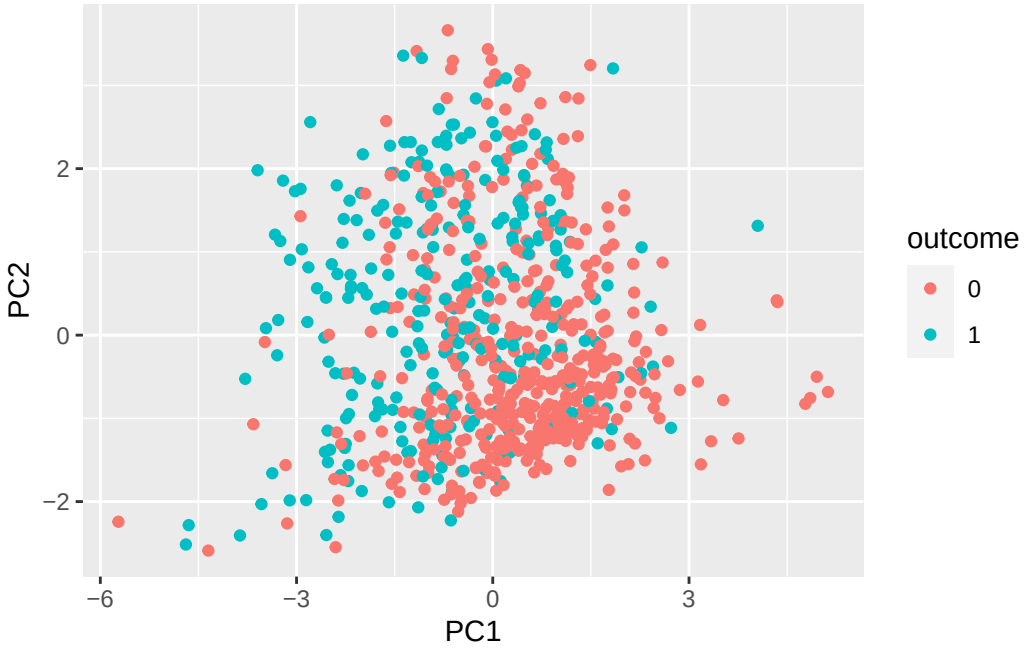
- El código actualizado realiza un análisis de componentes principales (PCA) utilizando la función **`prcomp()`** en R y crea un gráfico de dispersión de las dos primeras componentes principales utilizando la biblioteca **`ggplot2`**. La principal diferencia con el código anterior es que ahora se escala los datos antes de realizar el PCA mediante **`scale. = TRUE`**
- El argumento **`scale. = TRUE`** indica que los datos se deben escalar antes de realizar el PCA, lo cual significa que las variables se ajustarán para tener una varianza unitaria.

```
summary(datos)
```

```
 Pregnancies        Glucose      BloodPressure    SkinThickness
Min.   : 0.000   Min.   :  0.0   Min.   :  0.00   Min.   : 0.00
1st Qu.: 1.000   1st Qu.: 99.0   1st Qu.: 62.00   1st Qu.: 0.00
Median : 3.000   Median :117.0   Median : 72.00   Median :23.00
Mean   : 3.845   Mean   :120.9   Mean   : 69.11   Mean   :20.54
3rd Qu.: 6.000   3rd Qu.:140.2   3rd Qu.: 80.00   3rd Qu.:32.00
Max.   :17.000   Max.   :199.0   Max.   :122.00   Max.   :99.00
```

```
   Insulin           BMI        DiabetesPedigreeFunction      Age
Min.   :  0.0   Min.   : 0.00   Min.   :0.0780           Min.   :21.00
1st Qu.:  0.0   1st Qu.:27.30   1st Qu.:0.2437           1st Qu.:24.00
Median : 30.5   Median :32.00   Median :0.3725           Median :29.00
Mean   : 79.8   Mean   :31.99   Mean   :0.4719           Mean   :33.24
3rd Qu.:127.2   3rd Qu.:36.60   3rd Qu.:0.6262           3rd Qu.:41.00
Max.   :846.0   Max.   :67.10   Max.   :2.4200           Max.   :81.00
Outcome
0:500
1:268
```

```r
pcx <- prcomp(datos[,1:n1],scale. = T) ## escalamos por la variablidad de los datos

plotpca <- bind_cols(pcx$x,outcome=datos$Outcome)
ggplot(plotpca,aes(PC1,PC2,color=outcome))+geom_point()
```

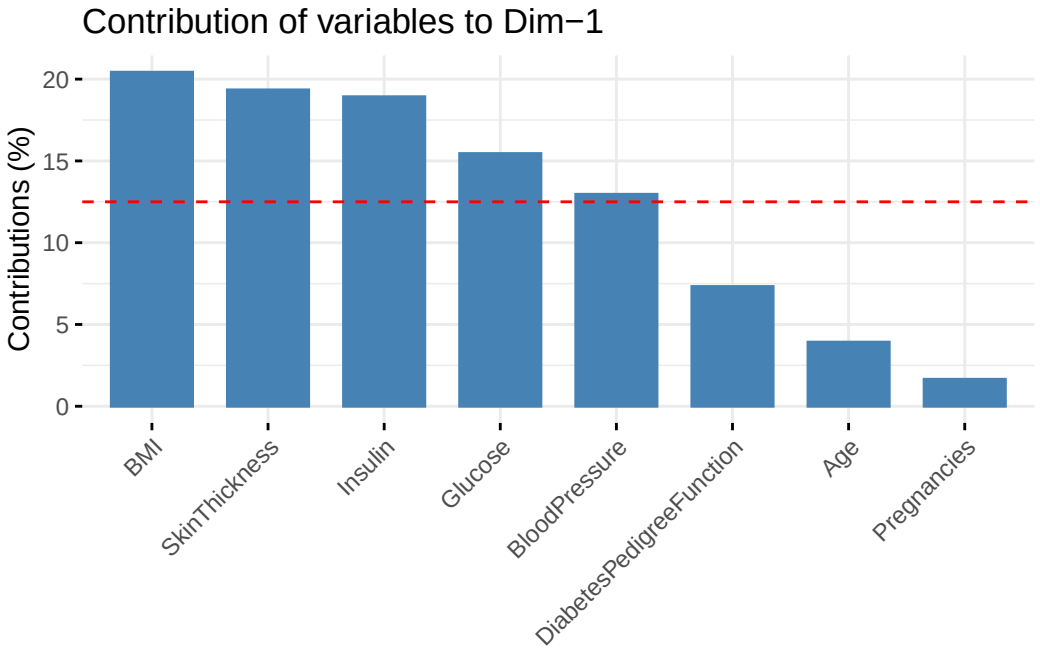


- La línea de código **`factoextra::fviz_contrib(pcx, "var")`** muestra un gráfico que ilustra las contribuciones de las variables al PCA realizado. Estas contribuciones pueden

ayudar a comprender qué variables tienen un mayor impacto en la estructura de los datos y cómo se relacionan con los componentes principales obtenidos.

- **pcx**: El objeto que contiene el resultado del análisis de componentes principales realizado previamente mediante **prcomp()**.
- **"var"**: Indica que se deben visualizar las contribuciones de las variables.

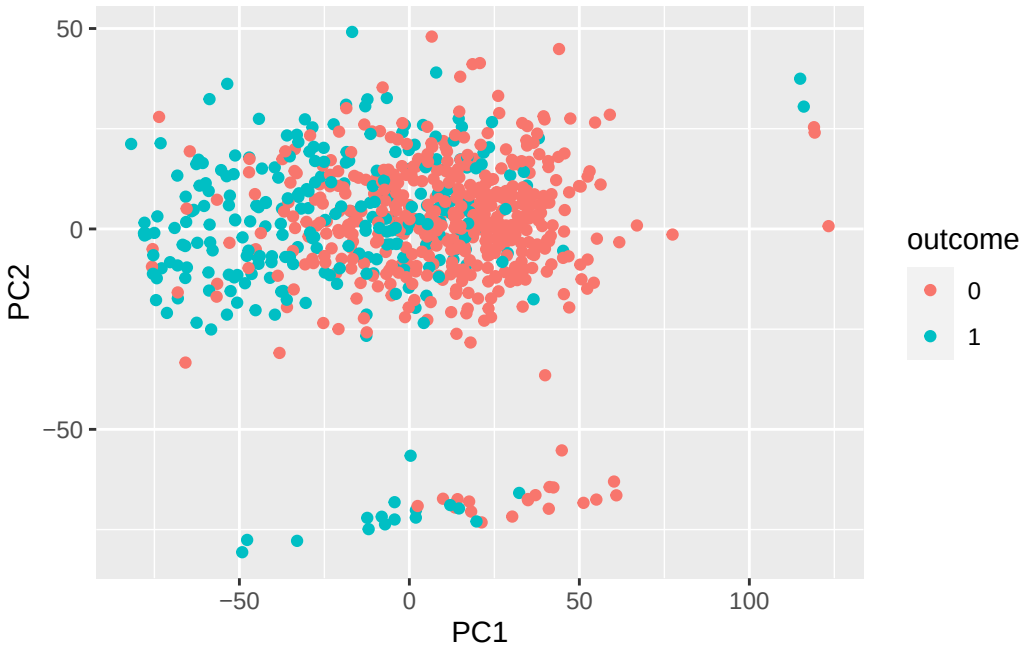
```
factoextra::fviz_contrib(pcx,"var")
```



Al parecer es la insulina la que está dando problemas.

- Esta línea utiliza la función **grep()** para buscar el término "insulin" en los nombres de las variables (**colnames(datos)**) del marco de datos "datos". Los índices de las columnas que contienen "insulin" se almacenan en el vector **w**.
- Esta línea realiza el análisis de componentes principales utilizando la función **prcomp()**. Se seleccionan todas las columnas de "datos" excepto las columnas indicadas en **w**.
- Se utiliza **ggplot2** para crear un gráfico de dispersión de las dos primeras componentes principales (**PC1** y **PC2**) utilizando **geom_point()**.

```r
## indices a quitar
w <- c(grep("insulin",ignore.case = T,colnames(datos)),ncol(datos))
pcx <- prcomp(datos[,-w],scale. = F) ## escalamos por la variablidad de los datos

plotpca <- bind_cols(pcx$x,outcome=datos$Outcome)
ggplot(plotpca,aes(PC1,PC2,color=outcome))+geom_point()
```



De hecho la insulina, tenía un aspecto raro, como sesgado, ver gráficos de arriba. Vamos a transformarla...

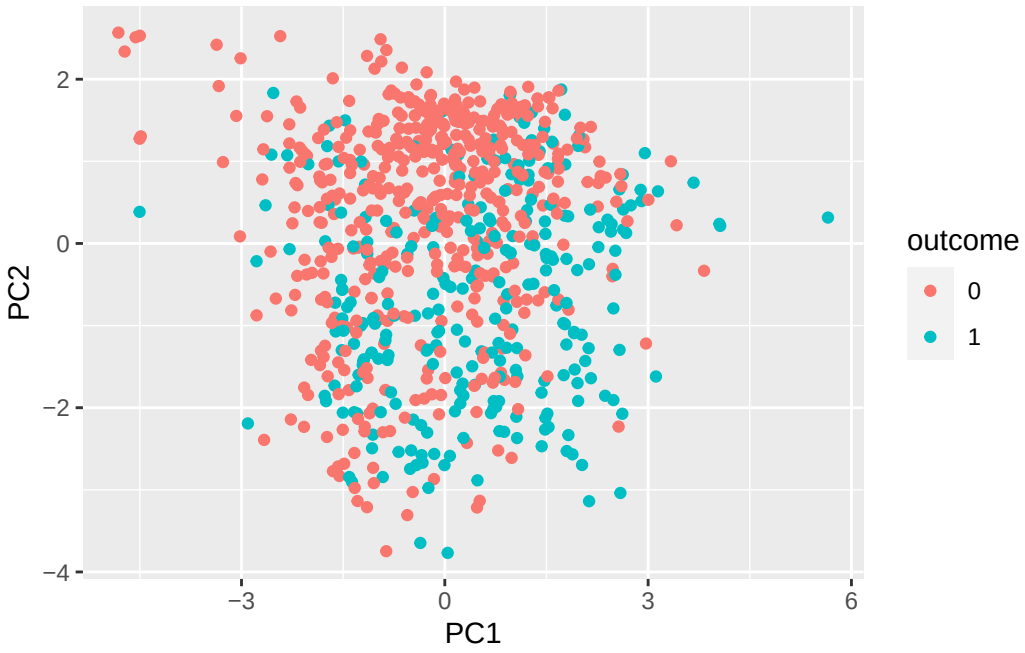
- Se realiza una transformación logarítmica en la variable "Insulin" de los datos antes de realizar el análisis de componentes principales (PCA). Luego se crea un gráfico de dispersión de las dos primeras componentes principales utilizando la biblioteca **ggplot2**.

```r
datos$Insulin  <- log(datos$Insulin+0.05)

summary(datos)
```

```
  Pregnancies        Glucose      BloodPressure    SkinThickness
 Min.   : 0.000   Min.   :  0.0   Min.   :  0.00   Min.   : 0.00
 1st Qu.: 1.000   1st Qu.: 99.0   1st Qu.: 62.00   1st Qu.: 0.00
```

```
Median : 3.000   Median :117.0   Median : 72.00   Median :23.00
Mean   : 3.845   Mean   :120.9   Mean   : 69.11   Mean   :20.54
3rd Qu.: 6.000   3rd Qu.:140.2   3rd Qu.: 80.00   3rd Qu.:32.00
Max.   :17.000   Max.   :199.0   Max.   :122.00   Max.   :99.00
   Insulin            BMI        DiabetesPedigreeFunction      Age
Min.   :-2.996   Min.   : 0.00   Min.   :0.0780           Min.   :21.00
1st Qu.:-2.996   1st Qu.:27.30   1st Qu.:0.2437           1st Qu.:24.00
Median : 3.418   Median :32.00   Median :0.3725           Median :29.00
Mean   : 1.008   Mean   :31.99   Mean   :0.4719           Mean   :33.24
3rd Qu.: 4.847   3rd Qu.:36.60   3rd Qu.:0.6262           3rd Qu.:41.00
Max.   : 6.741   Max.   :67.10   Max.   :2.4200           Max.   :81.00
Outcome
0:500
1:268
```

```r
pcx <- prcomp(datos[,1:n1],scale. = T) ## escalamos por la variablidad de los datos

plotpca <- bind_cols(pcx$x,outcome=datos$Outcome)
ggplot(plotpca,aes(PC1,PC2,color=outcome))+geom_point()
```



Cambia ! Esto significa que no hemos quitado la infromacion de la insulina, solamente lo hemos transformado

Es decir, cambia si transformamos los datos…a partir de esto, podemos realizar de nuevo pruebas de diferencia de medianas, pero ahora lo veremos condensado..

