Predicción de la diabetes

Edmond Géraud

Integrantes: María José Bustamante / Nicolás Jadán

Intro

Nota: En este repaso se agregaron comentarios que hacen referencia a lo que se entendió de cada código, aquellos sin comentarios signifcan que esa parte está entendida y no hay dudas.

Este sería un ejemplo de examen El siguiente conjunto de datos, consuste 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

```
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
```

```
library(caret)
```

```
Loading required package: lattice
```

```
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
```

Cargamos los datos

```
datos <- read.csv("./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

```
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

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

Análisis estadístico preliminar

```
dim(datos)
```

```
[1] 768   9
```

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

Histogramas

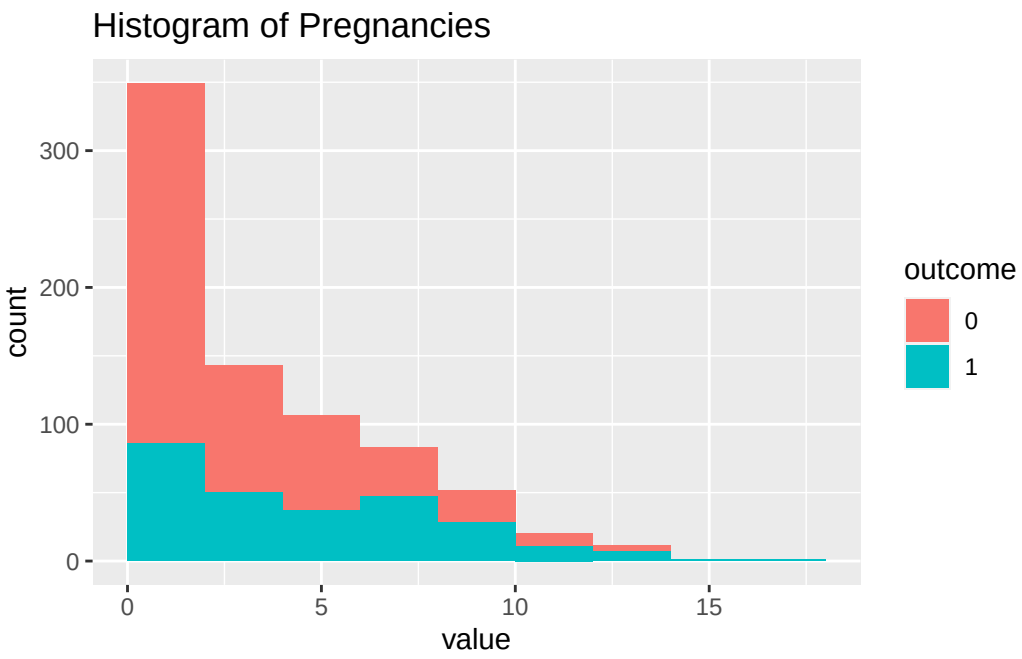
```
l.plots <- vector("list",length = ncol(datos)-1)
n1 <- ncol(datos) -1
for(j in 1:n1){

  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
}
```

COMENTARIO: En este código se crea un objeto llamado "datos.tmp" que contiene dos columnas: "value" y "outcome". La columna "value" contiene los valores de la columna "j" y la columna "outcome" contiene los valores de la columna "Outcome" del archivo "datos".

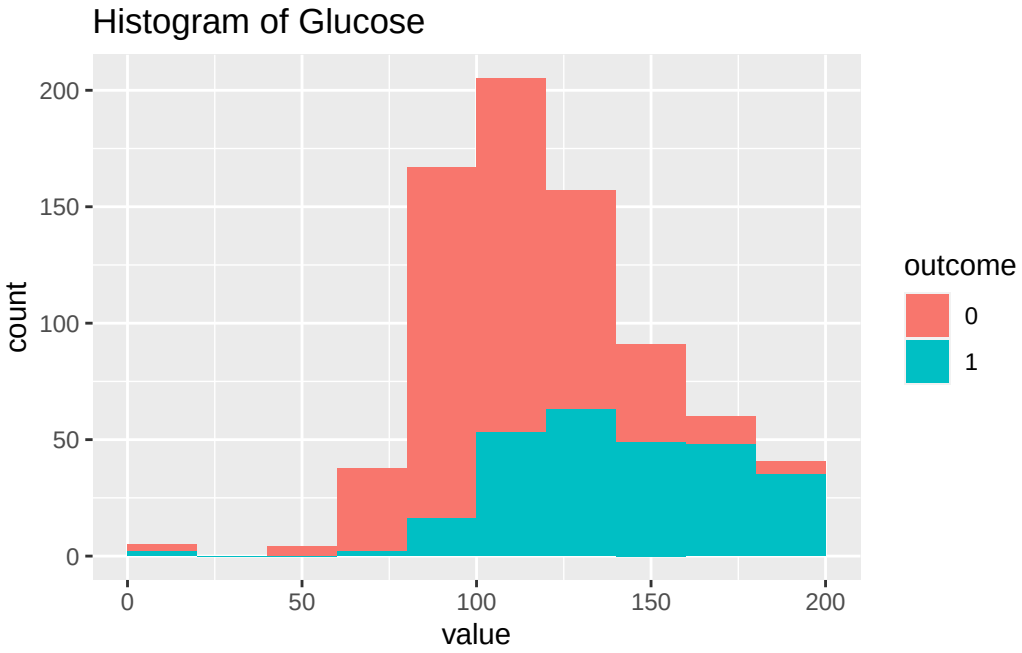
Cuando el bucle termina la lista "l.plots" contiene los gráficos de histograma generados para cada columna de "datos" (excepto la columna "Outcome").

```
l.plots
```