- Se leen los datos de un archivo CSV utilizando la función **`read.csv()`** y se asignan al objeto **`datos`**. Luego, se convierte la variable "Outcome" en un factor utilizando la función **`as.factor()`**. A continuación, se escalan las variables del marco de datos utilizando la función **`scale()`**, se utiliza para centrar las variables en cero y escalarlas para que tengan una desviación estándar de uno.

```
datos <- read.csv("./datos/diabetes.csv")
datos$Outcome <- as.factor(datos$Outcome)
datsc <- scale(datos[,-ncol(datos)])
```

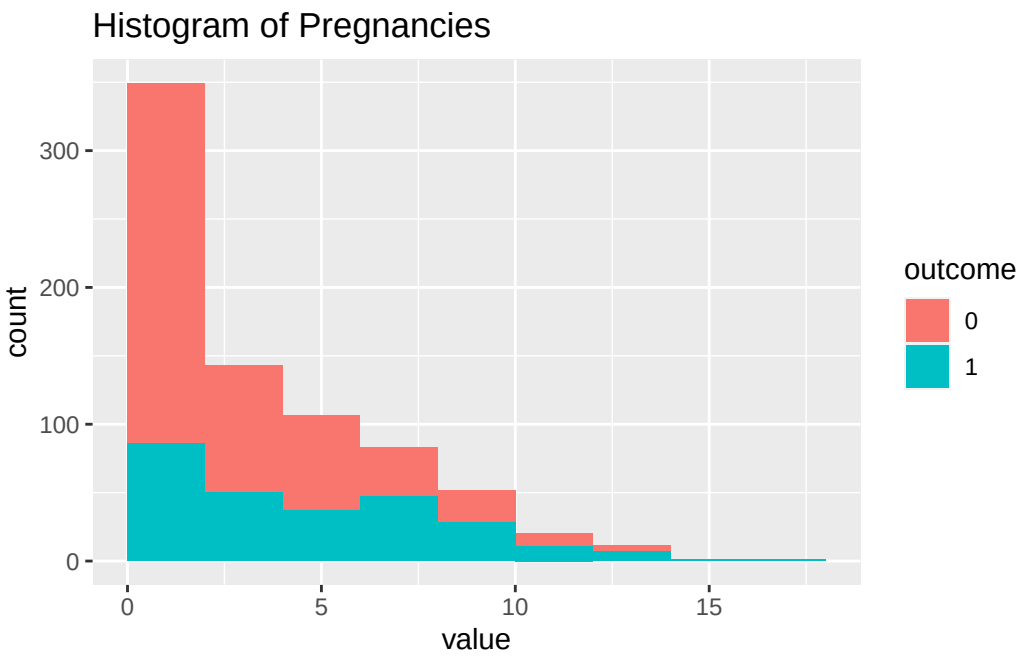
Veamos las distribuciones de nuevo….

- Crea una lista llamada **`l.plots`** utilizando la función **`vector()`**. Mediante **`length = ncol(datos)-1`**, se define la longitud de la lista, que es igual al número de columnas del conjunto de datos menos 1.
- Calcula el número de columnas en **`datos`** y lo almacena en la variable **`n1`**. Resta 1 para que no se incluya la columna "Outcome".

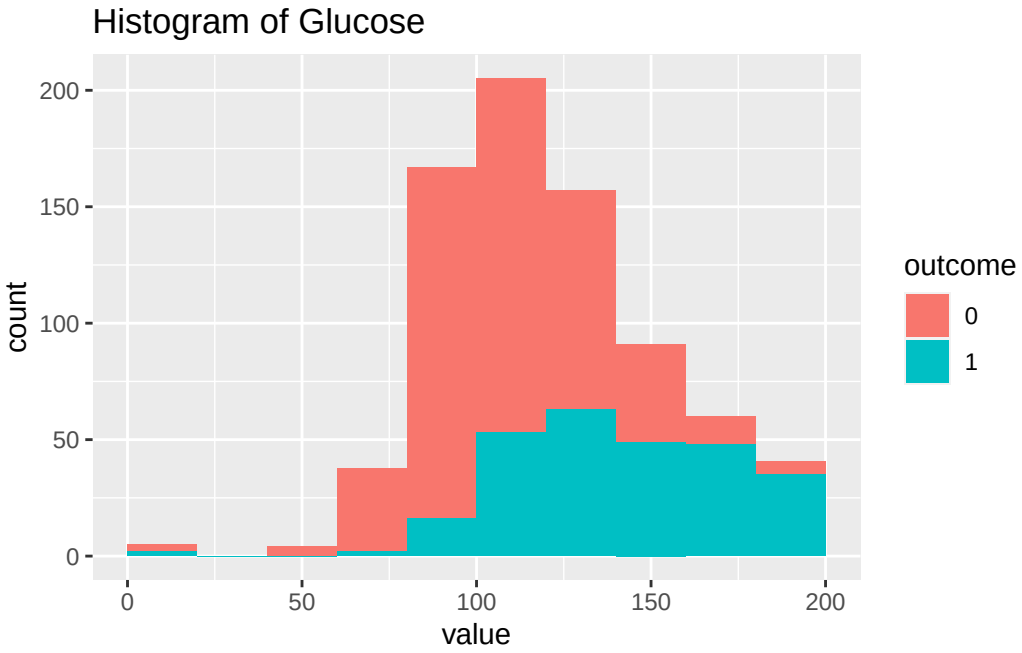
- Inicia un bucle **for** que iterará desde 1 hasta **n1**. Dentro del bucle, se realiza lo siguiente:
  - Se utiliza la función **hist()** para crear un histograma de la columna **j** de **datos** y **plot = F** se utiliza para indicar que el histograma no debe ser trazado en este paso, sino solo calculado y almacenado en la variable **h**.
  - Se crea **datos.tmp** que contiene dos columnas: "value" (valores de la columna **j**) y "outcome" (valores de la columna "Outcome").
  - Se utiliza la función **ggplot()** para un gráfico utilizando **datos.tmp** como datos y especificando **value** como el eje x y **fill** como el atributo que determina el color de las barras en el histograma.
  - Se agrega una capa de histograma al gráfico utilizando **geom_histogram()** y se establece **breaks** en **h$breaks** para usar los intervalos de clase calculados anteriormente en el paso 4a.
  - Se utiliza **ggtitle()** para agregar un título al gráfico. El gráfico resultante se almacena en el elemento **j** de la lista **l.plots** utilizando la notación de doble corchete (**[[j]]**).
- Después de que el bucle **for** haya terminado, se imprime la lista **l.plots**.

```
l.plots <- vector("list",length = ncol(datos)-1)
n1 <- ncol(datos) -1
for(j in 1:n1){

  h <-hist(datos[,j],plot = F)
  datos.tmp <- data.frame(value=datos[,j],outcome=datos$Outcome)
  p1 <- ggplot(datos.tmp,aes(value,fill=outcome))+geom_histogram(breaks=h$breaks) + ggtitl

  l.plots[[j]] <- p1
}
l.plots
```

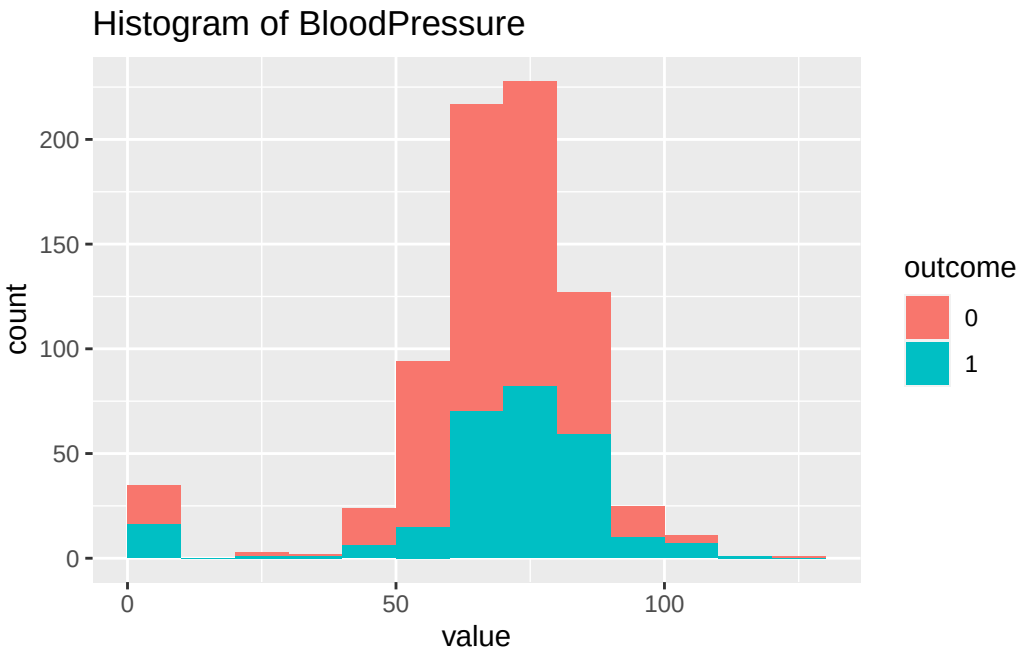
```
[[1]]
```


Histogram of Pregnancies
count
value
outcome
0
1
0
100
200
300
0
5
10
15

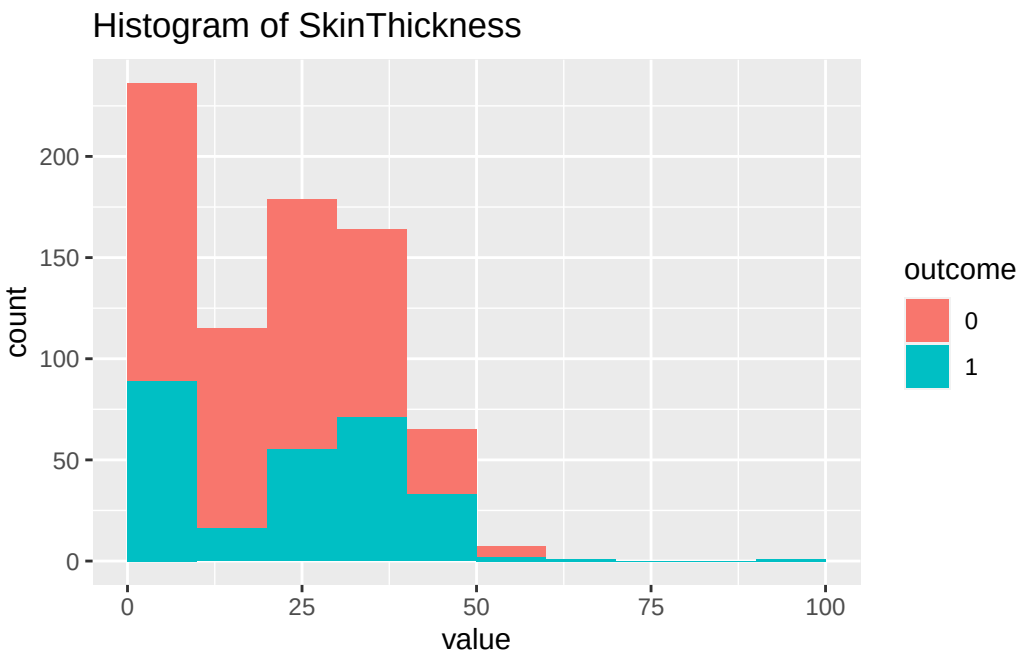
```
[[2]]
```


Histogram of Glucose
count
value
outcome
0
1
0
50
100
150
200
0
50
100
150
200

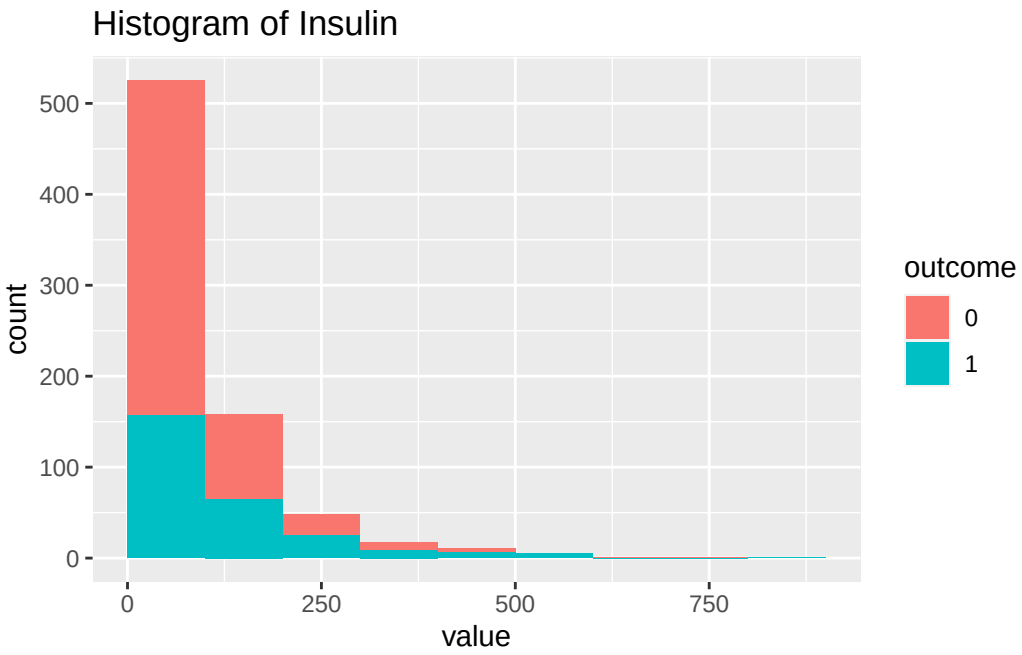
[[3]]


Histogram of BloodPressure
count
200
150
100
50
0
0
50
100
value
outcome
0
1

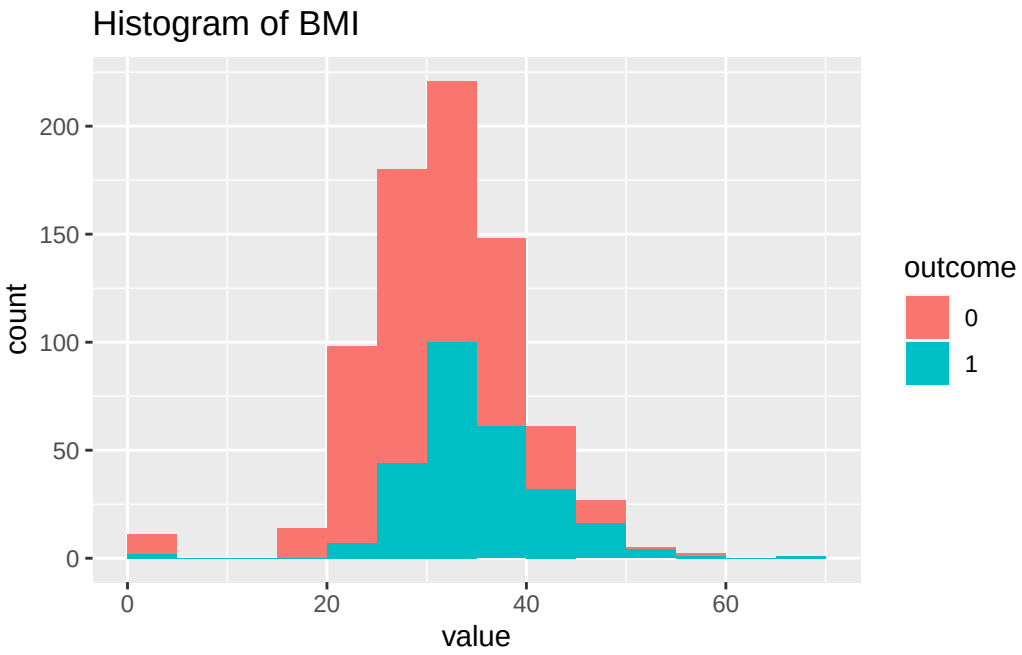
[[4]]


Histogram of SkinThickness
count
value
outcome
0
1
0
50
100
150
200
0
25
50
75
100

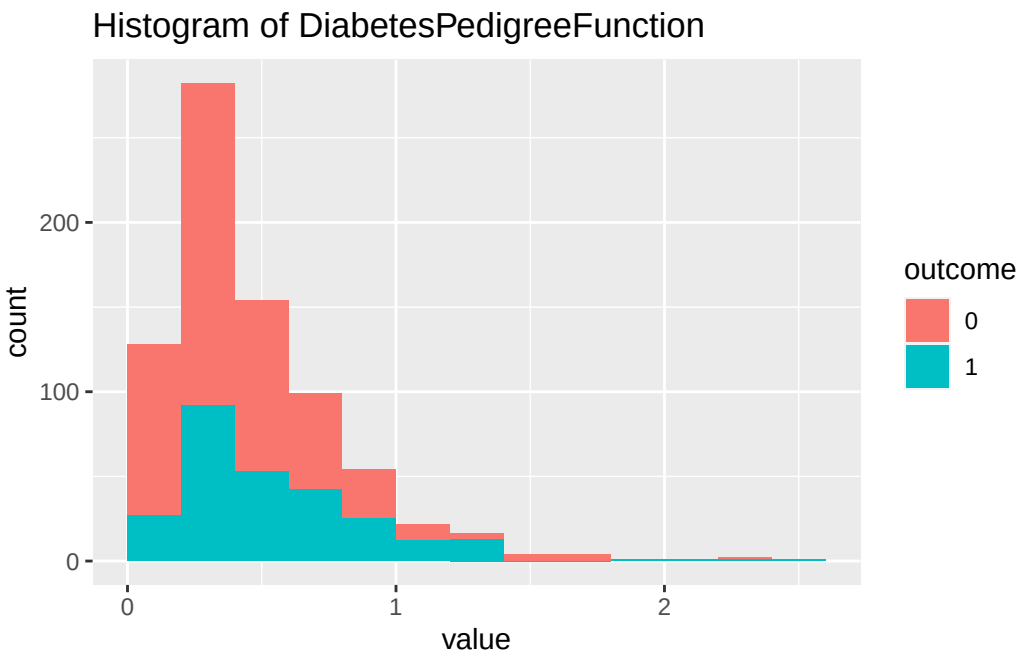
```
[[5]]
```


Histogram of Insulin
count
value
outcome
0
1
0
100
200
300
400
500
0
250
500
750

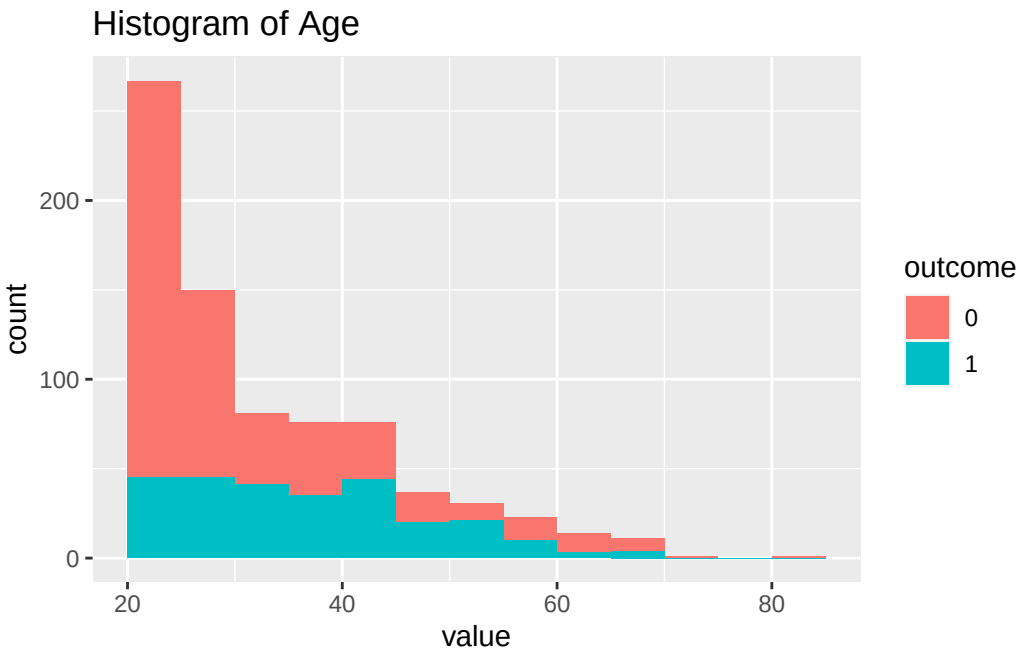
[[6]]


Histogram of BMI
count
200
150
100
50
0
0
20
40
60
value
outcome
0
1

[[7]]


Histogram of DiabetesPedigreeFunction
count
200
100
0
0
1
2
value
outcome
0
1

```
[[8]]
```

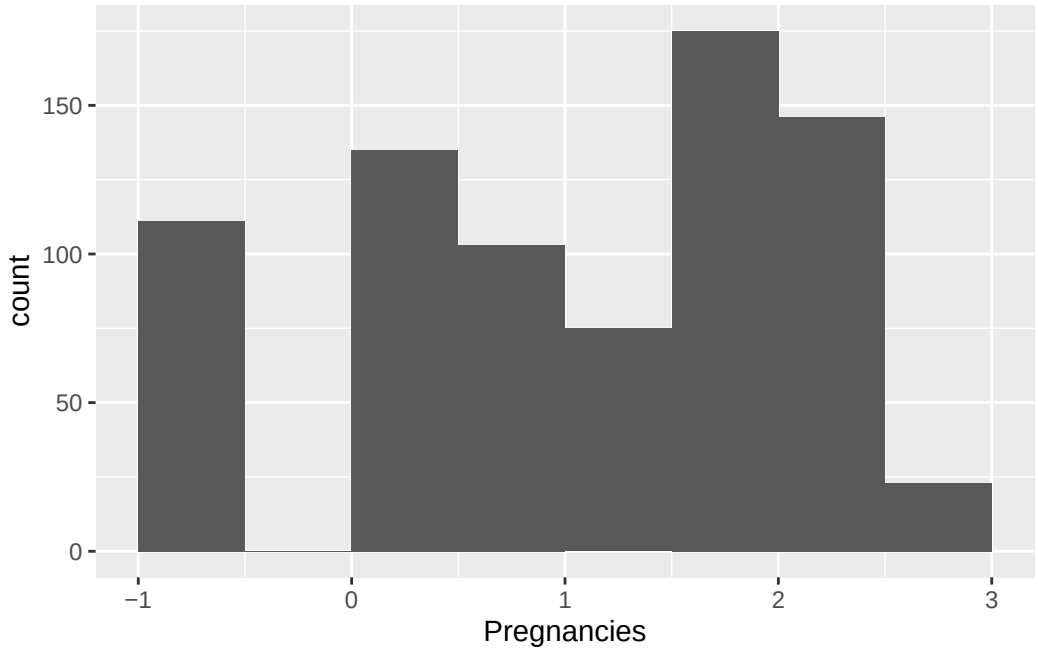

Histogram of Age
count
200
100
0
20
40
60
80
value
outcome
0
1

Curioso, los valores la insulina, han cambiado por la transformación en valor mas no la distribución, vamos a hacer unos arrelgos...

Al parecer la preñanza esta ligada a una esgala logaritmica de 2 Esto es otra cosa...

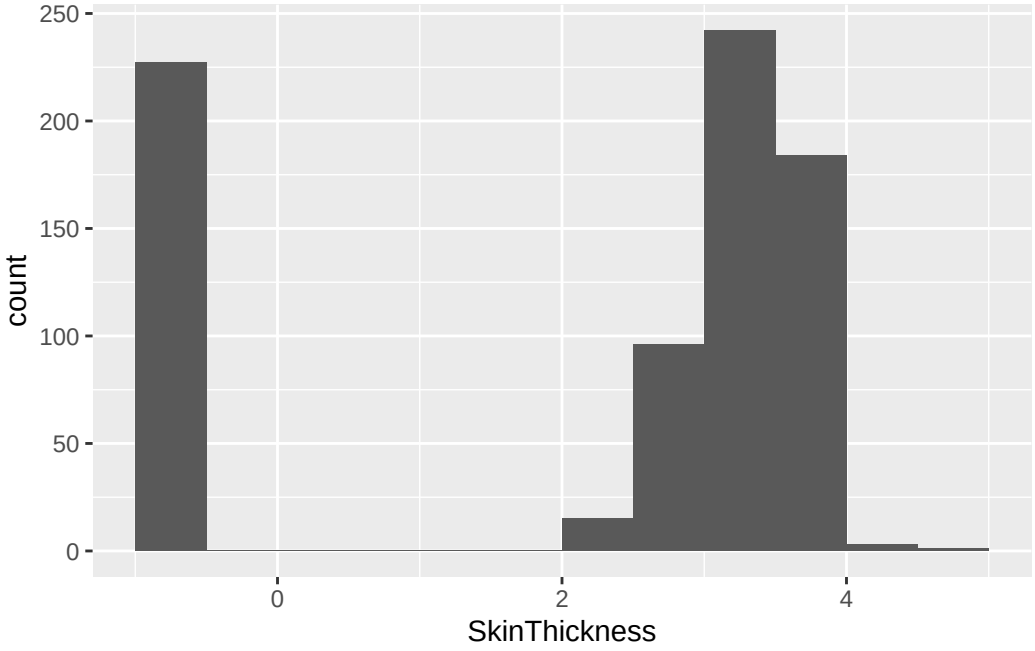
- Función **`read.csv()`** lee un archivo CSV. Los datos leídos se asignan al objeto **`datos`**.
- La línea **`datos$Outcome <- as.factor(datos$Outcome)`**, convierte la columna "Outcome" en una variable categórica, utilizando la función **`as.factor()`**. .
- La línea **`datos$Pregnancies <- log(datos$Pregnancies+0.5)`** realiza una transformación logarítmica en la columna "Pregnancies" de **`datos`**. Se suma 0.5 antes de aplicar el logaritmo para evitar tomar el logaritmo de valores cero o negativos.
- Función **`ggplot()`** para crear un gráfico. Los **`datos`** se pasan como el argumento principal, y se especifica **`Pregnancies`** como el eje x del gráfico utilizando **`aes(Pregnancies)`**.
- Se agrega una capa de histograma al gráfico utilizando **`geom_histogram()`**. El argumento **`breaks`** calcula los intervalos de clase del histograma para la columna "Pregnancies" utilizando la función **`hist()`** con **`plot = F`** (es decir, sin trazar el histograma).

```
datos <- read.csv("./datos/diabetes.csv")
datos$Outcome <- as.factor(datos$Outcome)
datos$Pregnancies  <- log(datos$Pregnancies+0.5)
ggplot(datos,aes(Pregnancies))+geom_histogram(breaks = hist(datos$Pregnancies,plot=F)$brea
```



Realizaremos lo mismo con la grosura de la piel

```
datos <- read.csv("./datos/diabetes.csv")
datos$Outcome <- as.factor(datos$Outcome)
datos$SkinThickness  <- log(datos$SkinThickness+0.5)
ggplot(datos,aes(SkinThickness))+geom_histogram(breaks = hist(datos$SkinThickness,plot=F)$
```

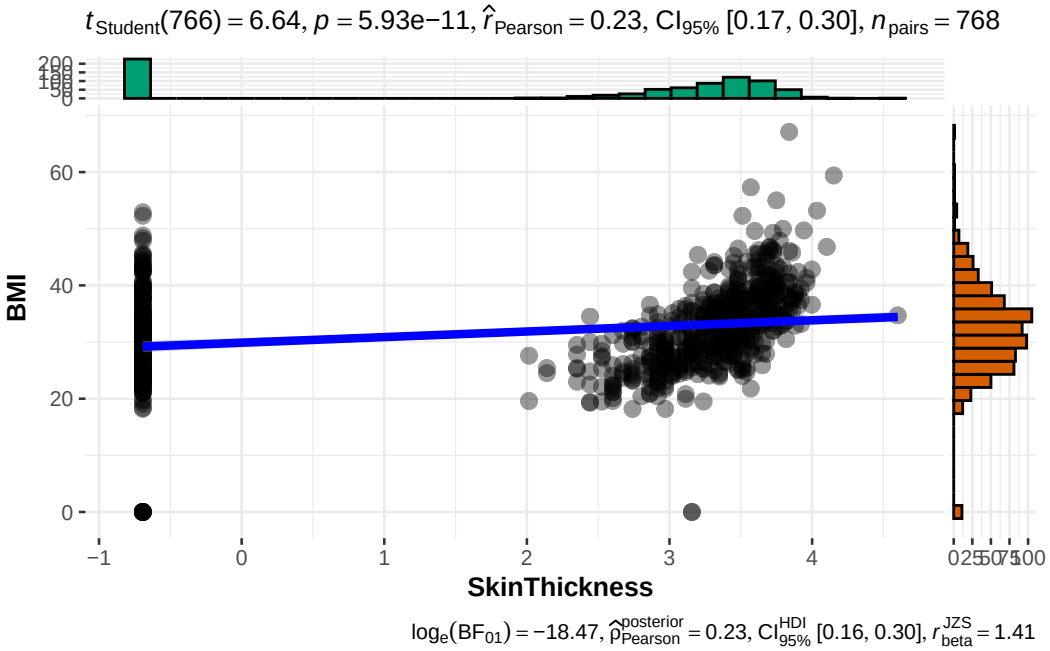


Tenemos algo raro, lo más posible sea por la obesidad...

- La función **ggscatterstats()** se utiliza para generar un gráfico de dispersión (scatter plot) con estadísticas adicionales relacionadas con las variables "SkinThickness" y "BMI" en el conjunto de datos **datos**.

```
ggscatterstats(datos,SkinThickness,BMI)
```

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



Curioso ! al parecer los datos tienen valores nulos, los cuales solo están en las otras variables que no sean pregnancies. Vamos a quitarlos...