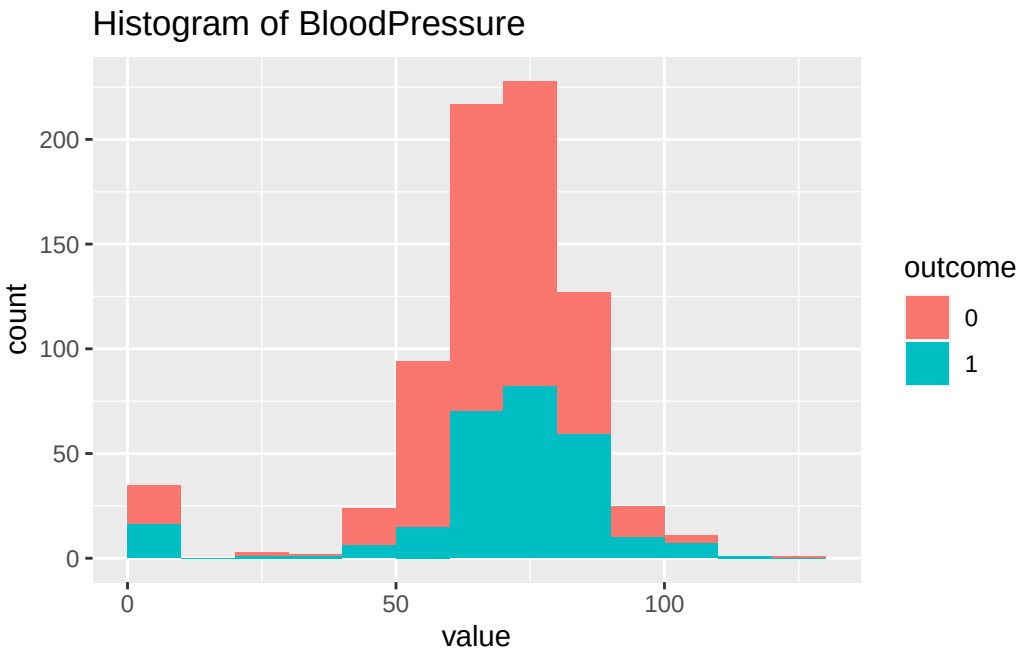
```
[[1]]
```


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

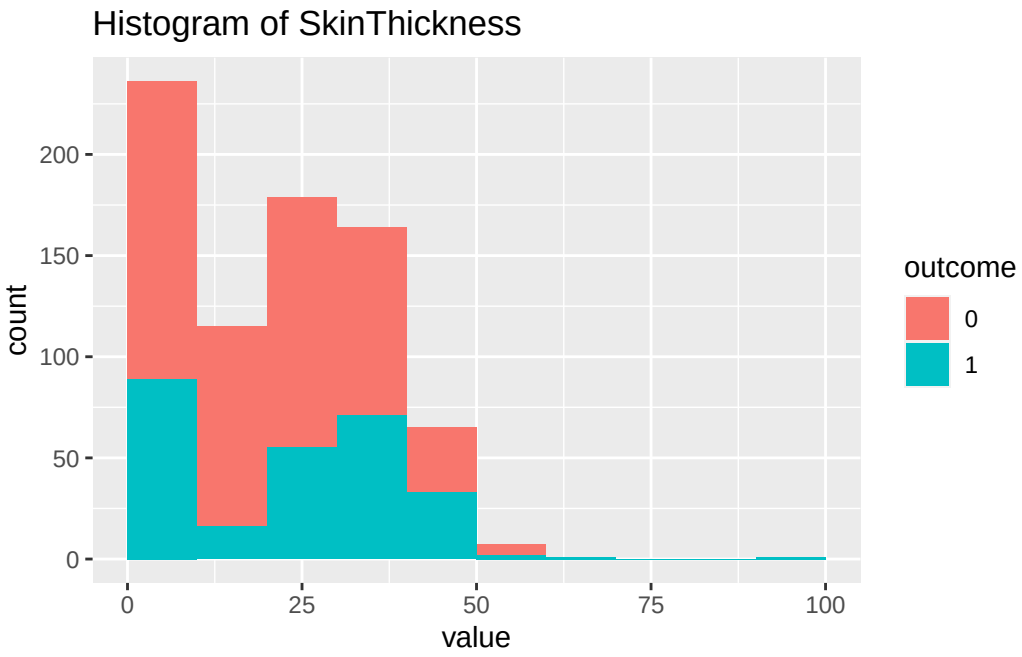
```
[[2]]
```


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

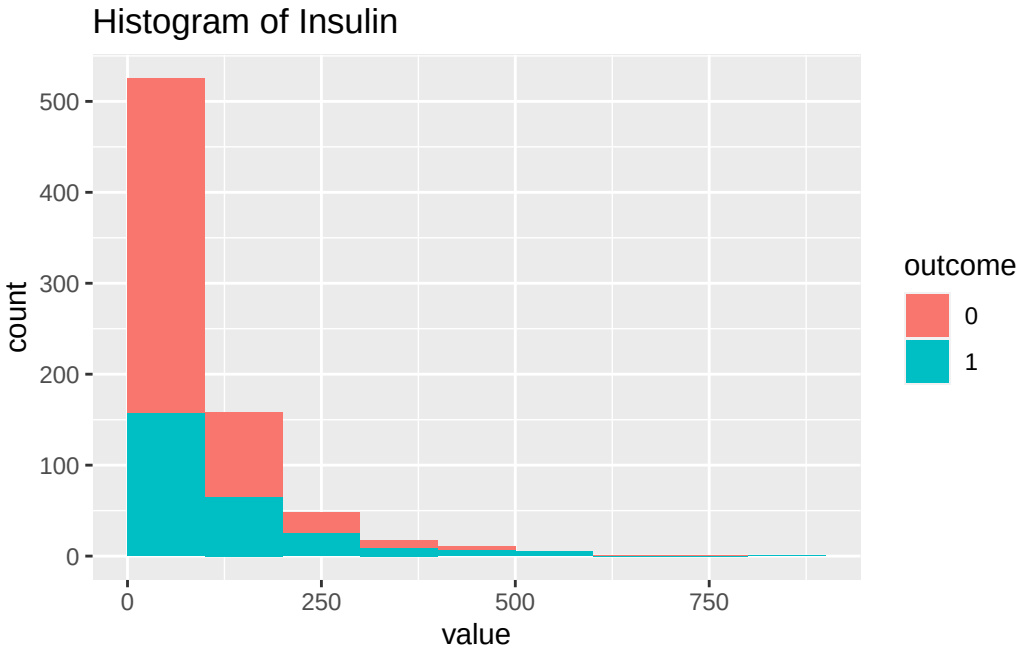
```
[[3]]
```


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

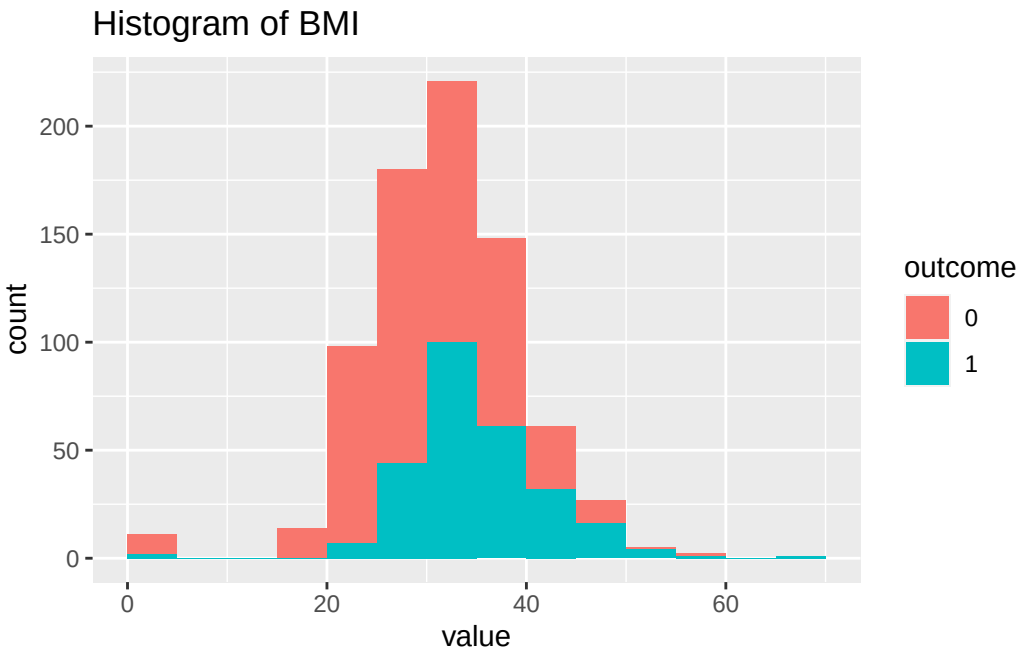
```
[[4]]
```


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

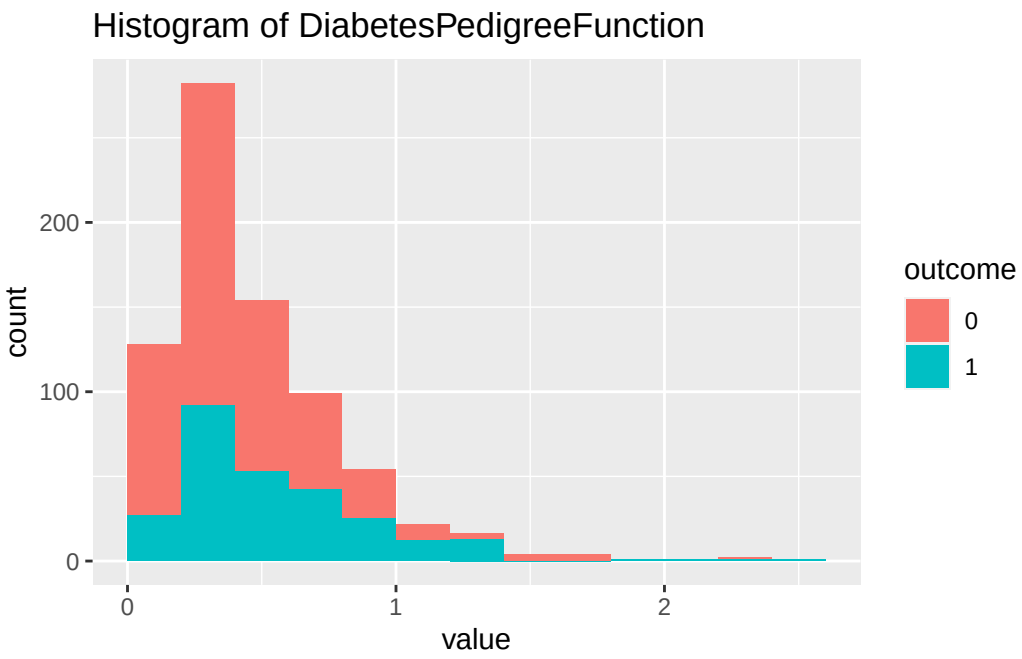
```
[[5]]
```


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

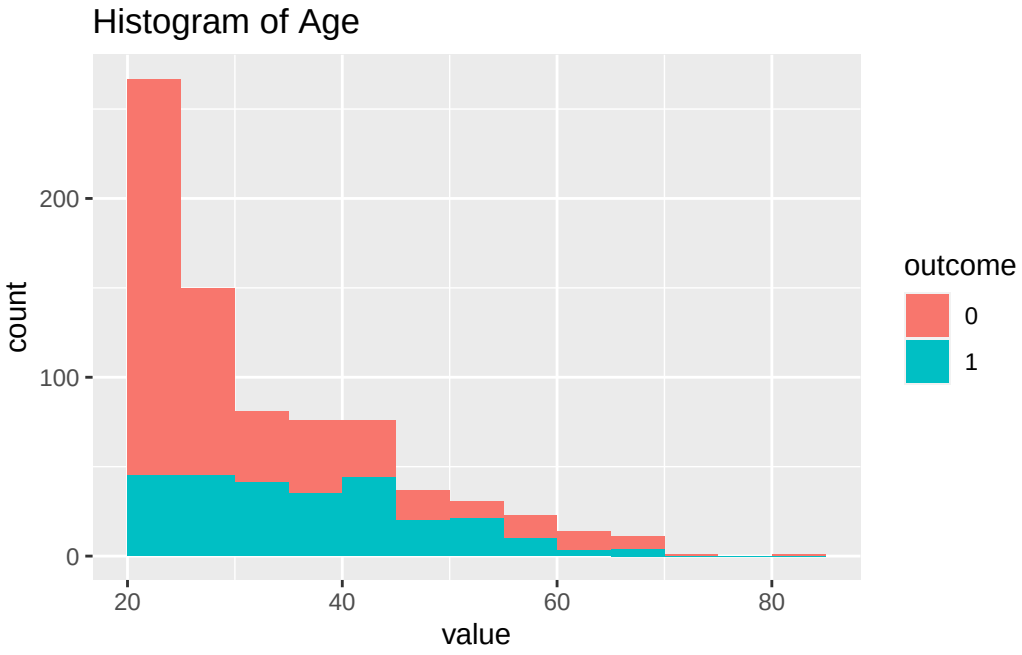
[[6]]


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

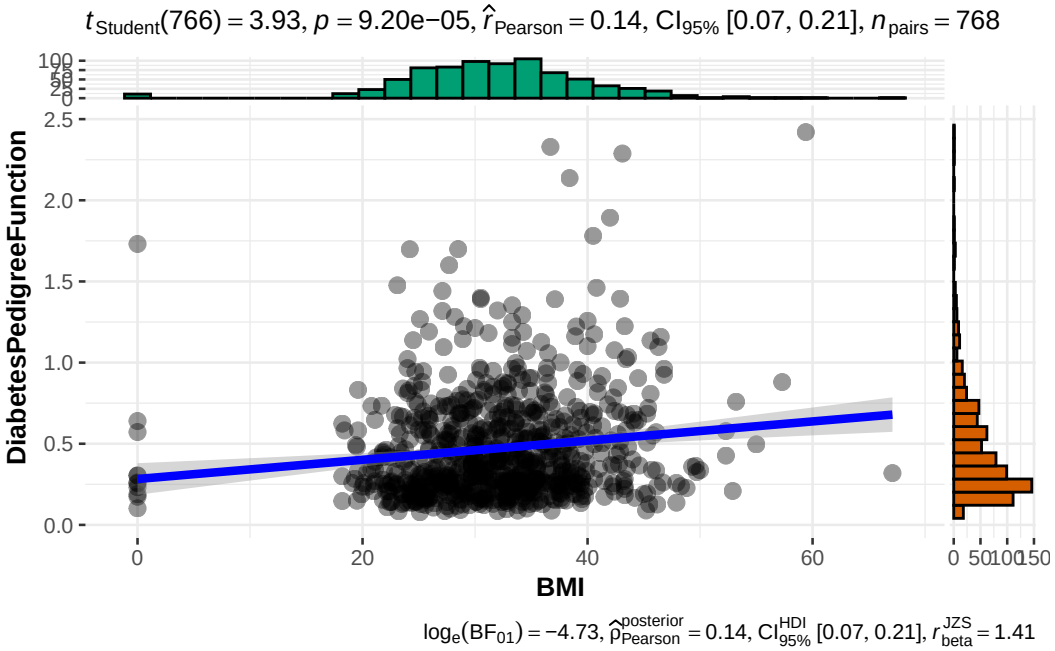
En lo particular la variable del pedigree se me hace importante, entonces vamos a realizar gráficos de dispersión

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

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

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

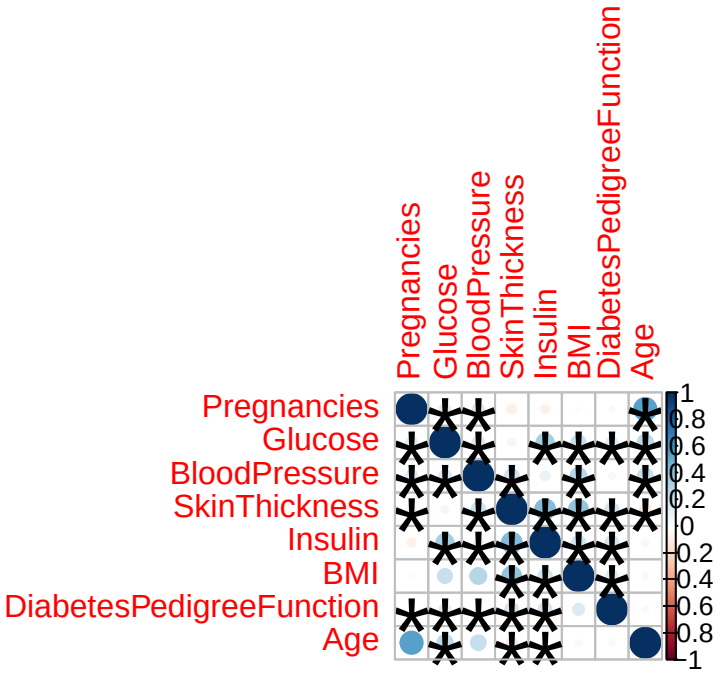
```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



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

```
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
```

```r
diag(p.values) <- 1
corrplot::corrplot(corr = obj.cor$r,p.mat = p.values,sig.level = 0.05,insig = "label_sig")
```



COMENTARIO:

La fución corr.test realiza una prueba de correlación en la que se utilizarán las columnas del conjunto de datos desde la primera columna hasta la n1.

Upper: Se utiliza para actualizar los valores de p en la parte superior de la matriz "p.values" con los valores ajustados obtenidos de "obj.cor$p.adj".

`upper.tri(p.values)` devuelve una matriz booleana con el mismo tamaño que "p.values", donde los elementos correspondientes a la parte superior de la matriz tienen el valor **`TRUE`** y los elementos correspondientes a la parte inferior y diagonal principal tienen el valor **`FALSE`**

La función **`corrplot`** del paquete "corrplot" genera un gráfico de matriz de correlación. En la que:

- **`corr`**: La matriz de correlación obtenida a partir de "obj.cor$r".
- **`p.mat`**: La matriz de valores de p ajustados obtenida anteriormente.
- **`sig.level`**: El nivel de significancia utilizado para determinar qué correlaciones se consideran significativas. En este caso, se establece en 0.05, lo que significa que solo se mostrarán en el gráfico las correlaciones con valores de p menores a 0.05.

En p.values se guardan los valores de p obtenidos de la prueba de correlación realizada.

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