- **`datos <- read.csv("./datos/diabetes.csv")`**:carga el archivo CSV.
- **`datos[,-c(1,9)]`**: selecciona todas las columnas excepto (**`-`**)la primera y la novena. El operador **`[,]`** se utiliza para subconjuntar filas y columnas.
- La función **`apply()`** para iterar. El argumento **`2`** indica que se debe aplicar la función columna por columna. **`function(x)`** se utiliza para verificar si cada valor **`x`** es igual a cero. Si es igual a cero, se reemplaza con **`NA`** (valor perdido); de lo contrario, se mantiene el valor original **`x`**.
- Luego, se convierte la columna "Outcome" en un factor utilizando la función **`as.factor()`**.

```
datos <- read.csv("./datos/diabetes.csv")
datos[,-c(1,9)] <- apply(datos[,-c(1,9)],2,function(x) ifelse(x==0,NA,x))

datos$Outcome <- as.factor(datos$Outcome)
```

Vamos a quitar estos valores

- **complete.cases** filtra el objeto "datos" para eliminar las filas que contienen valores (**NA**) en alguna columna.

```
datos <- datos[complete.cases(datos),]
```

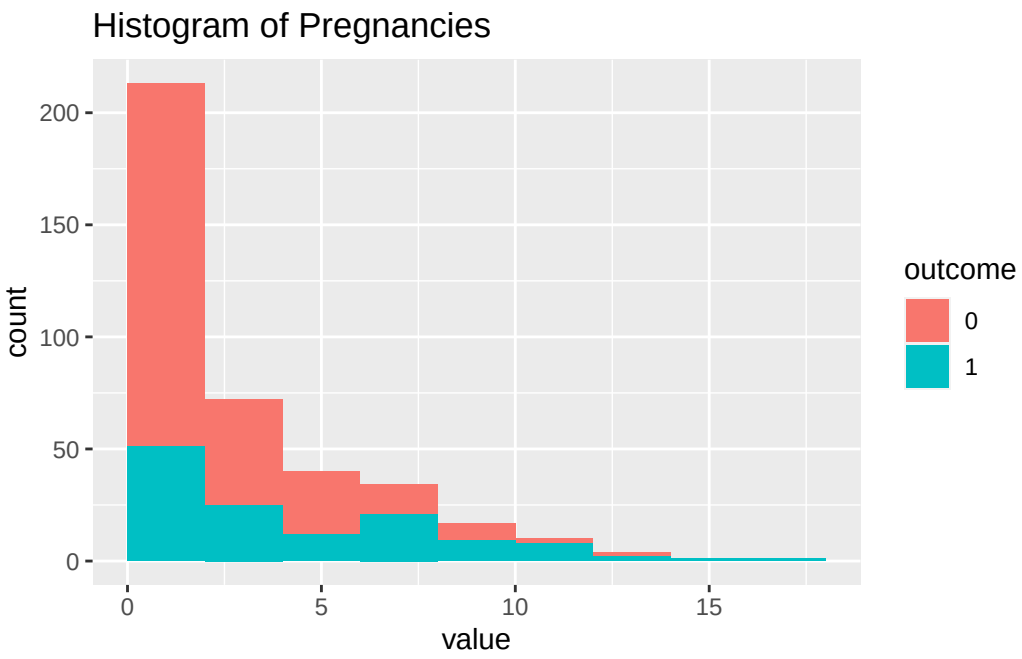
Se redujo el data set a 392 observaciones...

```
table(datos$Outcome)
```

```
  0   1
262 130
```

```
l.plots <- vector("list",length = ncol(datos)-1)
n1 <- ncol(datos) -1
for(j in 1:n1){

  h <-hist(datos[,j],plot = F)
  datos.tmp <- data.frame(value=datos[,j],outcome=datos$Outcome)
  p1 <- ggplot(datos.tmp,aes(value,fill=outcome))+geom_histogram(breaks=h$breaks) + ggtitl

  l.plots[[j]] <- p1
}
l.plots
```

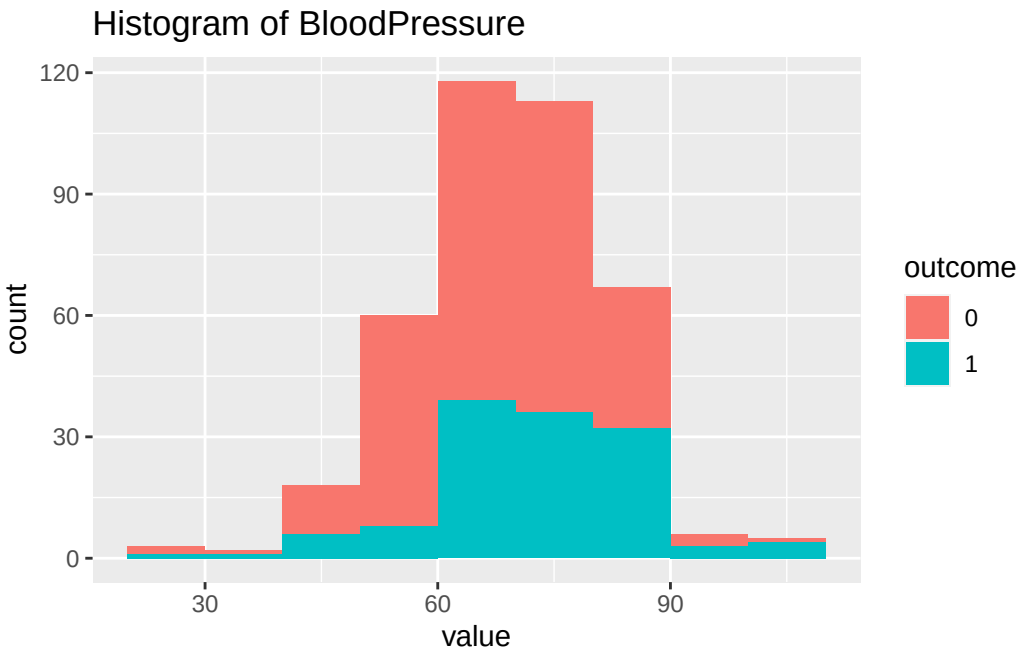
```
[[1]]
```


Histogram of Pregnancies
count
value
outcome
0
1
0
50
100
150
200
0
5
10
15

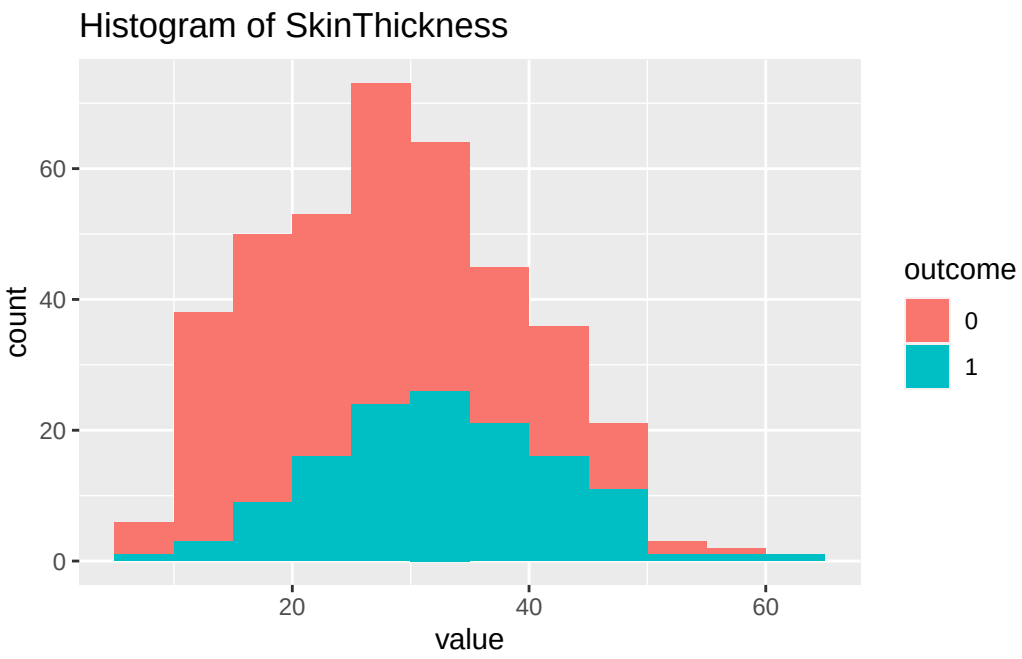
```
[[2]]
```


Histogram of Glucose
count
value
outcome
0
1
0
25
50
75
100
50
100
150
200

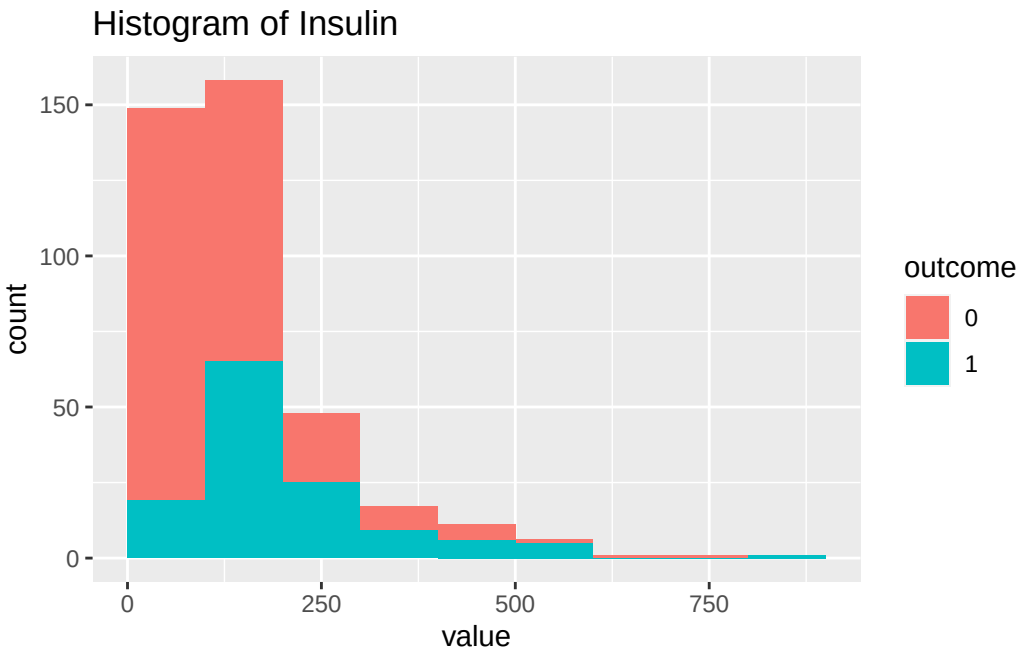
```
[[3]]
```


Histogram of BloodPressure
120
90
60
30
0
count
30
60
90
value
outcome
0
1

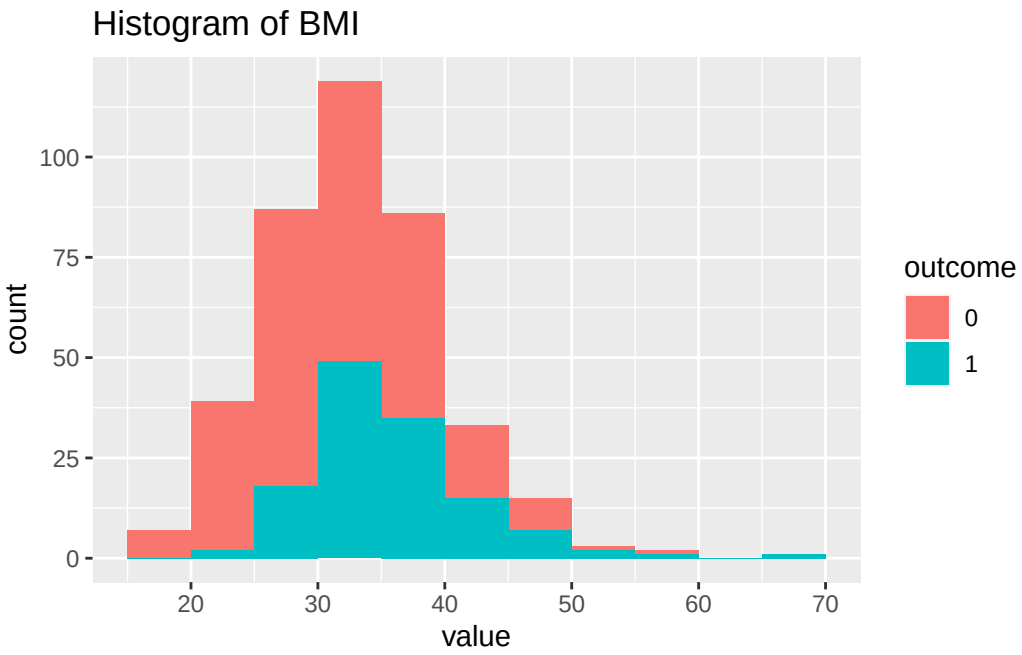
```
[[4]]
```


Histogram of SkinThickness
count
value
outcome
0
1
0
20
40
60
0
20
40
60

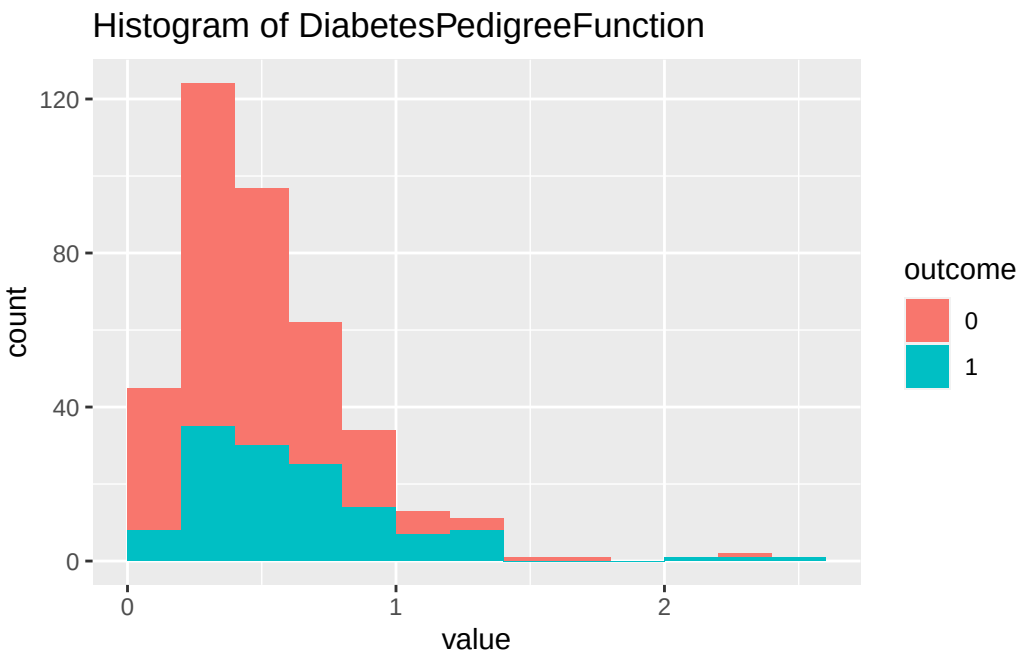
```
[[5]]
```


Histogram of Insulin
count
value
outcome
0
1
0
50
100
150
0
250
500
750

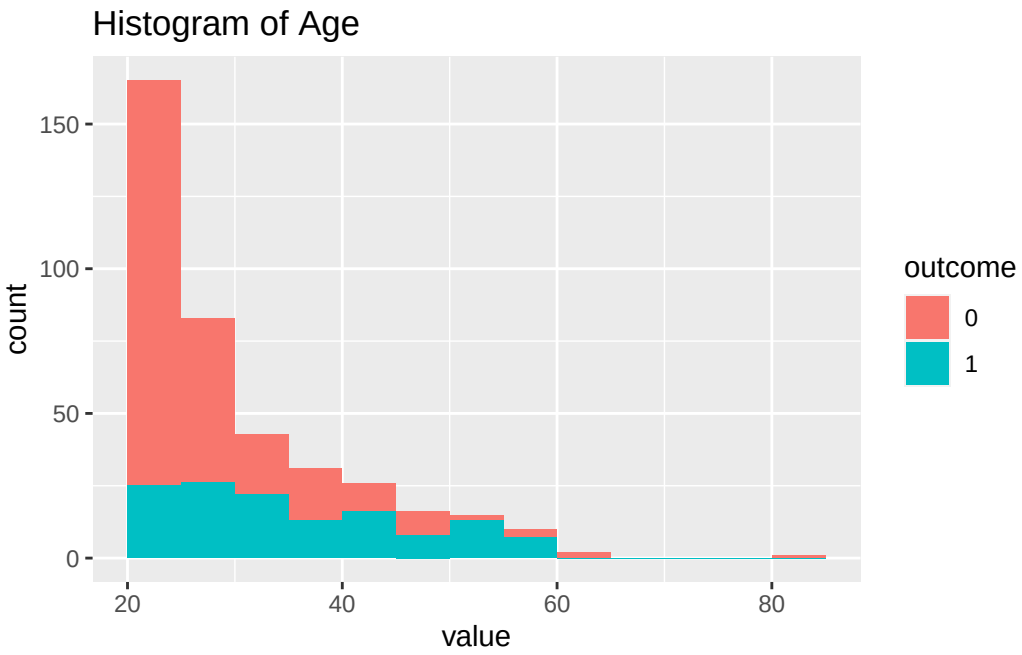
```
[[6]]
```


Histogram of BMI
count
100
75
50
25
0
20
30
40
50
60
70
value
outcome
0
1

```
[[7]]
```


Histogram of DiabetesPedigreeFunction
count
value
outcome
0
1
120
80
40
0
0
1
2

```
[[8]]
```

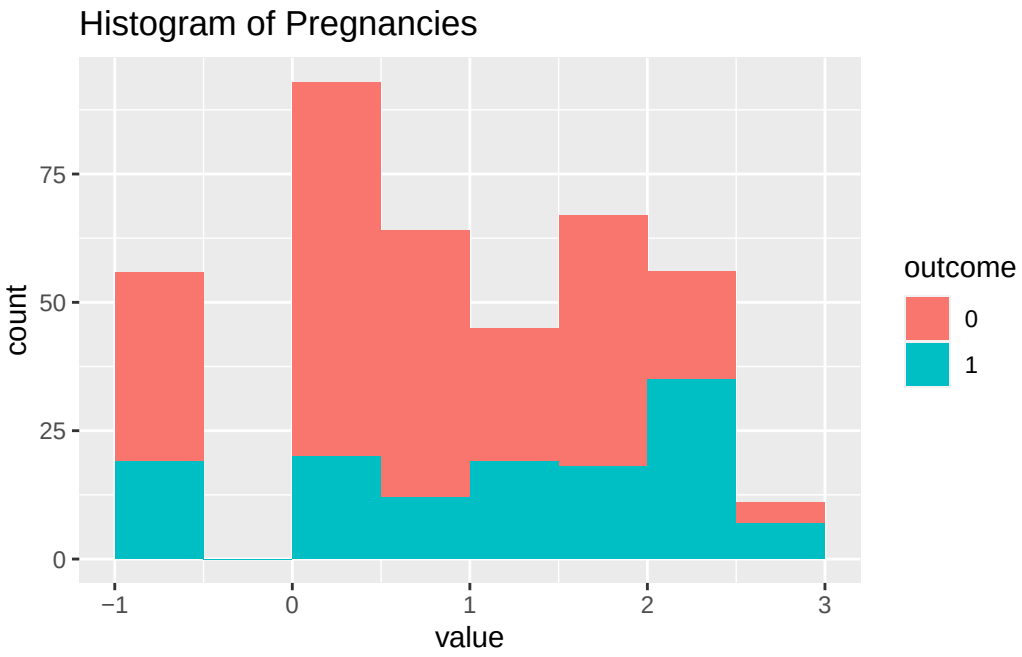

Histogram of Age
count
value
outcome
0
1
150
100
50
0
20
40
60
80

Ahora si podemos realizar las transformaciones.

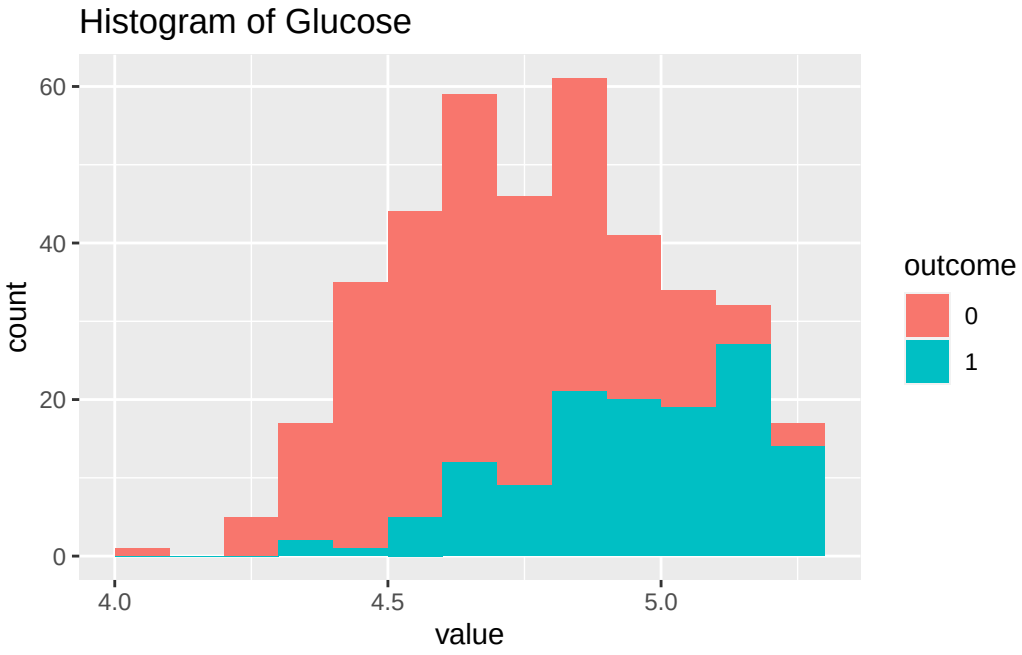
- **`log(datos$Insulin)`** realiza la transformación logarítmica de la variable `Insulin`. La función **`log()`** se utiliza para calcular el logaritmo natural de un valor. Al aplicar esta transformación, se reemplaza el valor original de `Insulin` con su logaritmo natural correspondiente.
- **`sqrt(datos$SkinThickness)`** aplica una transformación de raíz cuadrada a la columna "SkinThickness", la cual puede ser útil para tratar de reducir la asimetría en la distribución de los datos.

```
datos <- read.csv("./datos/diabetes.csv")
datos[,-c(1,9)] <- apply(datos[,-c(1,9)],2,function(x) ifelse(x==0,NA,x))
datos <- datos[complete.cases(datos),]

datos$Outcome <- as.factor(datos$Outcome)
datos$Insulin <- log(datos$Insulin)
datos$Pregnancies <- log(datos$Pregnancies+0.5)
datos$DiabetesPedigreeFunction <- log(datos$DiabetesPedigreeFunction)

datos$SkinThickness <- sqrt((datos$SkinThickness))
datos$Glucose <- log(datos$Glucose)
datos$Age <-log2(datos$Age)
l.plots <- vector("list",length = ncol(datos)-1)
n1 <- ncol(datos) -1
for(j in 1:n1){

  h <-hist(datos[,j],plot = F)
  datos.tmp <- data.frame(value=datos[,j],outcome=datos$Outcome)
  p1 <- ggplot(datos.tmp,aes(value,fill=outcome))+geom_histogram(breaks=h$breaks) + ggtitl

  l.plots[[j]] <- p1
}
l.plots
```