```
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
```

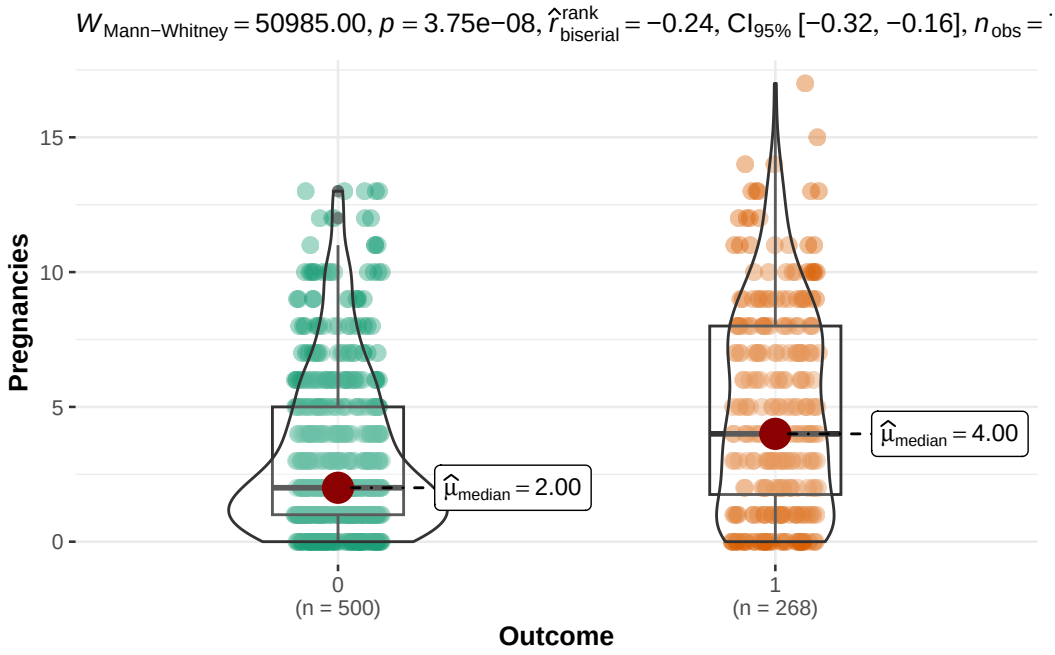
COMENTARIO: La función proporcionada ajusta un modelo de regresión lineal donde "x" es cada columna de datos y "datos$Outcome" es la variable de respuesta. Luego, se extraen los residuos del modelo usando: **`summary(lm(x~datos$Outcome))$residuals`**. Esto devuelve

una matriz donde cada columna contiene los residuos del modelo ajustado para cada columna de datos.

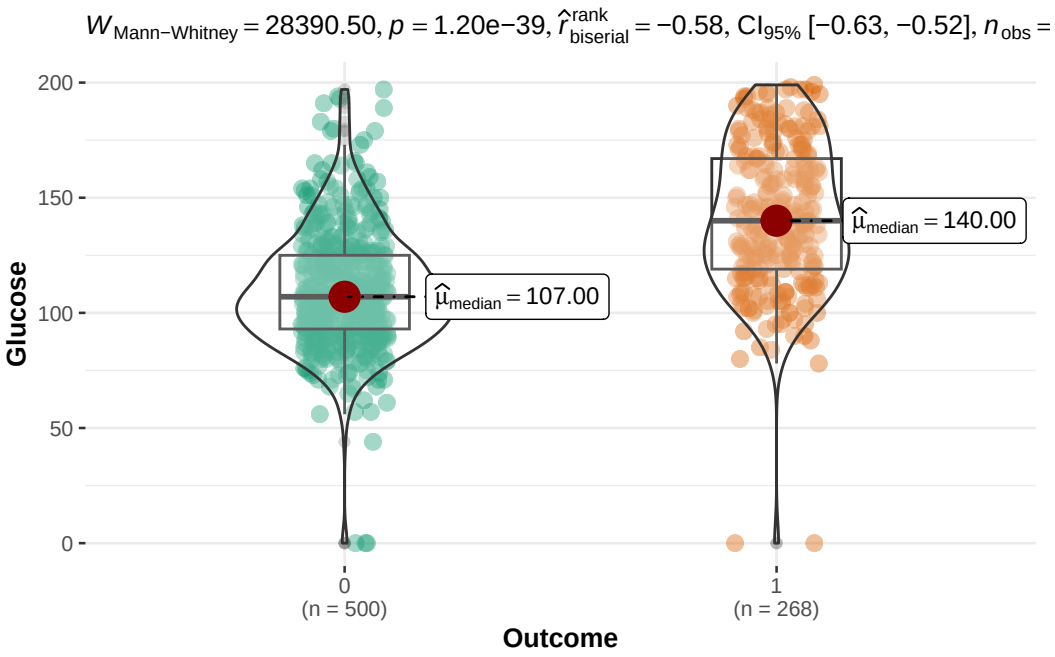
El vector p.norm contiene los valores de p de las pruebas de normalidad realizadas con los residuos de cada modelo.

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

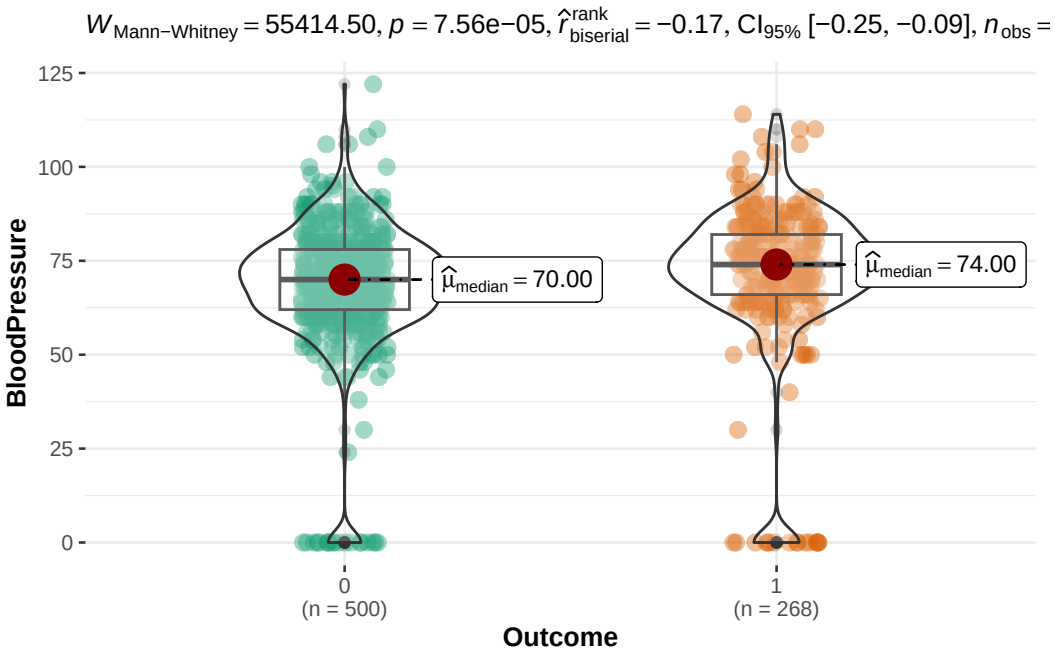
```
ggbetweenstats(datos,Outcome,Pregnancies,type = "nonparametric")
```



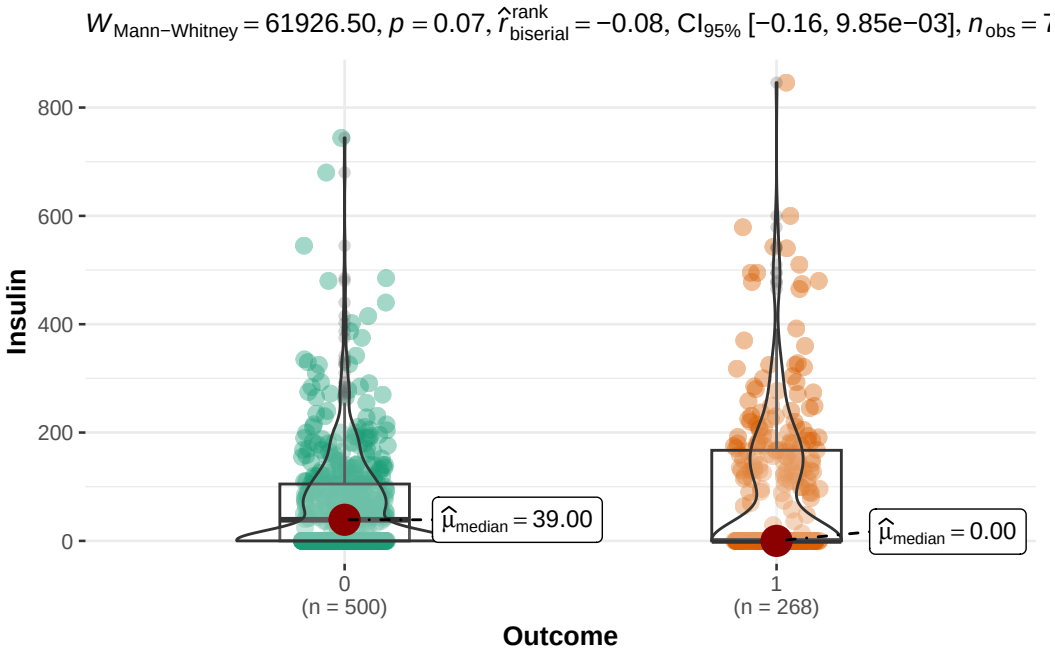

```
ggbetweenstats(datos,Outcome,Glucose,type = "nonparametric")
```



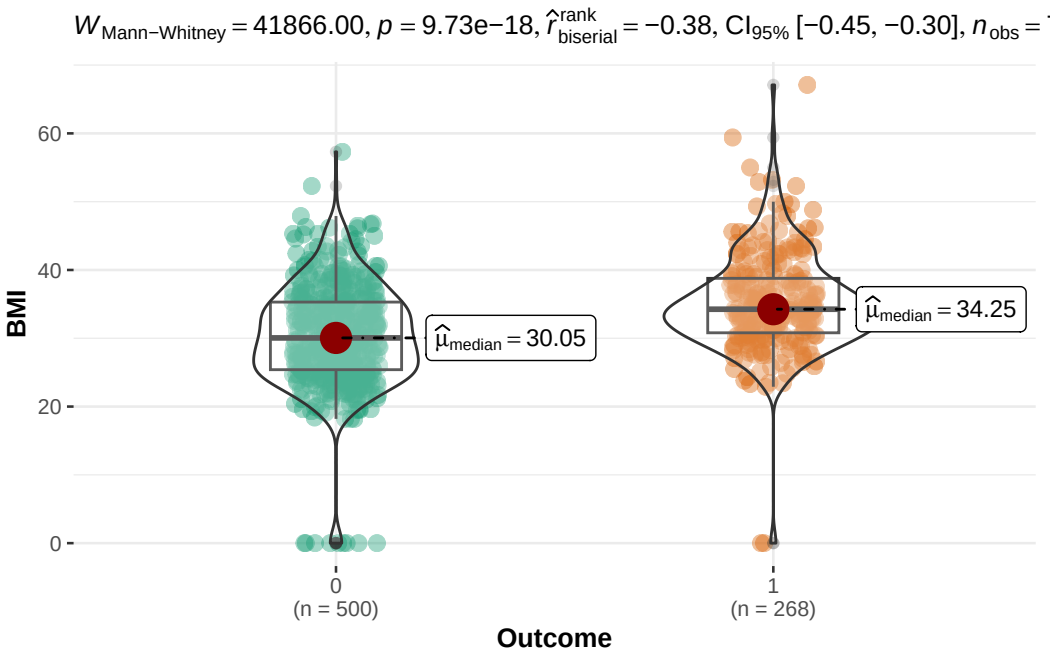

```
ggbetweenstats(datos,Outcome,BloodPressure,type = "nonparametric")
```



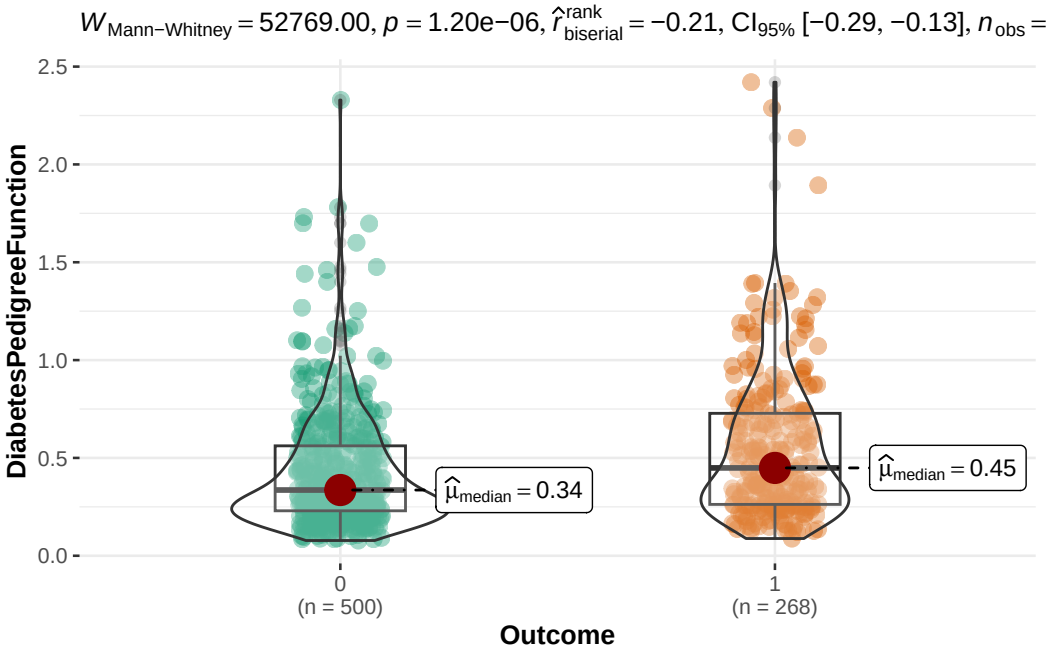
```
ggbetweenstats(datos,Outcome,Insulin,type = "nonparametric")
```



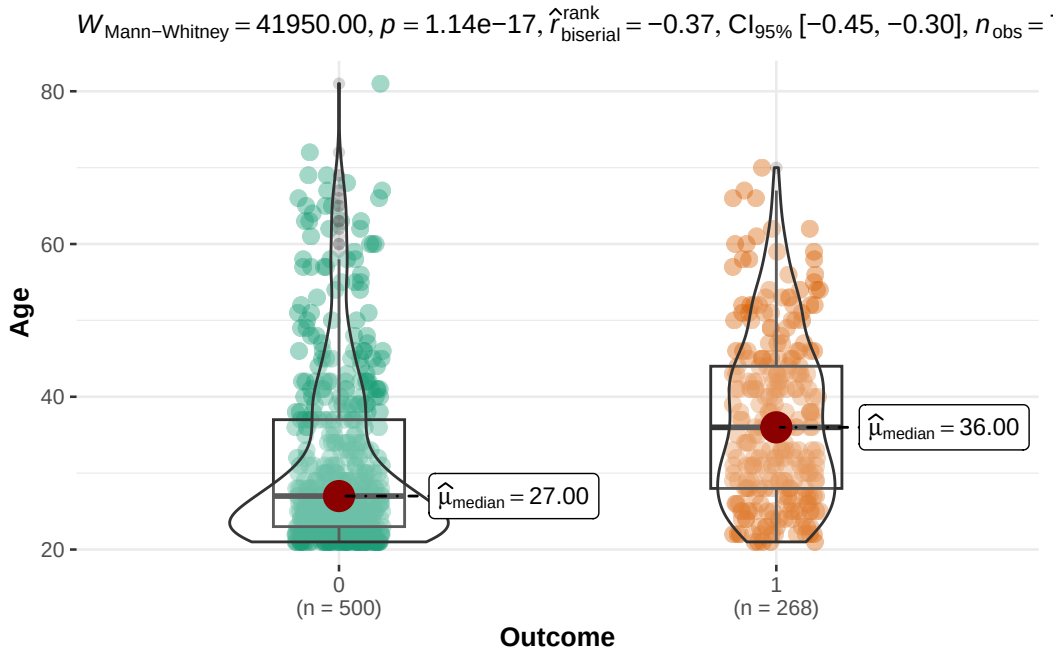