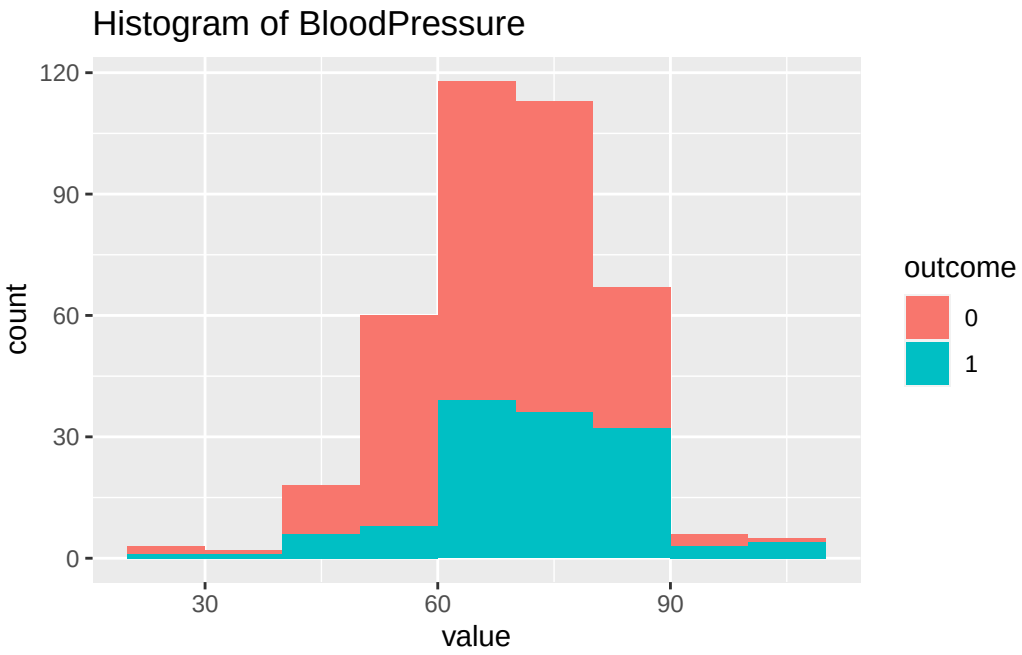
```
[[1]]
```


Histogram of Pregnancies
count
75
50
25
0
−1
0
1
2
3
value
outcome
0
1

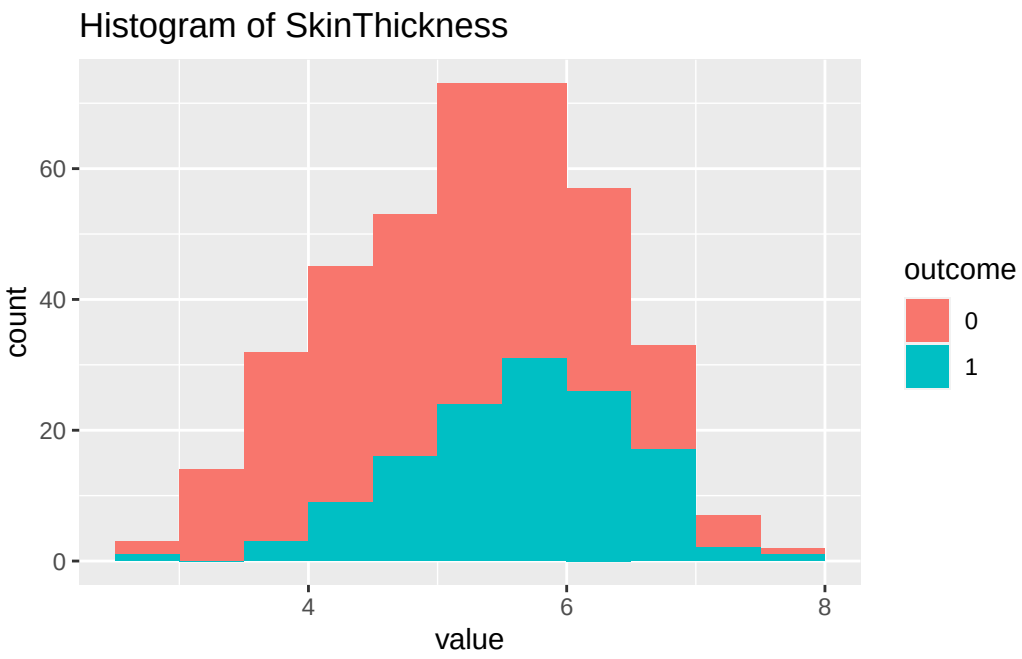
```
[[2]]
```


Histogram of Glucose
count
60
40
20
0
4.0
4.5
5.0
value
outcome
0
1

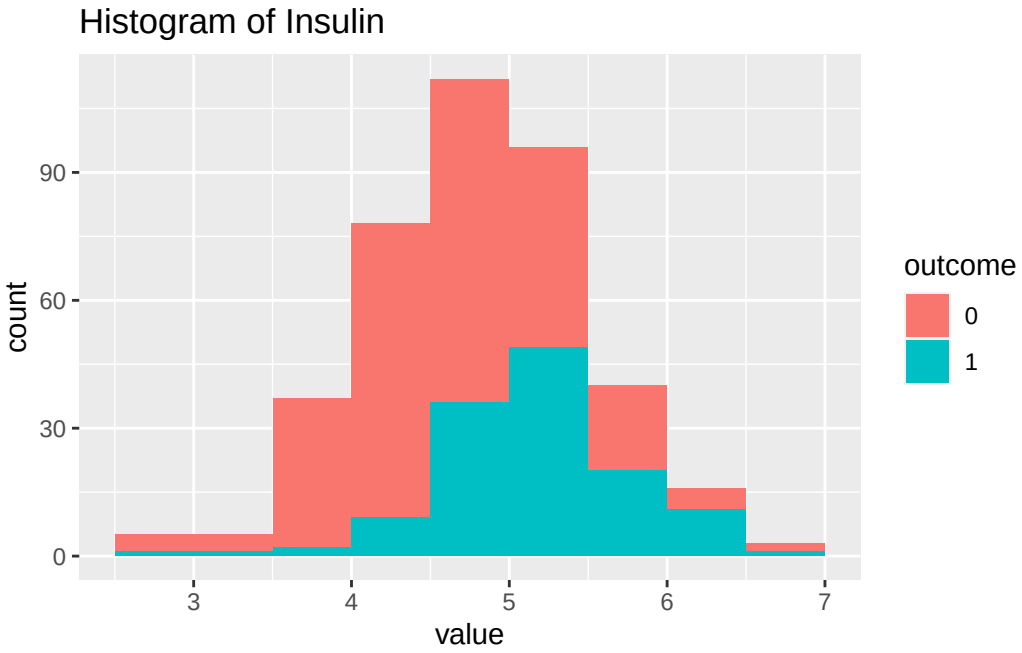
```
[[3]]
```


Histogram of BloodPressure
120
90
60
30
0
count
30
60
90
value
outcome
0
1

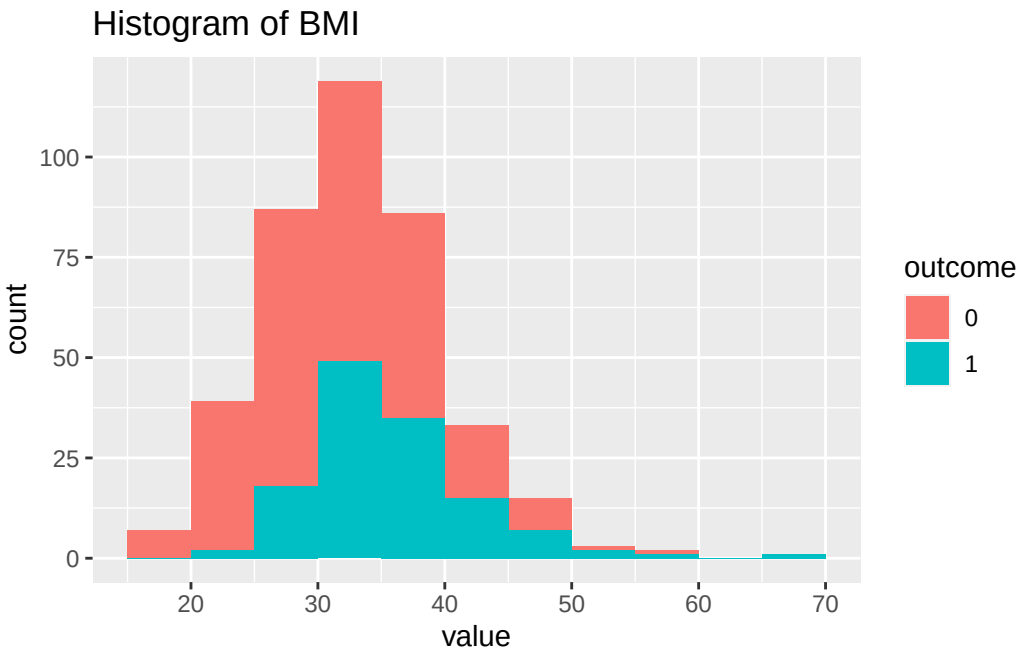
```
[[4]]
```


Histogram of SkinThickness
count
value
outcome
0
1
0
20
40
60
4
6
8

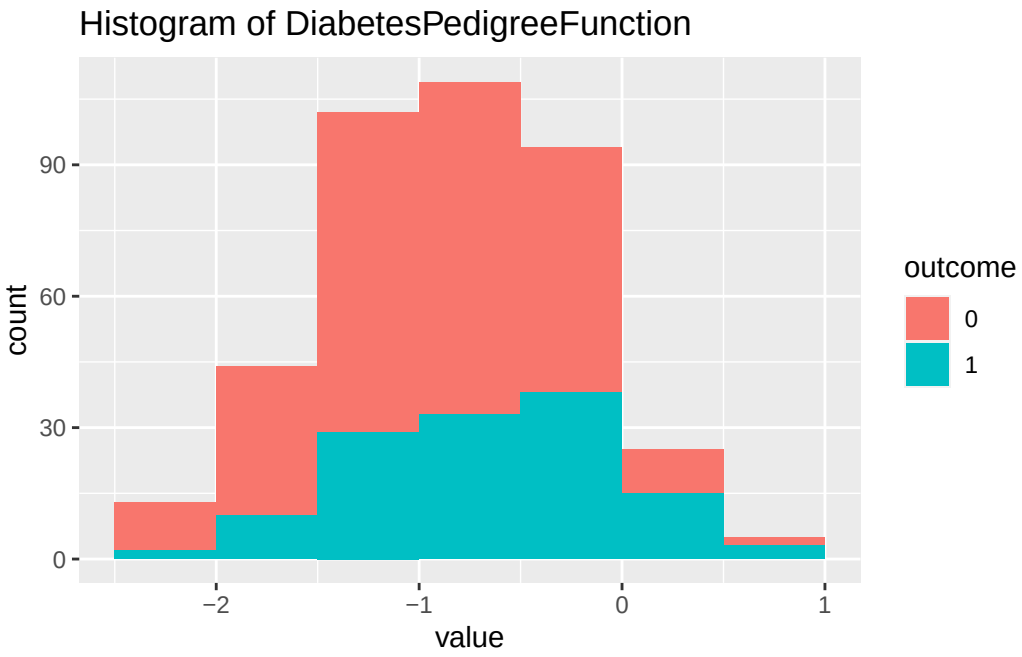
```
[[5]]
```


Histogram of Insulin
count
value
outcome
0
1
0
30
60
90
3
4
5
6
7

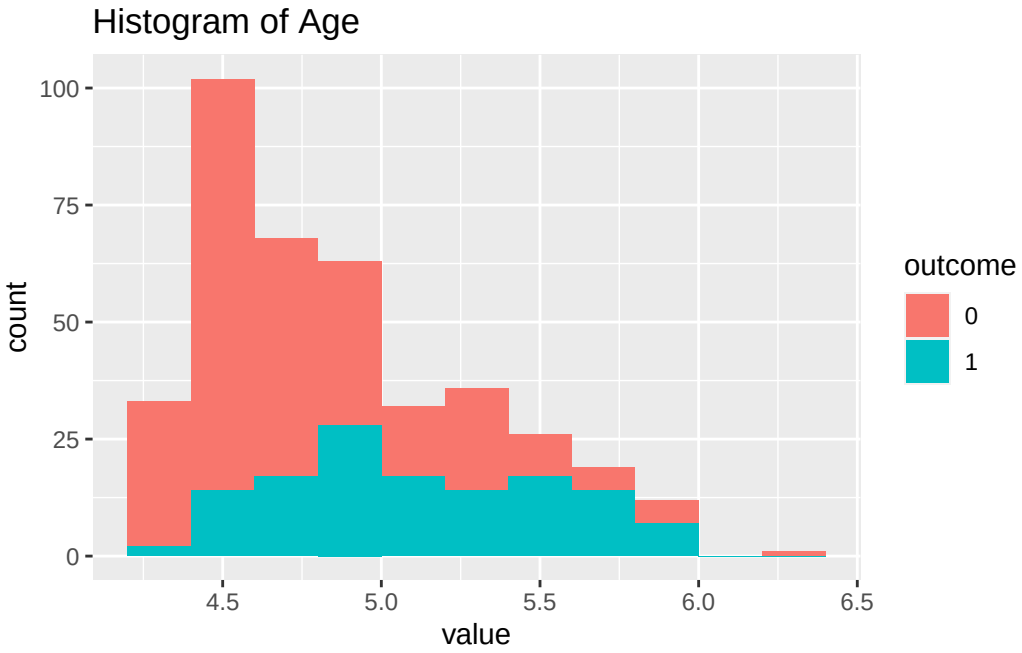
[[6]]


Histogram of BMI
count
100
75
50
25
0
20
30
40
50
60
70
value
outcome
0
1

[[7]]


Histogram of DiabetesPedigreeFunction
count
90
60
30
0
−2
−1
0
1
value
outcome
0
1

```
[[8]]
```


Histogram of Age
count
100
75
50
25
0
4.5
5.0
5.5
6.0
6.5
value
outcome
0
1

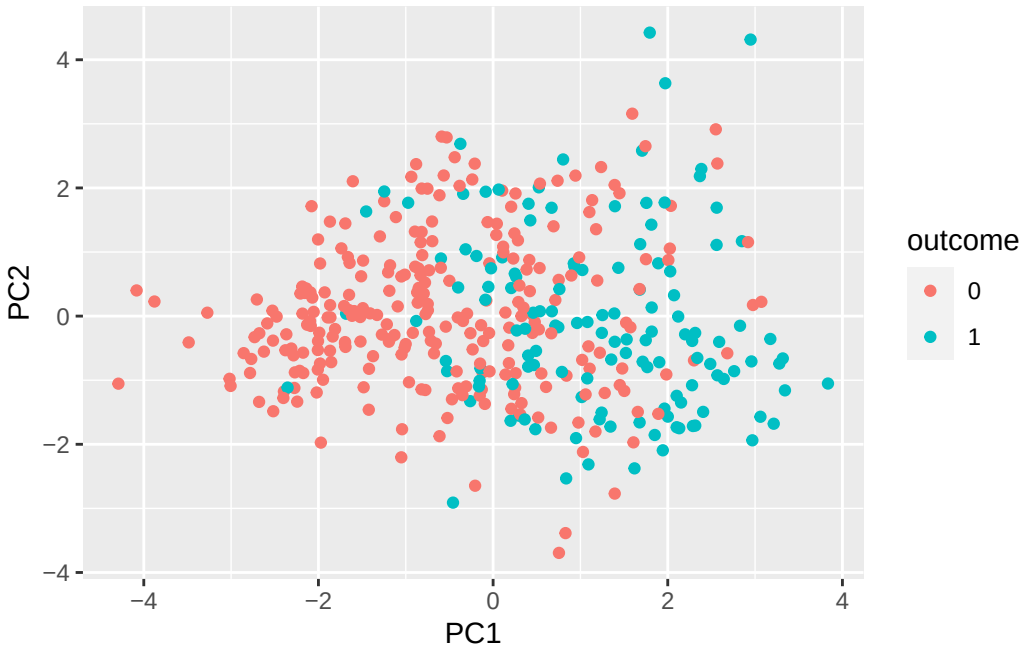
Con las anteriores transformaciones vamos a realizar el PCA de nuevo.

- La función **bind_cols()** se utiliza para unir columnas de diferentes conjuntos de datos en una sola estructura de datos.

```
summary(datos)
```

```
 Pregnancies          Glucose      BloodPressure    SkinThickness
Min.   :-0.6931   Min.   :4.025   Min.   : 24.00   Min.   :2.646
1st Qu.: 0.4055   1st Qu.:4.595   1st Qu.: 62.00   1st Qu.:4.583
Median : 0.9163   Median :4.779   Median : 70.00   Median :5.385
Mean   : 0.9590   Mean   :4.778   Mean   : 70.66   Mean   :5.305
3rd Qu.: 1.7047   3rd Qu.:4.963   3rd Qu.: 78.00   3rd Qu.:6.083
Max.   : 2.8622   Max.   :5.288   Max.   :110.00   Max.   :7.937
   Insulin           BMI        DiabetesPedigreeFunction      Age
Min.   :2.639   Min.   :18.20   Min.   :-2.4651          Min.   :4.392
1st Qu.:4.341   1st Qu.:28.40   1st Qu.:-1.3103          1st Qu.:4.524
Median :4.832   Median :33.20   Median :-0.7996          Median :4.755
Mean   :4.813   Mean   :33.09   Mean   :-0.8391          Mean   :4.882
3rd Qu.:5.247   3rd Qu.:37.10   3rd Qu.:-0.3754          3rd Qu.:5.170
Max.   :6.741   Max.   :67.10   Max.   : 0.8838          Max.   :6.340
Outcome
0:262
1:130
```

```
pcx <- prcomp(datos[,1:n1],scale. = T) ## escalamos por la variablidad de los datos

plotpca <- bind_cols(pcx$x,outcome=datos$Outcome)
ggplot(plotpca,aes(PC1,PC2,color=outcome))+geom_point()
```



Ahora vamos a realizar las pruebas de medianas.

- **`scale(datos[,1:n1])`**: selecciona una submatriz de **`datos`** que contiene las columnas desde la columna 1 hasta la columna **`n1`**. La función **`scale`** se aplica a esta submatriz para estandarizar las variables en cada columna.
- Se realiza una regresión lineal simple utilizando **`lm(x~datos$Outcome)`**. Esto ajusta un modelo lineal donde **`x`** es la variable de respuesta y **`datos$Outcome`** es la variable predictora.
- El primer **`apply`** aplica la función definida en el punto anterior a la submatriz estandarizada. La opción **`2`** indica que la función se aplicará a cada columna de la submatriz.
- El segundo **`apply`** aplica la prueba de Shapiro-Wilk (**`shapiro.test`**) a cada columna de la matriz de residuos. La prueba de normalidad se utiliza para evaluar si una muestra proviene de una distribución normal.

```
p.norm <- apply(apply(scale(datos[,1:n1]),
                2,
                function(x) summary(lm(x~datos$Outcome))$residuals),
      2,
      shapiro.test)
```

```
p.norm
```

```
$Pregnancies

    Shapiro-Wilk normality test

data:  newX[, i]
W = 0.95146, p-value = 4.684e-10


$Glucose

    Shapiro-Wilk normality test

data:  newX[, i]
W = 0.9958, p-value = 0.3813


$BloodPressure

    Shapiro-Wilk normality test

data:  newX[, i]
W = 0.99011, p-value = 0.009686


$SkinThickness

    Shapiro-Wilk normality test

data:  newX[, i]
W = 0.99384, p-value = 0.1123


$Insulin

    Shapiro-Wilk normality test

data:  newX[, i]
W = 0.99054, p-value = 0.0128
```

```
$BMI

    Shapiro-Wilk normality test

data:  newX[, i]
W = 0.97122, p-value = 5.374e-07


$DiabetesPedigreeFunction

    Shapiro-Wilk normality test

data:  newX[, i]
W = 0.99456, p-value = 0.1796


$Age

    Shapiro-Wilk normality test

data:  newX[, i]
W = 0.93053, p-value = 1.561e-12
```

Hemos conseguido la normalidad en solo dos variables, si fueran mas procederiamos con t test pero como no es asi, con test de Wilcoxon.

- La prueba de Wilcoxon **`wilcox.test`** es no paramétrica, utilizada para determinar si hay diferencias significativas entre dos grupos independientes.
- **`$p.value`**: Extraer el valor de p resultante de la prueba.

```r
p.norm <- apply(scale(datos[,1:n1]),
                2,
                function(x) wilcox.test(x~datos$Outcome)$p.value)
```

Observamos que en una primera instancia ahora todas tienen diferencias significativas, esto tenemos que corregir.

- **`p.adjust`** realiza ajustes de p-valores. Recibe dos argumentos principales: los p-valores que se desean ajustar y el método de ajuste a aplicar.

- **"BH"**: Se refiere al método de Benjamini-Hochberg (procedimiento de control de la tasa de falsos descubrimientos (FDR)). Este método es utilizado para ajustar p-valores en pruebas múltiples.

```
p.adj <- p.adjust(p.norm,"BH")
```

Todas siguen siendo significativas, ahora vamos a ver cuales aumentan o disminyuen respecto las otras.

- **split** divide el conjunto de datos en subconjuntos basados en la variable **Outcome**, creando una lista donde cada elemento contiene un subconjunto de datos correspondiente a cada nivel.
- **lapply()** se encarga de aplicar la función definida a cada elemento de **datos.split**, lo que resulta en una lista de medianas calculadas para cada subconjunto de datos.
- **data.frame** generará un data frame con dos columnas. La columna "medianas" contendrá los resultados de **datos.median**, y la columna "p.values" contendrá los p-valores ajustados almacenados en **p.adj**.

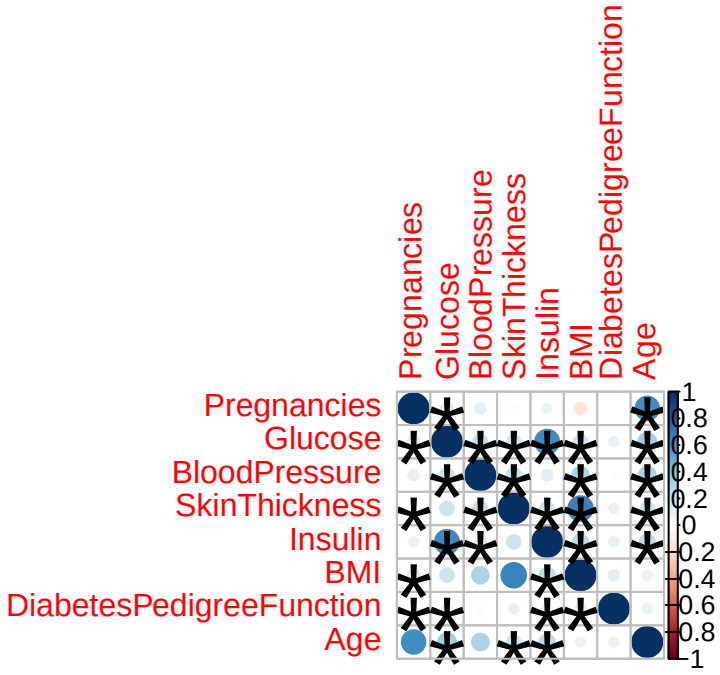
```
datos.split <- split(datos,datos$Outcome)

datos.median <- lapply(datos.split, function(x) apply(x[,-ncol(x)],2,median))


toplot <- data.frame(medianas=Reduce("-",datos.median)
,p.values=p.adj)

toplot
```

```
                           medianas     p.values
Pregnancies              -0.3364722 8.957407e-05
Glucose                  -0.2957935 4.902429e-22
BloodPressure            -4.0000000 8.957407e-05
SkinThickness            -0.5484102 4.309442e-07
Insulin                  -0.4788534 3.241934e-13
BMI                      -3.3500000 2.574728e-07
DiabetesPedigreeFunction -0.2779529 8.957407e-05
Age                      -0.4005379 1.577456e-14
```

Ahora Todos los valores son significativos respecto a la obesidad.