```
ggbetweenstats(datos,Outcome,BMI,type = "nonparametric")
```




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



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



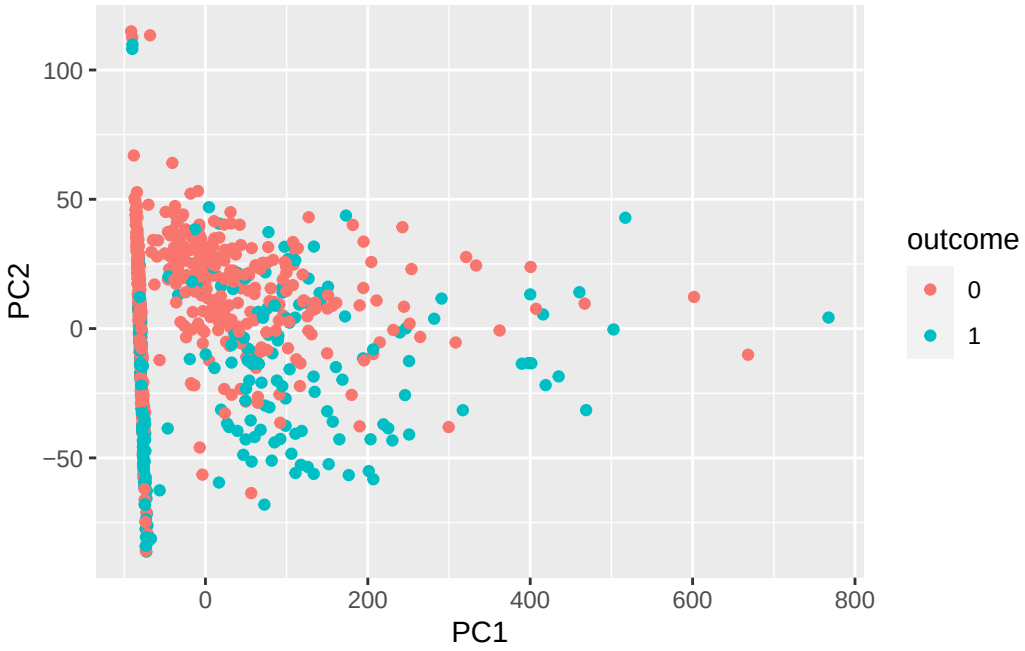
PCA

```
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
```

```
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()
```



COMENTARIO:

La función **prcomp** se utiliza para calcular las componentes principales. El argumento **scale. = F** indica que no se deben escalar los datos por la variabilidad.

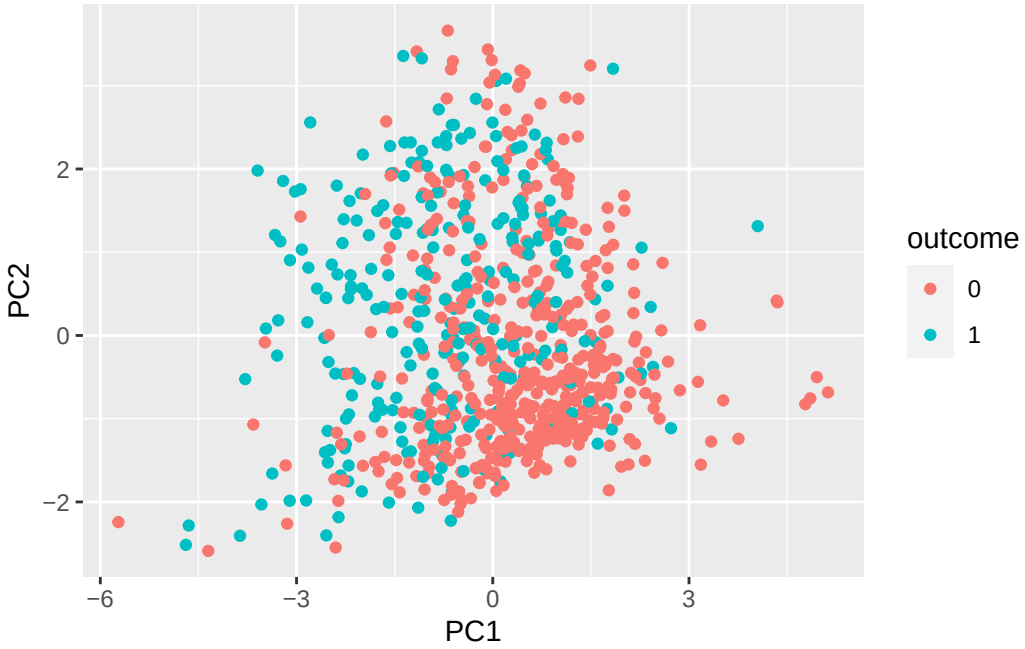
Ahora vamos a ver si haciendo unas transformaciones esto cambia. Pero antes debemos de ver las variables sospechosas...

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

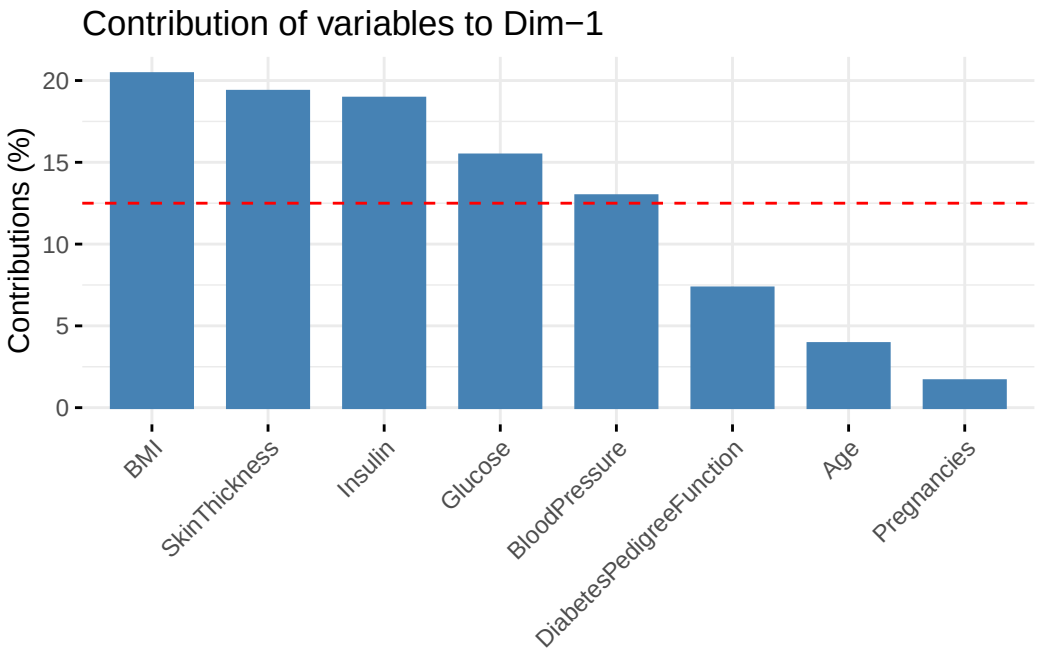
```
summary(datos)
```

```
 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
```

```
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()
```




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



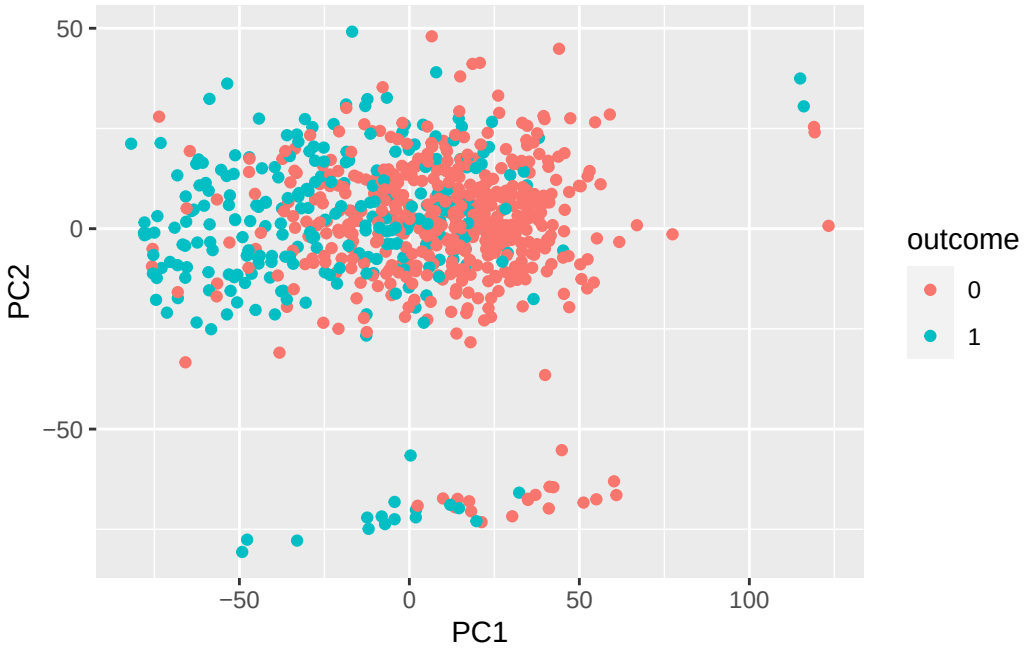
COMENTARIO:

La función **`fviz_contrib`** calcula y visualiza las contribuciones de las variables a las componentes principales. Esto se hace a través de un gráfico de barras, donde cada barra representa la contribución relativa de una variable a cada componente principal.

Este código indica cómo cada variable influye en la estructura de las componentes principales y ayuda a identificar qué variables tienen un mayor impacto en la variabilidad capturada por el PCA.

Al parecer es la insulina la que está dando problemas

```
## 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()
```



COMENTARIO:

grep se utiliza para buscar las columnas del objeto "datos" que contengan la cadena de caracteres "insulin", ignorando la distinción entre mayúsculas y minúsculas (**ignore.case = TRUE**). Las columnas que coincidan se agregan al vector "w".

Este código realiza un PCA en el conjunto de datos, excluyendo las columnas que contienen la cadena "insulin" en su nombre. Luego, visualiza las dos primeras componentes principales en un gráfico de dispersión, donde los puntos se colorearán según los valores de la variable "Outcome"

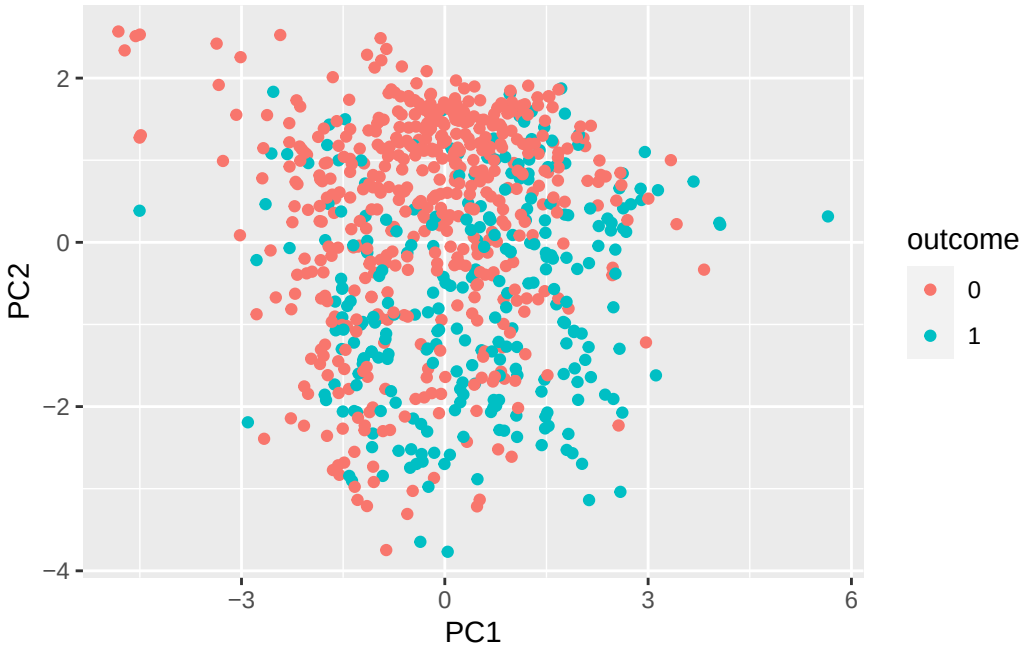
De hecho la insulina, tenía un aspecto raro, como sesgado, ver gráficos de arriba. Vamos a transformala...

```
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
```

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

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



COMENTARIO:

`datos$Insulin <- log(datos$Insulin+0.05)` calcula el logaritmo natural (base e) de la variable "Insulin"y lo asigna nuevamente a la misma columna . Se agrega 0.05 al valor original antes de aplicar el logaritmo para evitar problemas con valores cercanos a cero.

Este código realiza una transformación logarítmica en la variable "Insulin" y luego realiza un PCA en las columnas de datos transformadas. Luego, visualiza las dos primeras componentes principales en un gráfico de dispersión, donde los puntos se colorearán según los valores de la variable "Outcome".

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

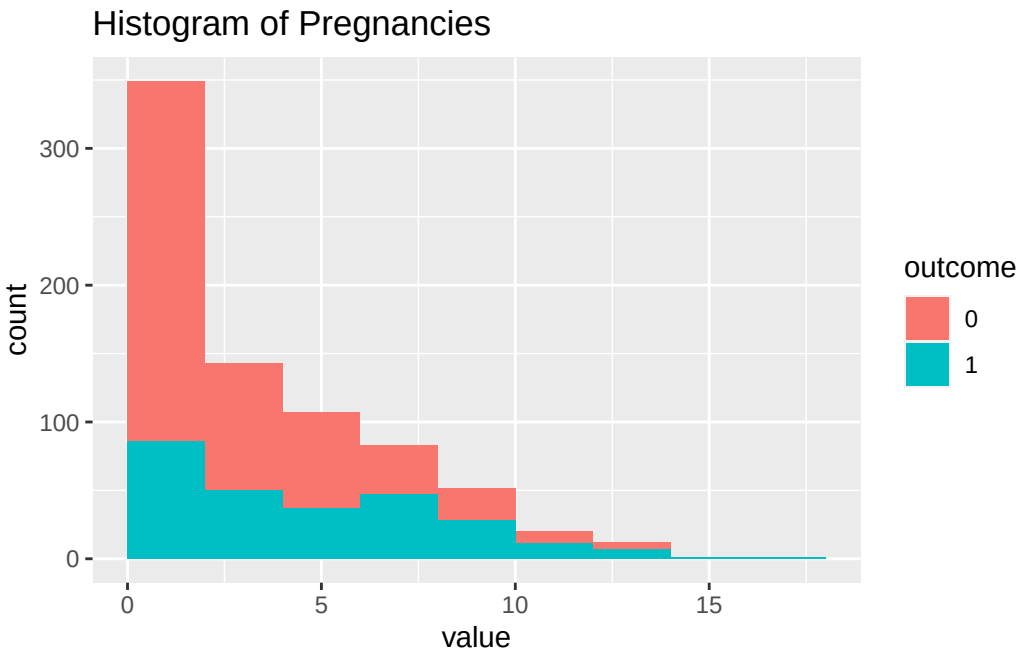
Es decir, cambia si transformamos los datos…a partir de esto, podemos realizar de nuevo pruebas de diferencia de medianas, pero ahora lo veremos condensado..

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

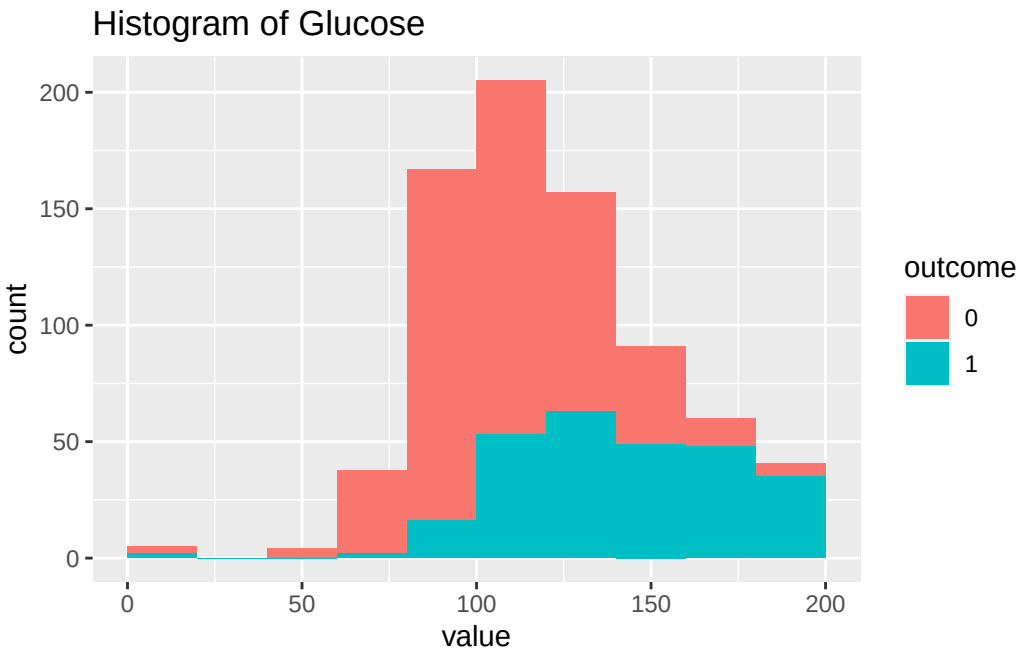
Veamos las distribuciones de nuevo....

```r
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
```

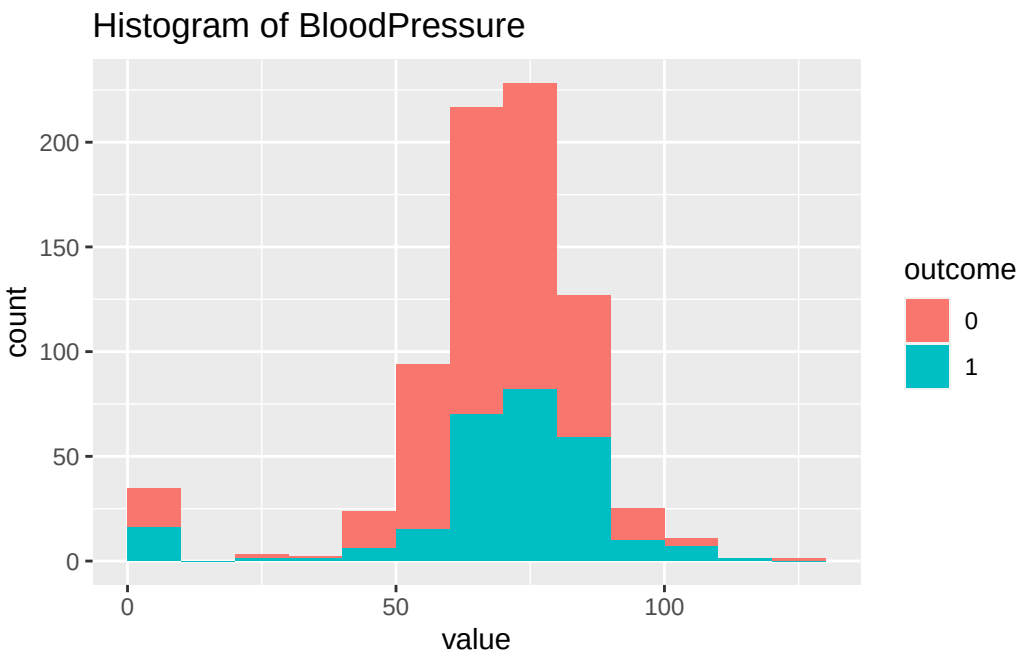
```
[[1]]
```



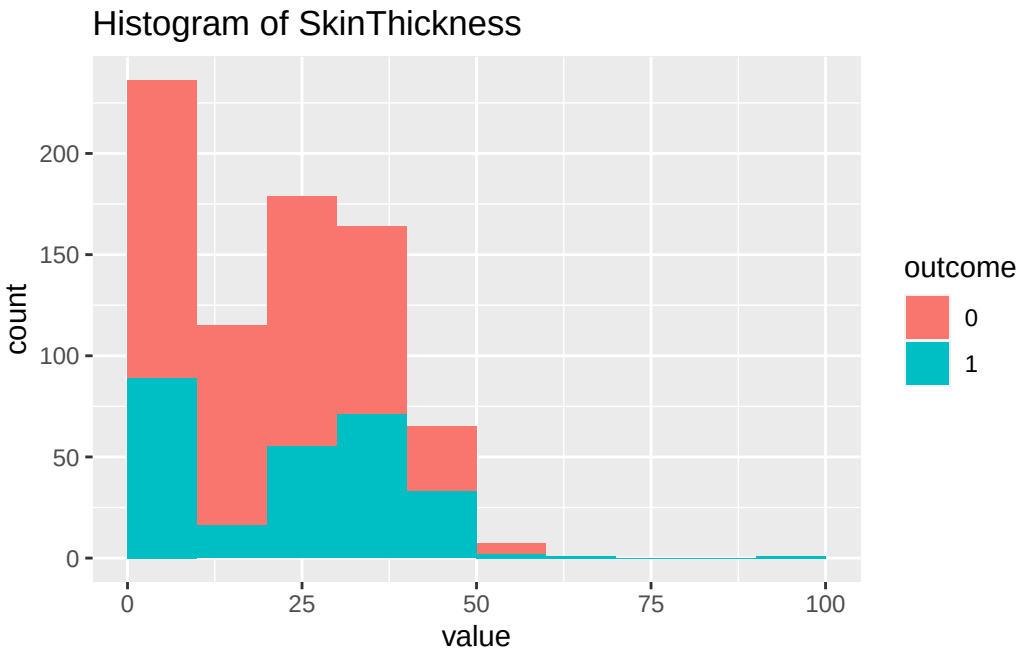
```
[[2]]
```



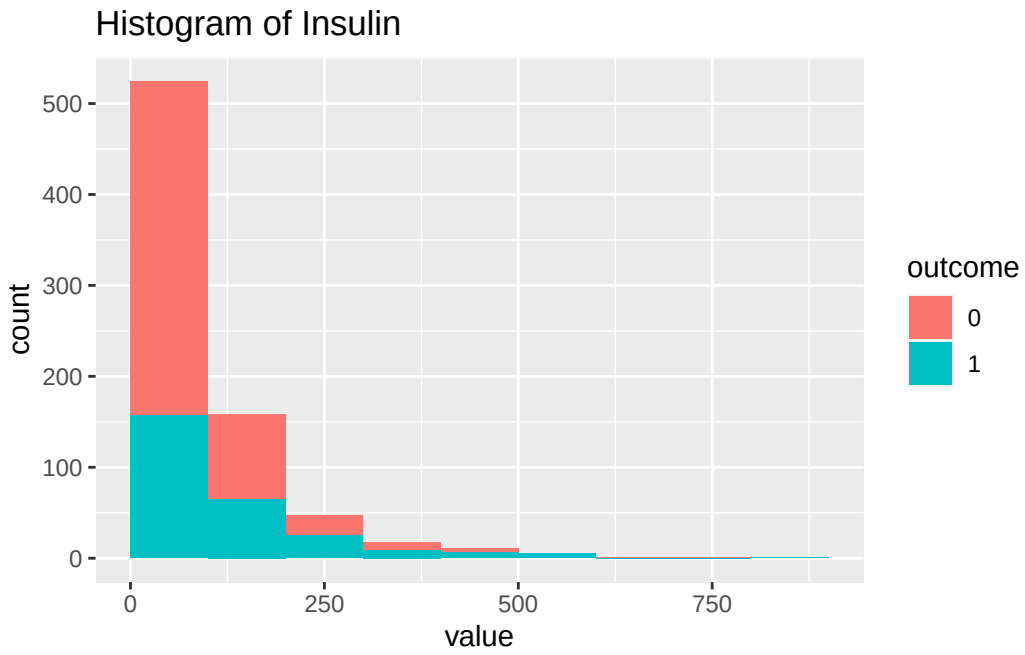