- **psych::corr.test()** realiza pruebas de correlación en los datos. Toma como entrada una matriz o dataframe y calcula varios coeficientes de correlación, como el coeficiente de correlación de Pearson y el coeficiente de correlación de Spearman, junto con sus respectivos valores de p-valor.
- **corrplot** permite representar gráficamente los coeficientes de correlación entre variables en forma de un gráfico de correlación.

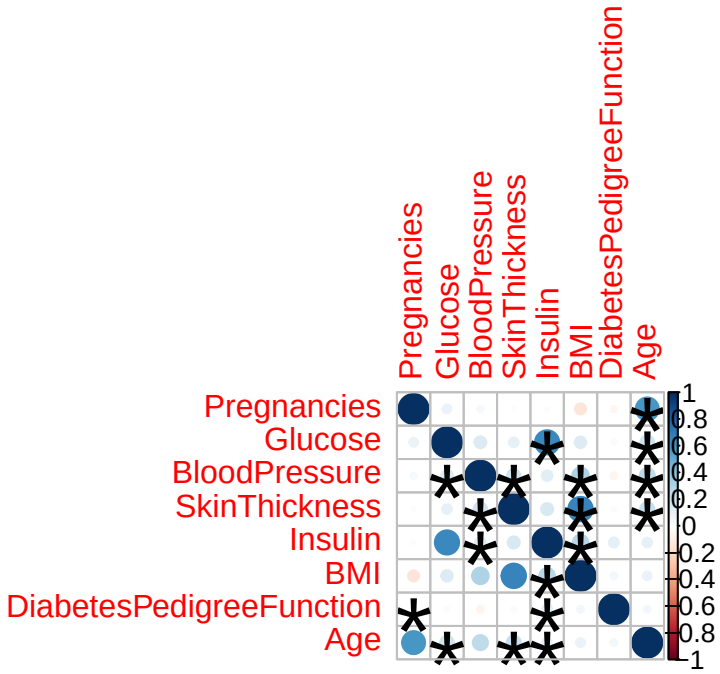
```
obj.cor <- psych::corr.test(datos[,1:n1])
p.values <- obj.cor$p
p.values[upper.tri(p.values)] <- obj.cor$p.adj
p.values[lower.tri(p.values)] <- obj.cor$p.adj
diag(p.values) <- 1
corrplot::corrplot(corr = obj.cor$r,p.mat = p.values,sig.level = 0.05,insig = "label_sig")
```



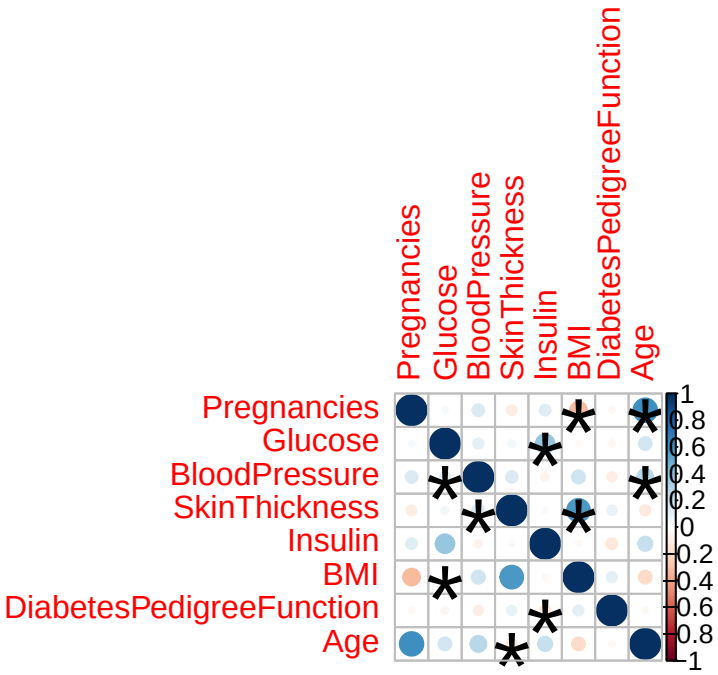
También podemos observar como cambian las relaciones segun la diabetes.

```
obj.cor <- psych::corr.test(datos[datos$Outcome==0,1:n1])
p.values <- obj.cor$p
p.values[upper.tri(p.values)] <- obj.cor$p.adj
p.values[lower.tri(p.values)] <- obj.cor$p.adj
diag(p.values) <- 1
```

```
corrplot::corrplot(corr = obj.cor$r,p.mat = p.values,sig.level = 0.05,insig = "label_sig")
```




```
obj.cor <- psych::corr.test(datos[datos$Outcome==1,1:n1])
p.values <- obj.cor$p
p.values[upper.tri(p.values)] <- obj.cor$p.adj
p.values[lower.tri(p.values)] <- obj.cor$p.adj
diag(p.values) <- 1
corrplot::corrplot(corr = obj.cor$r,p.mat = p.values,sig.level = 0.05,insig = "label_sig")
```



Es decir, existen correlaciones únicas de la obesidad y no obesidad, y existen otras correlaciones que son debidas a otros factores.

Particion de datos

- Se utiliza **`as.data.frame`** para convertir la matriz resultante en un data frame y se asigna nuevamente a las mismas columnas de **`datos`**.
- **`levels(datos$Outcome)`** establece los niveles de la variable categórica **`Outcome`** en "D" y "N".
- Luego divide el conjunto de datos en conjuntos de entrenamiento (**`dat.train`**) y prueba (**`dat.test`**) utilizando un muestreo aleatorio del 70% de las filas para entrenamiento.

```
datos[,1:n1] <- as.data.frame(scale(datos[,-ncol(datos)]))
levels(datos$Outcome) <- c("D","N")
train <- sample(nrow(datos),size = nrow(datos)*0.7)

dat.train <- datos[train,]
dat.test <- datos[-train,]
```

Modelado

- Se realiza un modelo de regresión logística **`Outcome ~ .`**, especifica que la variable "Outcome" es la variable de respuesta y todas las demás variables en el dataframe "dat.train" se utilizan como variables predictoras. El argumento **`family = "binomial"`** se utiliza para indicar que se trata de un modelo de regresión logística para datos binarios.
- Se calcula una matriz **`confusionMatrix`** de confusión para evaluar el rendimiento de las predicciones en comparación con los valores reales.

```
datos[,1:n1] <- as.data.frame(scale(datos[,-ncol(datos)]))

glm.mod <- glm(Outcome ~.,data=dat.train,family = "binomial")

prediccion <- as.factor(ifelse(predict(glm.mod,dat.test,type="response")>=0.5,"N","D"))

caret::confusionMatrix(prediccion,dat.test$Outcome)
```

```
Confusion Matrix and Statistics

          Reference
Prediction  D  N
         D 68 17
         N  5 28

               Accuracy : 0.8136
                 95% CI : (0.7314, 0.8793)
    No Information Rate : 0.6186
    P-Value [Acc > NIR] : 3.829e-06

                  Kappa : 0.5836

 Mcnemar's Test P-Value : 0.01902

            Sensitivity : 0.9315
            Specificity : 0.6222
         Pos Pred Value : 0.8000
         Neg Pred Value : 0.8485
             Prevalence : 0.6186
         Detection Rate : 0.5763
   Detection Prevalence : 0.7203
      Balanced Accuracy : 0.7769
```

```
      'Positive' Class : D
```

RIDGE

- **expand.grid** se utiliza para crear un data frame que representa todas las combinaciones posibles de los valores proporcionados como argumentos.
- Se utiliza el método de regularización **glmnet**. **Lambda** varía de 0 a 1 en incrementos de 0.001. Estos valores se combinan en todas las posibles combinaciones para la sintonización del modelo.
- Se utiliza el método de validación cruzada repetida **repeatedcv** con 10 pliegues y 3 repeticiones, es decir, el modelo se entrenará y evaluará en 10 subconjuntos de datos diferentes, repitiendo el proceso 3 veces.
- **classProbs = T** solicita las probabilidades de clase en lugar de las etiquetas de clase predichas, lo que puede proporcionar información adicional y permitir un ajuste más fino de las clasificaciones.

```
tuneGrid=expand.grid(
              .alpha=0,
              .lambda=seq(0, 1, by = 0.001))
trainControl <- trainControl(method = "repeatedcv",
                        number = 10,
                        repeats = 3,
                        # prSummary needs calculated class,
                        classProbs = T)

model <- train(Outcome ~ ., data = dat.train, method = "glmnet", trControl = trainControl,
                                        metric="Accuracy"
)

confusionMatrix(predict(model,dat.test[,-ncol(dat.test)]),dat.test$Outcome)
```

```
Confusion Matrix and Statistics

          Reference
Prediction  D  N
         D 69 19
         N  4 26
```

```
               Accuracy : 0.8051
                 95% CI : (0.722, 0.8722)
    No Information Rate : 0.6186
    P-Value [Acc > NIR] : 1.016e-05

                  Kappa : 0.5587

 Mcnemar's Test P-Value : 0.003509

            Sensitivity : 0.9452
            Specificity : 0.5778
         Pos Pred Value : 0.7841
         Neg Pred Value : 0.8667
             Prevalence : 0.6186
         Detection Rate : 0.5847
   Detection Prevalence : 0.7458
      Balanced Accuracy : 0.7615

       'Positive' Class : D
```

LASSO

- Se trabaja con la misma metdología de RIDGE, con la única diferencia que el valor de **alpha** es 1 y ya no 0.

```
tuneGrid=expand.grid(
               .alpha=1,
               .lambda=seq(0, 1, by = 0.0001))
trainControl <- trainControl(method = "repeatedcv",
                         number = 10,
                         repeats = 3,
                         # prSummary needs calculated class,
                         classProbs = T)

model <- train(Outcome ~ ., data = dat.train, method = "glmnet", trControl = trainControl,
                                        metric="Accuracy"
)

confusionMatrix(predict(model,dat.test[,-ncol(dat.test)]),dat.test$Outcome)
```

```
Confusion Matrix and Statistics
```

```
          Reference
Prediction  D  N
         D 68 17
         N  5 28

               Accuracy : 0.8136
                 95% CI : (0.7314, 0.8793)
    No Information Rate : 0.6186
    P-Value [Acc > NIR] : 3.829e-06

                  Kappa : 0.5836

 Mcnemar's Test P-Value : 0.01902

            Sensitivity : 0.9315
            Specificity : 0.6222
         Pos Pred Value : 0.8000
         Neg Pred Value : 0.8485
             Prevalence : 0.6186
         Detection Rate : 0.5763
   Detection Prevalence : 0.7203
      Balanced Accuracy : 0.7769

       'Positive' Class : D
```

NAIVE BAYES

- **laplace = 0** indica que no se debe aplicar el ajuste de Laplace. Este modelo tiene una accuracy del 83% y se establece cómo positivo es decir que hay diabetes.

```
datos[,1:n1] <- as.data.frame(scale(datos[,-ncol(datos)]))
levels(datos$Outcome) <- c("D","N")
train <- sample(nrow(datos),size = nrow(datos)*0.7)

dat.train <- datos[train,]
dat.test <- datos[-train,]
mdl <- naiveBayes(Outcome ~ .,data=dat.train,laplace = 0)
prediccion <-predict(mdl,dat.test[,-ncol(dat.test)])
confusionMatrix(prediccion,dat.test$Outcome)
```

```
Confusion Matrix and Statistics

          Reference
Prediction  D  N
         D 63 13
         N 14 28

               Accuracy : 0.7712
                 95% CI : (0.6848, 0.8435)
    No Information Rate : 0.6525
    P-Value [Acc > NIR] : 0.00363

                  Kappa : 0.4983

 Mcnemar's Test P-Value : 1.00000

            Sensitivity : 0.8182
            Specificity : 0.6829
         Pos Pred Value : 0.8289
         Neg Pred Value : 0.6667
             Prevalence : 0.6525
         Detection Rate : 0.5339
   Detection Prevalence : 0.6441
      Balanced Accuracy : 0.7506

       'Positive' Class : D
```

- Calcula **lambda_use** como el valor de **lambda** más cercano o igual al valor óptimo, encuentra su posición en el vector de **lambda**.
- Crea un data frame (**featsele**) que contiene los coeficientes del modelo en esa posición. Esta operación es comúnmente utilizada en selección de características para identificar las características seleccionadas o relevantes basadas en el valor de **lambda** elegido.

```r
lambda_use <- min(model$finalModel$lambda[model$finalModel$lambda >= model$bestTune$lambda
position <- which(model$finalModel$lambda == lambda_use)
featsele <- data.frame(coef(model$finalModel)[, position])
```

- Esta linea de codigo devuelve los nombres de fila en el data frame **featsele** que corresponden a los coeficientes del modelo que no son iguales a cero. Estos nombres representan las características seleccionadas o relevantes según el criterio utilizado en la selección de características.

```r
rownames(featsele)[featsele$coef.model.finalModel....position.!=0]
```

```
[1] "(Intercept)"              "Pregnancies"
[3] "Glucose"                  "BloodPressure"
[5] "SkinThickness"            "Insulin"
[7] "BMI"                      "DiabetesPedigreeFunction"
[9] "Age"
```

- Se construye un modelo de clasificación de Bayes con la función **naiveBayes**. El modelo se ajusta utilizando los datos de entrenamiento **dat.train**.
- Luego, se utilizan los datos de prueba **dat.test** para realizar predicciones utilizando el modelo **mdl.sel**. La función **predict** se utiliza para generar las predicciones basadas en el modelo. El argumento **dat.test[,-ncol(dat.test)]** se utiliza para seleccionar todas las columnas de **dat.test** excepto la última columna, que es la columna de la variable objetivo.
- La matriz de confusión **confusionMatrix** compara las predicciones (**prediccion**) con los valores reales de la variable objetivo (**dat.test$Outcome**).
- Se puede observar que este modelo tiene una accuracy del 85%.

```r
mdl.sel <-naiveBayes(Outcome ~ Insulin+Glucose+DiabetesPedigreeFunction+Age,data = dat.tra

prediccion <- predict(mdl.sel,dat.test[,-ncol(dat.test)])

confusionMatrix(prediccion,dat.test$Outcome)
```

```
Confusion Matrix and Statistics

          Reference
Prediction  D  N
         D 62 13
         N 15 28

               Accuracy : 0.7627
                 95% CI : (0.6756, 0.8362)
    No Information Rate : 0.6525
    P-Value [Acc > NIR] : 0.006613

                  Kappa : 0.4826
```

```
 Mcnemar's Test P-Value : 0.850107

            Sensitivity : 0.8052
            Specificity : 0.6829
         Pos Pred Value : 0.8267
         Neg Pred Value : 0.6512
             Prevalence : 0.6525
         Detection Rate : 0.5254
   Detection Prevalence : 0.6356
      Balanced Accuracy : 0.7441

       'Positive' Class : D
```

- Se crea una configuración de control para el entrenamiento del modelo utilizando **trainControl**. La opción **method="repeatedcv"** especifica que se utilizará validación cruzada repetida como método de evaluación y se realizarán 3 repeticiones.
- Se ajusta un modelo de clasificación kNN utilizando la función **train**. La fórmula **Outcome ~ .** especifica que la variable objetivo es **Outcome**. El argumento **preProcess = c("center","scale")** indica que se deben centrar y escalar las variables predictoras antes de ajustar el modelo. Se realizarán 50 combinaciones diferentes de parámetros (**tuneLength**) durante la búsqueda de los mejores parámetros del modelo.
- Finalmente, la variable **knnFit** contiene los resultados del ajuste del modelo kNN.