```
[[3]]
```


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

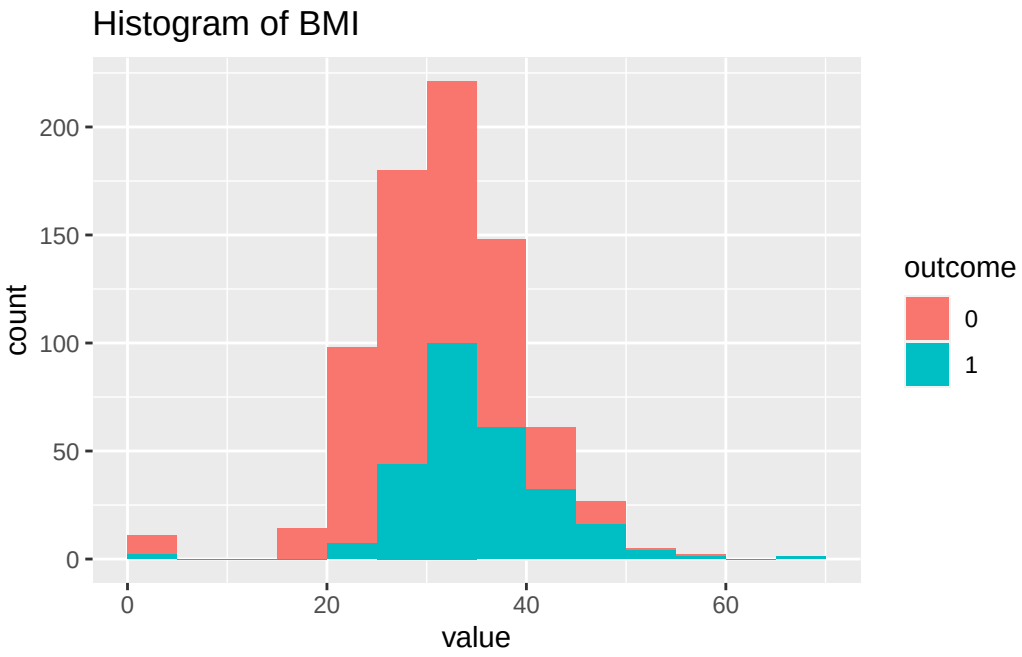
```
[[4]]
```


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

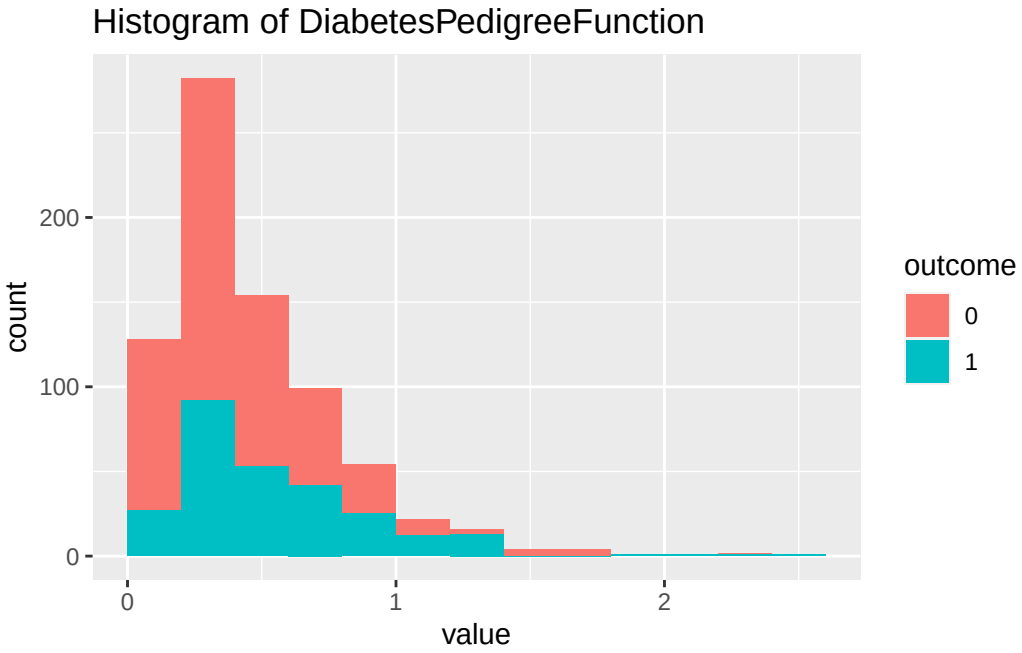
[[5]]


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

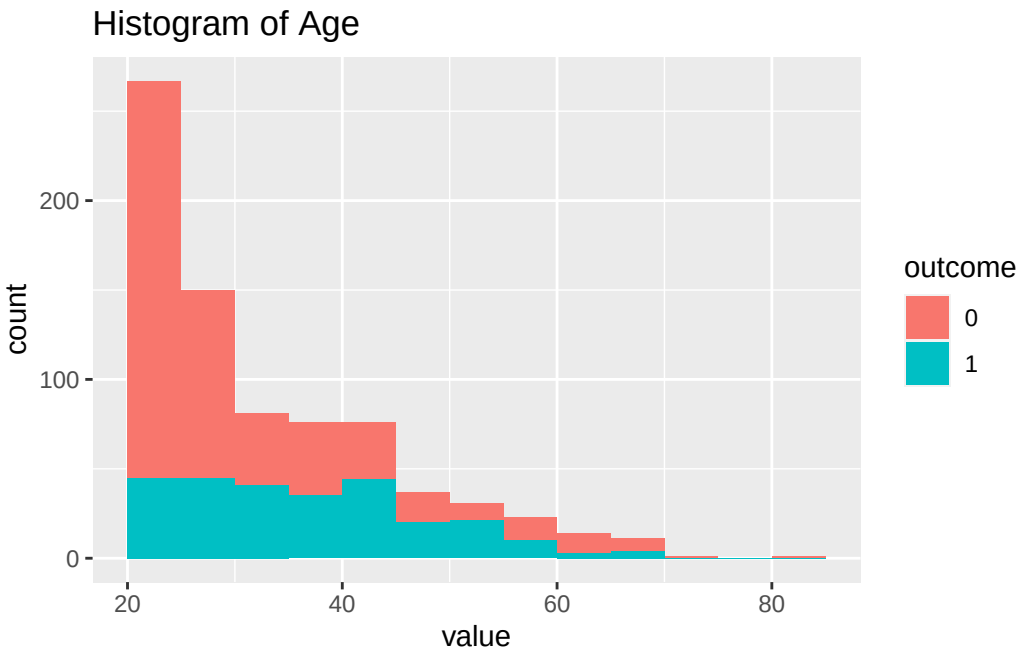
[[6]]


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

```
[[7]]
```


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

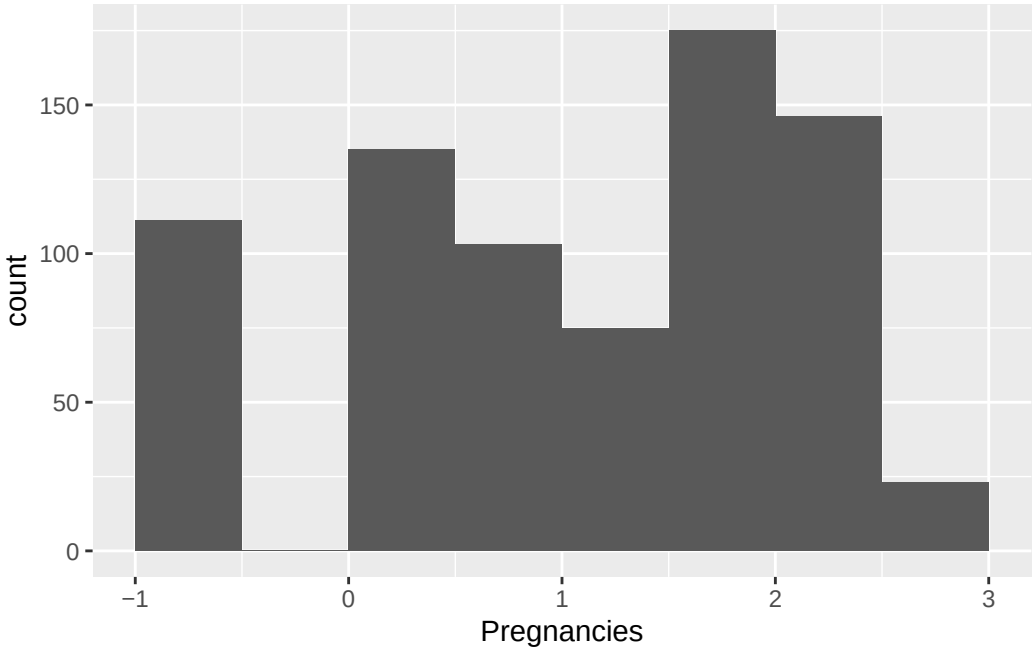
```
[[8]]
```



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...

```
datos <- read.csv("./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
```

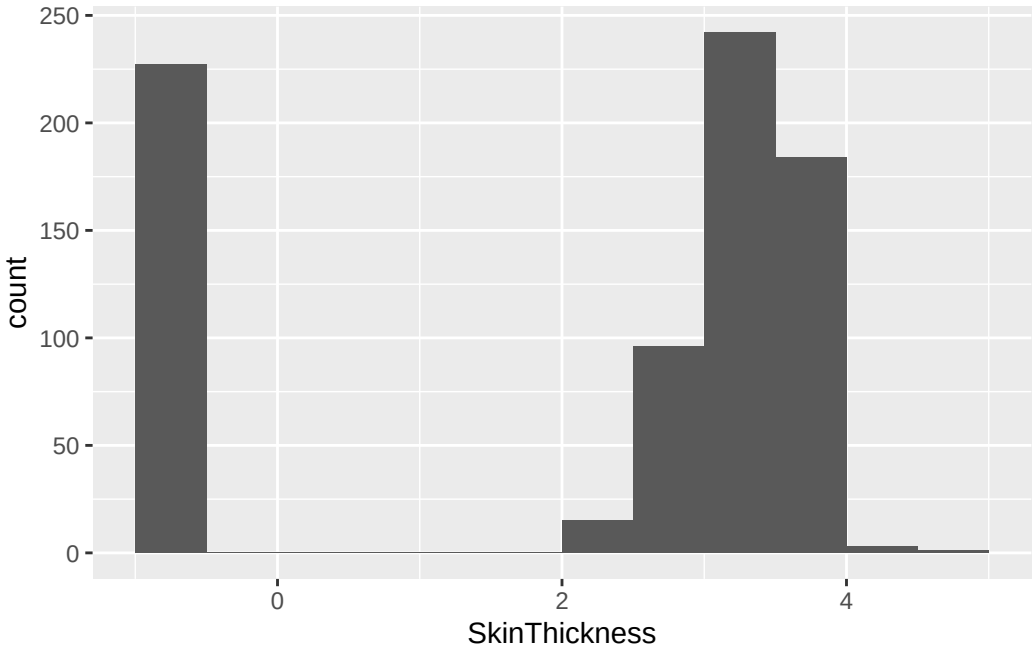


COMENTARIO:

Aquí unicamente se agrega una tranformación logarítmica a "Pregnancies" y se realiza un diagrama de barras.

Realizaremos lo mismo con la grosura de la piel

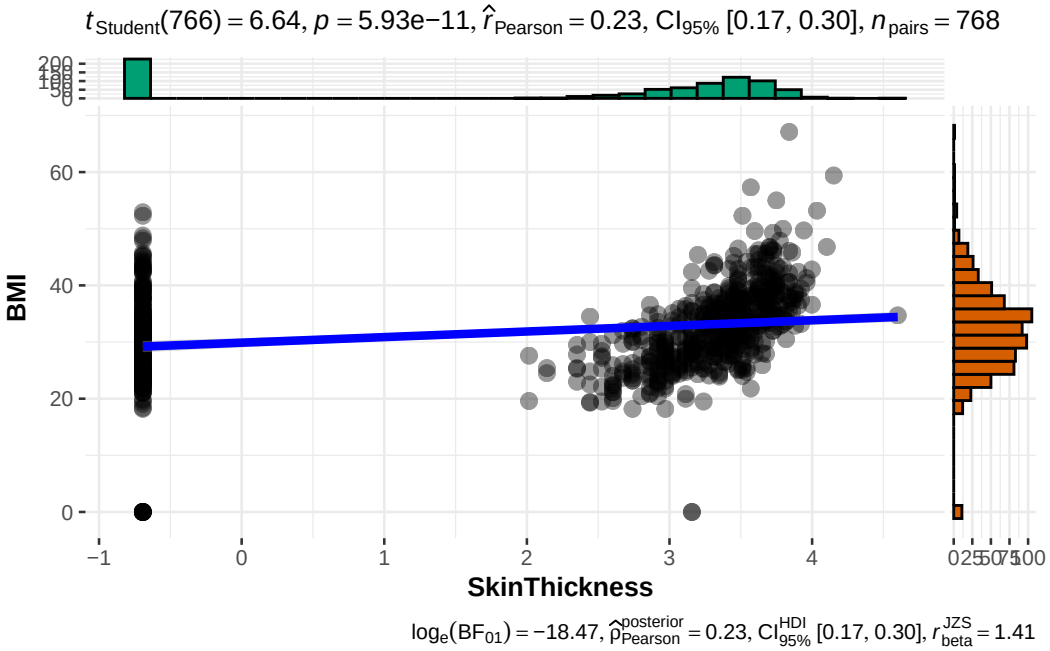
```
datos <- read.csv("./diabetes.csv")
datos$Outcome <- as.factor(datos$Outcome)
datos$SkinThickness  <- log(datos$SkinThickness+0.5)
ggplot(datos,aes(SkinThickness))+geom_histogram(breaks = hist(datos$SkinThickness,plot=F)$
```



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

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

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



COMENTARIO: Cada punto en el gráfico representa una observación en el conjunto de datos, y la posición del punto en el eje x corresponde al valor de "SkinThickness", mientras que la posición en el eje y corresponde al valor de "BMI".

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("./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)
```

COMENTARIO: Se utiliza **`ifelse`** para reemplazar los valores iguales a cero por **`NA`** (valores perdidos), y mantener los demás valores sin cambios.

---

vamos a quitar estos valores

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

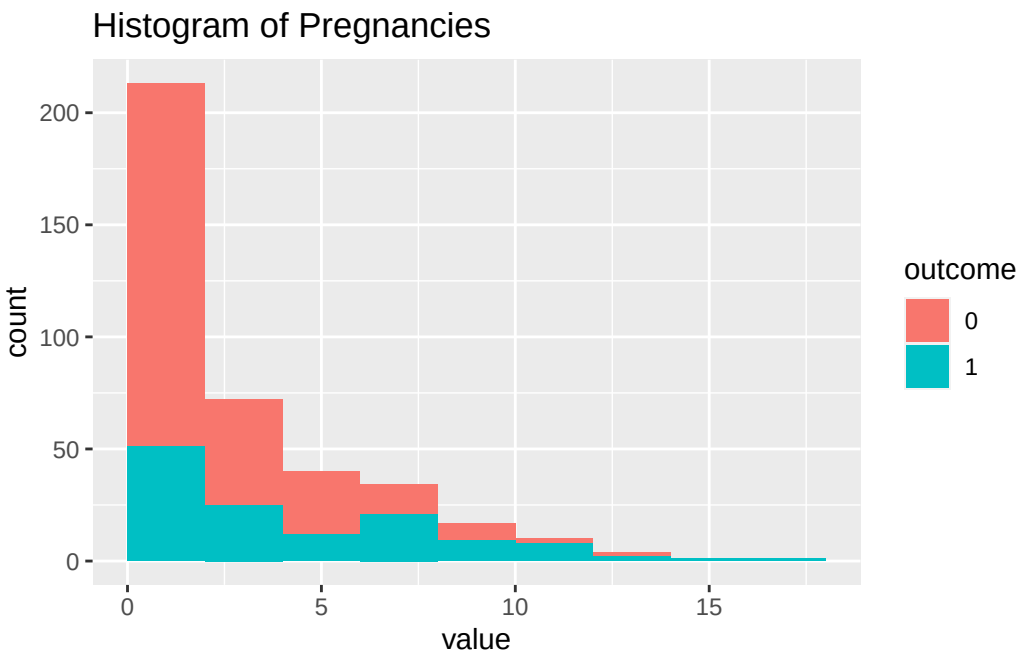
Se redujo el data set a 392 observaciones...

```
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
```