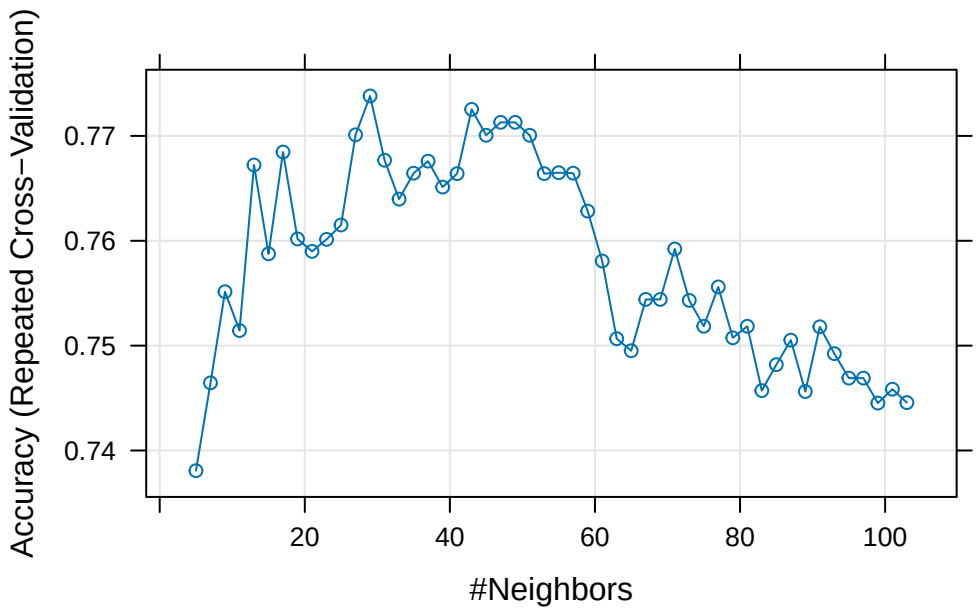
```
library(ISLR)
library(caret)
set.seed(400)
ctrl <- trainControl(method="repeatedcv",repeats = 3) #,classProbs=TRUE,summaryFunction =
knnFit <- train(Outcome ~ ., data = dat.train, method = "knn", trControl = ctrl, preProces

#Output of kNN fit
knnFit
```

```
k-Nearest Neighbors

274 samples
  8 predictor
  2 classes: 'D', 'N'

Pre-processing: centered (8), scaled (8)
Resampling: Cross-Validated (10 fold, repeated 3 times)
```

```
Summary of sample sizes: 246, 247, 246, 246, 247, 247, ...
Resampling results across tuning parameters:

  k    Accuracy   Kappa
    5  0.7380749  0.3795354
    7  0.7464489  0.3977005
    9  0.7551418  0.4156305
   11  0.7514381  0.3959916
   13  0.7672263  0.4263419
   15  0.7587573  0.4046521
   17  0.7684643  0.4198470
   19  0.7601750  0.3979912
   21  0.7589879  0.3922429
   23  0.7601343  0.3910551
   25  0.7615045  0.3957535
   27  0.7701058  0.4170260
   29  0.7738095  0.4212566
   31  0.7676808  0.4071904
   33  0.7639737  0.4018174
   35  0.7664462  0.4091511
   37  0.7675926  0.4135106
   39  0.7651201  0.4062207
   41  0.7664055  0.4099736
   43  0.7725343  0.4238028
   45  0.7700617  0.4162539
   47  0.7712963  0.4187708
   49  0.7712963  0.4159246
   51  0.7700617  0.4090756
   53  0.7664021  0.3998136
   55  0.7664903  0.4017396
   57  0.7664462  0.4025320
   59  0.7628307  0.3903020
   61  0.7580722  0.3736716
   63  0.7506614  0.3545030
   65  0.7495150  0.3492514
   67  0.7544092  0.3581077
   69  0.7544092  0.3564957
   71  0.7592152  0.3663664
   73  0.7543176  0.3527812
   75  0.7518485  0.3417694
   77  0.7555996  0.3506217
   79  0.7507462  0.3366262
   81  0.7518451  0.3348108
```

```
 83  0.7457129  0.3205155
 85  0.7481855  0.3283301
 87  0.7505223  0.3319946
 89  0.7456247  0.3172973
 91  0.7517976  0.3344064
 93  0.7492369  0.3223484
 95  0.7469034  0.3139858
 97  0.7469034  0.3112685
 99  0.7445225  0.3024955
101  0.7458452  0.3013711
103  0.7445665  0.2946654

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was k = 29.
```

```r
plot(knnFit)
```



- El código realiza la predicción utilizando el modelo k-NN entrenado y luego utiliza la matriz de confusión para evaluar la precisión y otros parámetros de rendimiento del modelo.

```r
knnPredict <- predict(knnFit,newdata = dat.test[,-ncol(dat.test)] )
#Get the confusion matrix to see accuracy value and other parameter values
confusionMatrix(knnPredict, dat.test$Outcome )
```

```
Confusion Matrix and Statistics

          Reference
Prediction  D  N
         D 72 21
         N  5 20

               Accuracy : 0.7797
                 95% CI : (0.6941, 0.8507)
    No Information Rate : 0.6525
    P-Value [Acc > NIR] : 0.001906

                  Kappa : 0.4653

 Mcnemar's Test P-Value : 0.003264

            Sensitivity : 0.9351
            Specificity : 0.4878
         Pos Pred Value : 0.7742
         Neg Pred Value : 0.8000
             Prevalence : 0.6525
         Detection Rate : 0.6102
   Detection Prevalence : 0.7881
      Balanced Accuracy : 0.7114

       'Positive' Class : D
```

- La variable **plsda** contiene los resultados del ajuste del modelo PLS-DA.
- Se realizan predicciones utilizando el modelo PLS-DA ajustado. La función **predict** se utiliza para generar las predicciones basadas en el modelo. El argumento **newdata = dat.test[,-ncol(datos)]** especifica los datos de prueba en los cuales se realizarán las predicciones.
- Se calcula la matriz de confusión utilizando la función **confusionMatrix** de **caret**. La matriz de confusión compara las predicciones (**prediccion**) con los valores reales de la variable objetivo (**dat.test$Outcome**).

- El modelo PLS alcanzó una precisión de 0,74, que es la proporción de predicciones correctas. El valor Kappa de 0,3833 indica la concordancia entre las predicciones del modelo y las clases verdaderas. La sensibilidad mide la proporción de casos D reales predichos correctamente, mientras que la especificidad mide la proporción de casos N reales predichos correctamente. En general, el modelo PLS obtuvo un rendimiento moderado, con una sensibilidad destacada (0,87) pero una especificidad inferior (0,48).

```
library(caret)
datos <- read.csv("./datos/diabetes.csv")
datos$Outcome <-as.factor(datos$Outcome)
datos[,1:n1] <- as.data.frame(scale(datos[,-ncol(datos)]))
levels(datos$Outcome) <- c("D","N")
train <- sample(nrow(datos),size = nrow(datos)*0.7)

dat.train <- datos[train,]
dat.test <- datos[-train,]
set.seed(1001)
ctrl<-trainControl(method="repeatedcv",number=10,classProbs = TRUE,summaryFunction = twoCl
plsda<-train(x=dat.train[,-ncol(datos)], # spectral data
             y=dat.train$Outcome, # factor vector
             method="pls", # pls-da algorithm
             tuneLength=10, # number of components
             trControl=ctrl, # ctrl contained cross-validation option
             preProc=c("center","scale"), # the data are centered and scaled
             metric="ROC") # metric is ROC for 2 classes
plsda
```

```
Partial Least Squares

537 samples
  8 predictor
  2 classes: 'D', 'N'

Pre-processing: centered (8), scaled (8)
Resampling: Cross-Validated (10 fold, repeated 1 times)
Summary of sample sizes: 483, 484, 483, 483, 483, 483, ...
Resampling results across tuning parameters:

  ncomp  ROC        Sens       Spec
  1      0.8183485  0.8468067  0.5657895
  2      0.8348713  0.8667227  0.6181579
```

```
  3      0.8346068  0.8814286  0.6023684
  4      0.8342848  0.8756303  0.6076316
  5      0.8338425  0.8784874  0.6023684
  6      0.8336922  0.8784874  0.6023684
  7      0.8336922  0.8784874  0.6023684

ROC was used to select the optimal model using the largest value.
The final value used for the model was ncomp = 2.
```

```r
prediccion <- predict(plsda,newdata = dat.test[,-ncol(datos)])

confusionMatrix(prediccion,dat.test$Outcome)
```

```
Confusion Matrix and Statistics

          Reference
Prediction   D   N
         D 135  40
         N  19  37

               Accuracy : 0.7446
                 95% CI : (0.6833, 0.7995)
    No Information Rate : 0.6667
    P-Value [Acc > NIR] : 0.006419

                  Kappa : 0.3833

 Mcnemar's Test P-Value : 0.009220

            Sensitivity : 0.8766
            Specificity : 0.4805
         Pos Pred Value : 0.7714
         Neg Pred Value : 0.6607
             Prevalence : 0.6667
         Detection Rate : 0.5844
   Detection Prevalence : 0.7576
      Balanced Accuracy : 0.6786

       'Positive' Class : D
```

Si tuneamos lambda

- Se crea un vector llamado "lambda" que contiene valores secuenciales de 0 a 50 con incrementos de 0.1.
- Carga y prepara los datos de diabetes, divide los datos en conjuntos de entrenamiento y prueba, ajusta un modelo de Naive Bayes y genera predicciones utilizando el modelo ajustado. Luego, se evalúa la precisión global del modelo utilizando la matriz de confusión.

```
datos <- read.csv("./datos/diabetes.csv")
datos$Outcome <-as.factor(datos$Outcome)
levels(datos$Outcome) <- c("D","N")
train <- sample(nrow(datos),size = nrow(datos)*0.7)

dat.train <- datos[train,]
dat.test <- datos[-train,]
lambda <- seq(0,50,0.1)

  modelo <- naiveBayes(dat.train[,-ncol(datos)],dat.train$Outcome)

  predicciones <- predict(modelo,dat.test[,-ncol(datos)])

confusionMatrix(predicciones,dat.test$Outcome)$overall[1]
```

```
 Accuracy
0.7705628
```

- Se ajusta un modelo de clasificación PLS-DA utilizando la función **`train`**. El argumento **`x = dat.train[,c(2,5,7,8)]`** especifica las variables predictoras del conjunto de entrenamiento y selecciona las columnas 2, 5, 7 y 8 como variables predictoras. El argumento **`trControl = ctrl`** especifica la configuración de control definida anteriormente. El argumento **`preProc = c("center","scale")`** indica que se deben centrar y escalar las variables predictoras antes de ajustar el modelo. El argumento **`metric = "ROC"`** indica que la métrica utilizada para evaluar el modelo es el Área bajo la Curva ROC (Receiver Operating Characteristic).
- La función **`predict`** se utiliza para generar las predicciones basadas en el modelo.
- Se calcula la matriz de confusión utilizando la función **`confusionMatrix`**.
- La precisión es de 0.7316, lo que indica que el modelo clasificó correctamente el 73.16% de los casos. La sensibilidad es de 0.8675, lo que indica que el modelo identificó correctamente el 86.75% de los casos de diabetes. La especificidad es de 0.4750, lo que indica que

el modelo identificó correctamente el 47.50% de los casos sin diabetes. El valor predictivo positivo es de 0.7572, lo que indica que el 75.72% de los casos predichos como diabetes son realmente diabetes.

```
datos <- read.csv("./datos/diabetes.csv")
datos$Outcome <-as.factor(datos$Outcome)
datos[,1:n1] <- as.data.frame(scale(datos[,-ncol(datos)]))
levels(datos$Outcome) <- c("D","N")
train <- sample(nrow(datos),size = nrow(datos)*0.7)

dat.train <- datos[train,]
dat.test <- datos[-train,]
library(caret)
set.seed(1001)
ctrl<-trainControl(method="repeatedcv",number=10,classProbs = TRUE,summaryFunction = twoCl
plsda<-train(x=dat.train[,c(2,5,7,8)], # spectral data
             y=dat.train$Outcome, # factor vector
             method="pls", # pls-da algorithm
             tuneLength=10, # number of components
             trControl=ctrl, # ctrl contained cross-validation option
             preProc=c("center","scale"), # the data are centered and scaled
             metric="ROC") # metric is ROC for 2 classes

prediccion <- predict(plsda,dat.test[,c(2,5,7,8)])
confusionMatrix(prediccion,dat.test$Outcome)
```

```
Confusion Matrix and Statistics

          Reference
Prediction   D   N
         D 136  42
         N   9  44

               Accuracy : 0.7792
                 95% CI : (0.7201, 0.831)
    No Information Rate : 0.6277
    P-Value [Acc > NIR] : 5.532e-07

                  Kappa : 0.4876

 Mcnemar's Test P-Value : 7.433e-06
```

```
            Sensitivity : 0.9379
            Specificity : 0.5116
         Pos Pred Value : 0.7640
         Neg Pred Value : 0.8302
             Prevalence : 0.6277
         Detection Rate : 0.5887
   Detection Prevalence : 0.7706
      Balanced Accuracy : 0.7248

       'Positive' Class : D
```

Finalmente podríamos hacer un análisis de la varianza multivariante.

- Se realiza el análisis de varianza multivariado utilizando la función **adonis2** de **vegan**. El argumento **datos[,-ncol(datos)]** especifica las variables predictoras del conjunto de datos, excluyendo la última columna que corresponde a la variable objetivo. El argumento **datos$Outcome** especifica la variable objetivo del conjunto de datos. La fórmula **~** indica que se desea analizar la relación entre las variables predictoras y la variable objetivo. El argumento **method = "euclidean"** indica que se utilizará la distancia euclidiana como medida de disimilitud entre las observaciones.
- Es importante tener en cuenta que el análisis de varianza multivariado es adecuado para datos que presentan una estructura de respuesta multivariada, como en estudios ecológicos o en análisis de composición de especies.

```
library(vegan)
```

```
Loading required package: permute
```

```
This is vegan 2.6-4
```

```
Attaching package: 'vegan'
```

```
The following object is masked from 'package:caret':

    tolerance
```

```
adonis2(datos[,-ncol(datos)] ~datos$Outcome,method = "euclidean")
```

```
Permutation test for adonis under reduced model
Terms added sequentially (first to last)
Permutation: free
Number of permutations: 999

adonis2(formula = datos[, -ncol(datos)] ~ datos$Outcome, method = "euclidean")
               Df SumOfSqs      R2      F Pr(>F)
datos$Outcome   1    357.8 0.05831 47.434  0.001 ***
Residual      766   5778.2 0.94169
Total         767   6136.0 1.00000
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Es decir, como conlusión aunque las variables no pueden detectar la diabetes, siendo variables independientes, si por otro lado las consideramos dependientes de la diabetes.

Es decir, la diabetes es una condición en la que influye en los parámetros, mientras que es menos probable que la diabetes sea la causa de estas alteraciones, con una mejor precisón del 77 por ciento.

Es decir, por un lado tenemos las variables que nos explican solo un 77 porciento de la diabetes, mientras que la condición en sí nos separa más entre la media global.

Se podría investigar más esto. Por ejemplo, se podría hacer una correlación parcial, dada la diabetes, e identificar aquellas variables especificamente relacionadas con esta.