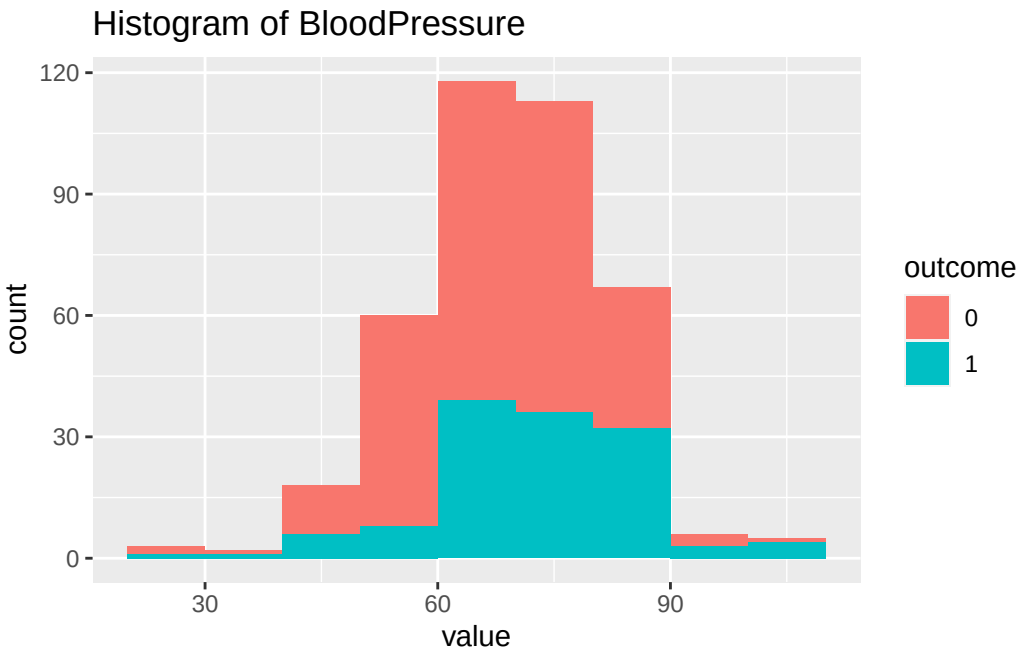
```
[[1]]
```


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

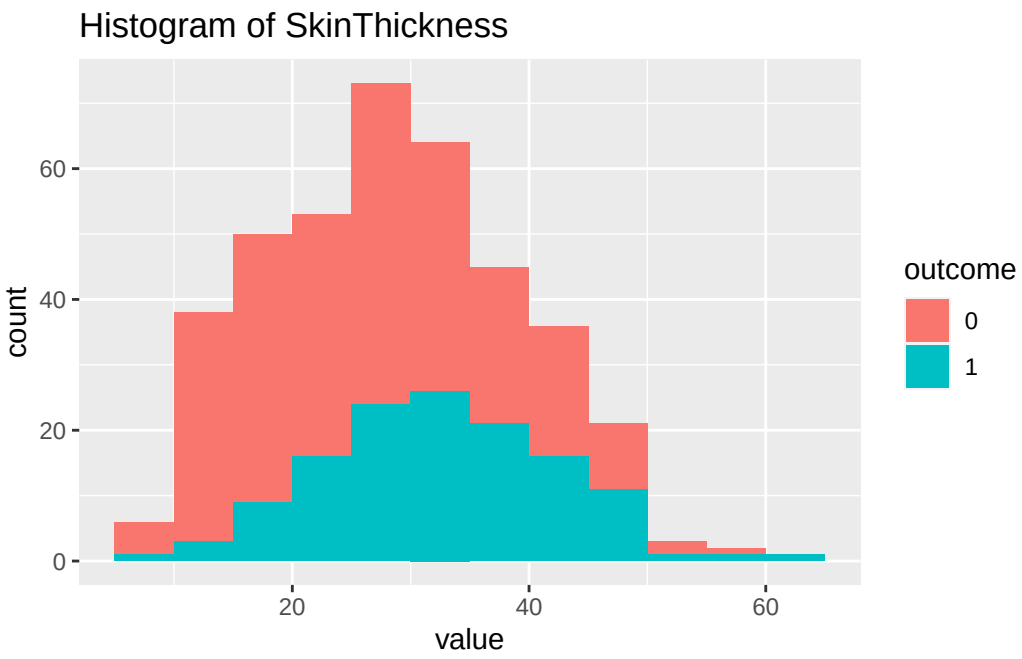
```
[[2]]
```


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

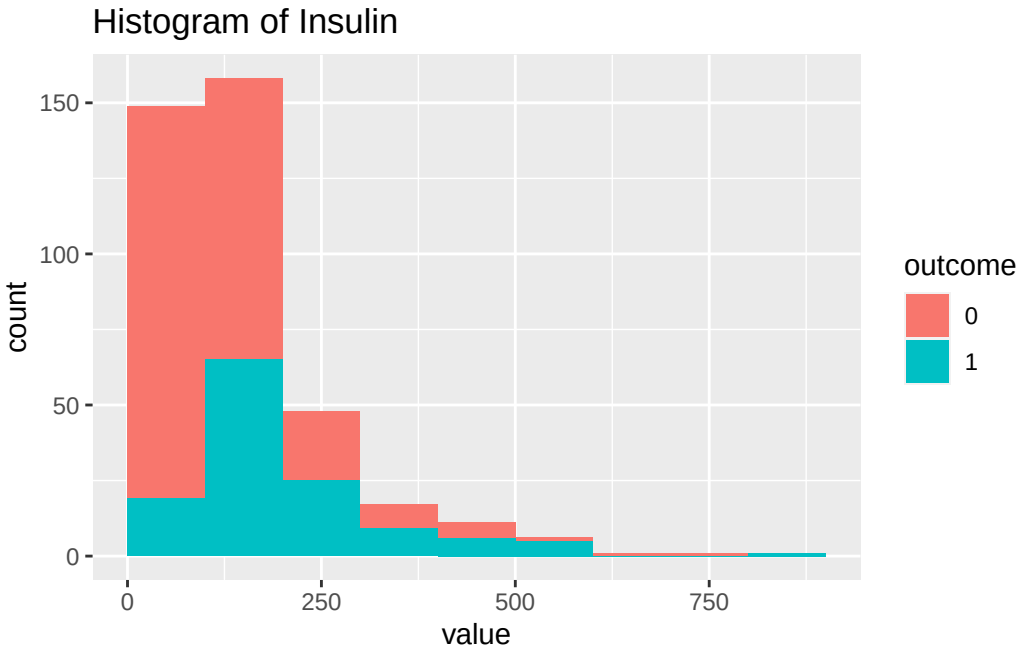
```
[[3]]
```


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

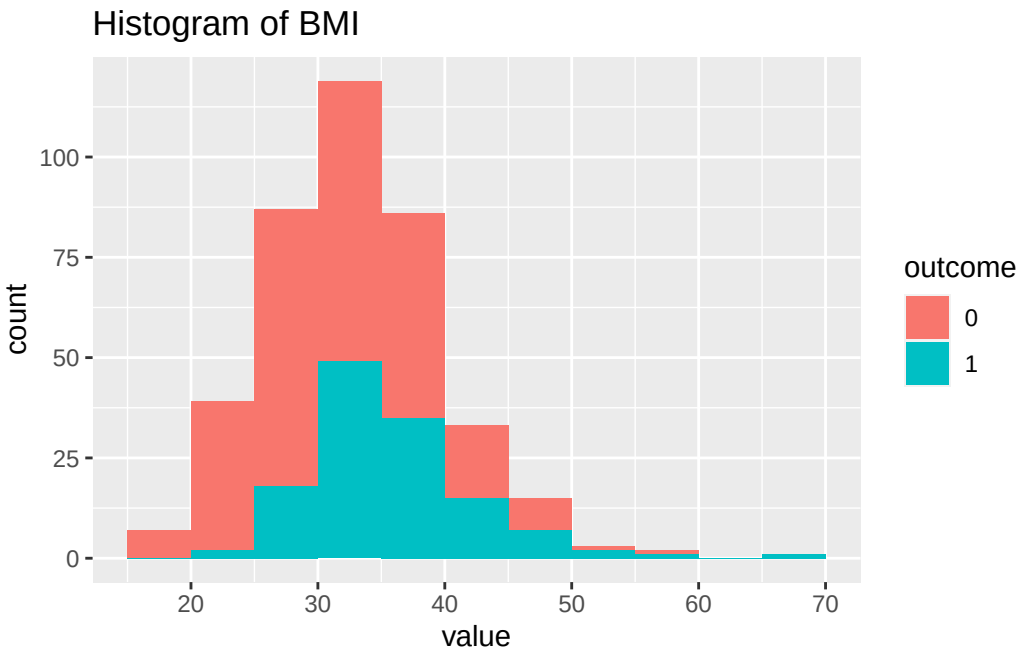
```
[[4]]
```


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

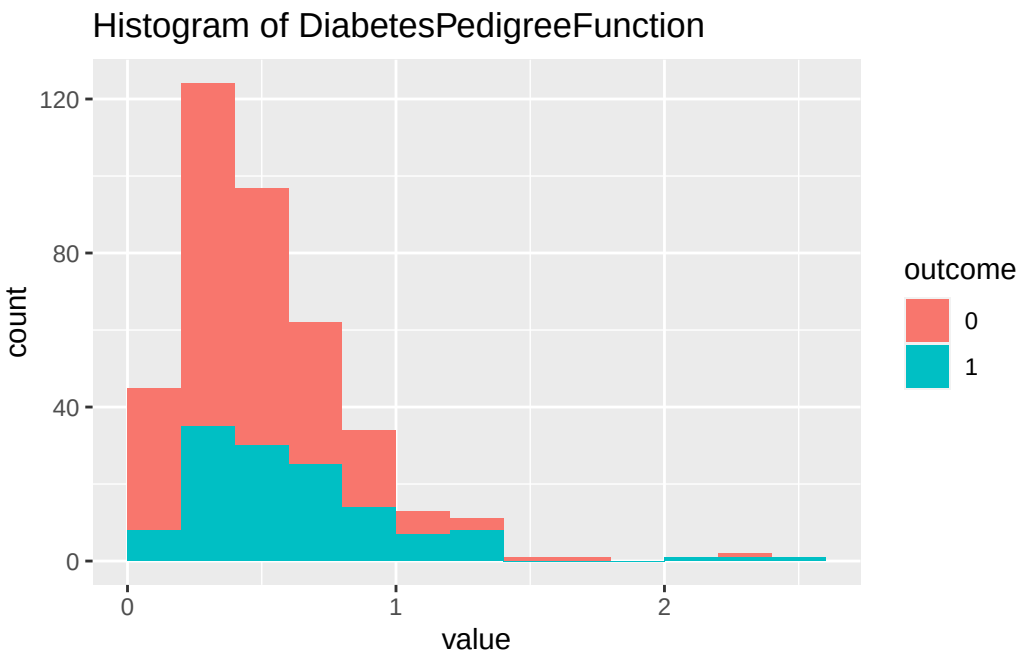
```
[[5]]
```


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

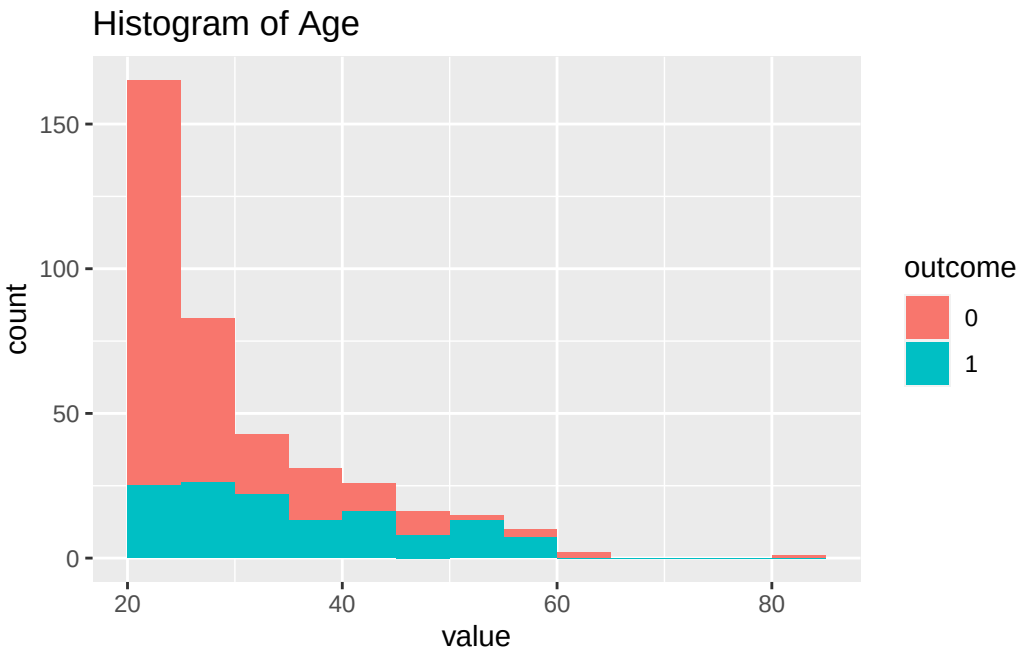
[[6]]


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

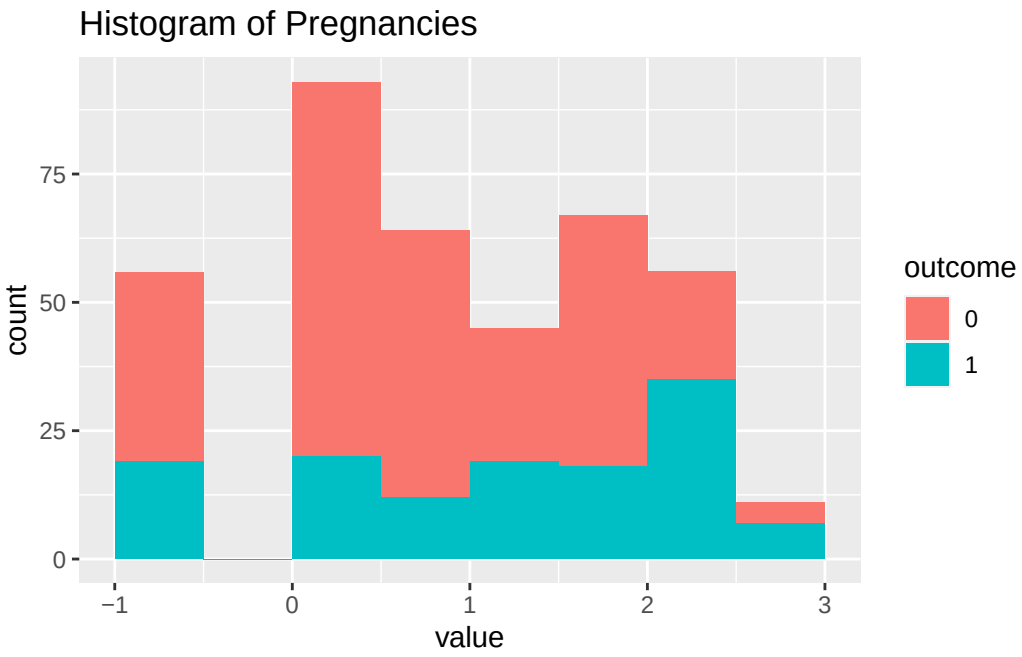
```
[[8]]
```


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

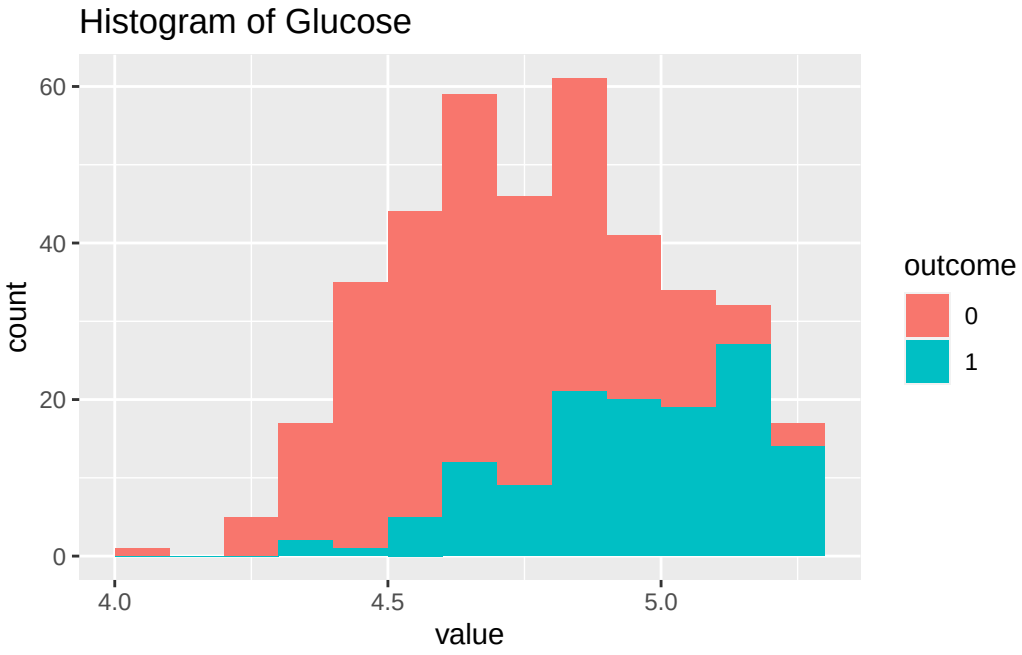
Ahora si podemos realizar las transfomraciones

```
datos <- read.csv("./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
```

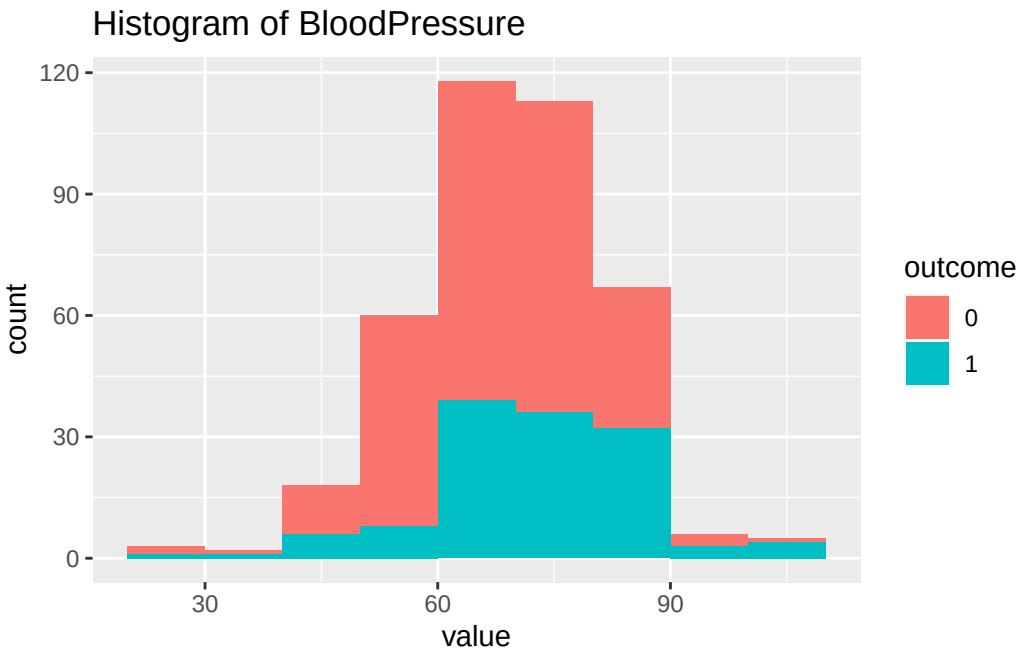
```
[[1]]
```


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

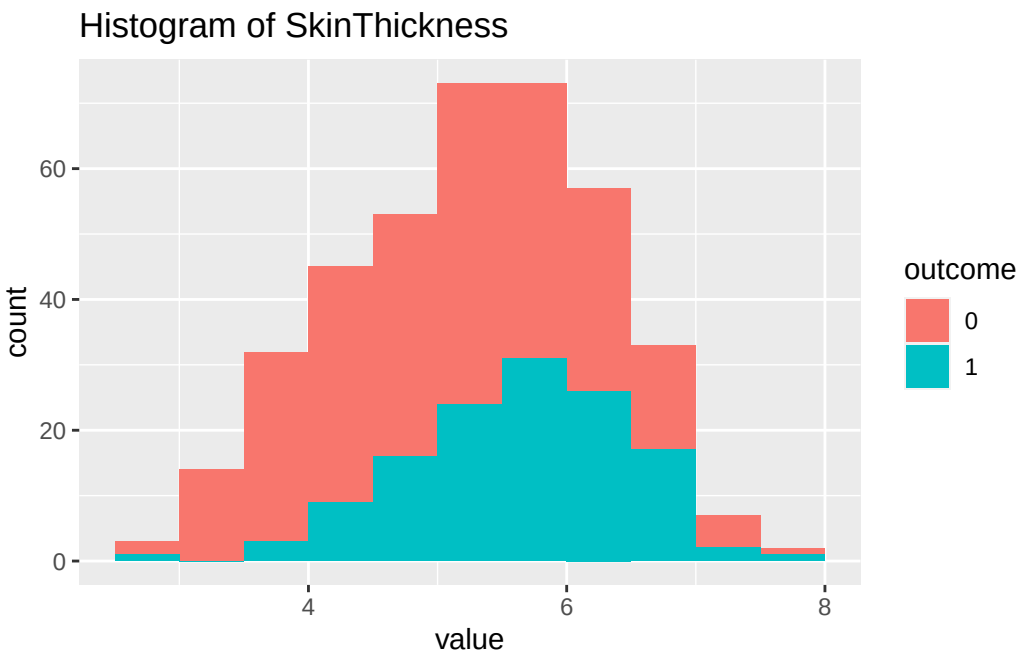
```
[[2]]
```


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

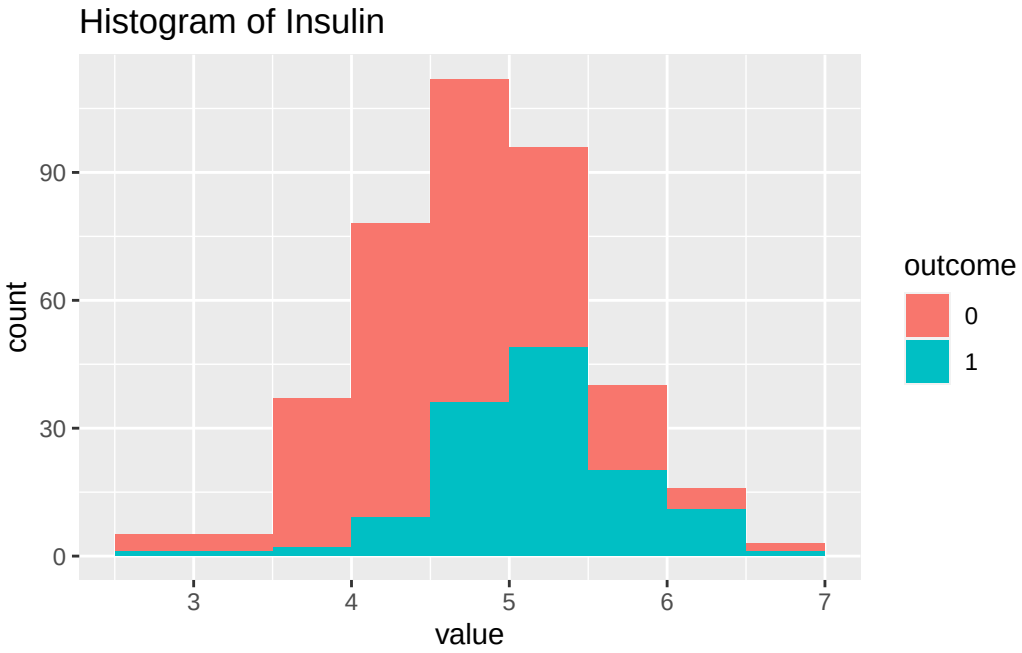
```
[[3]]
```


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

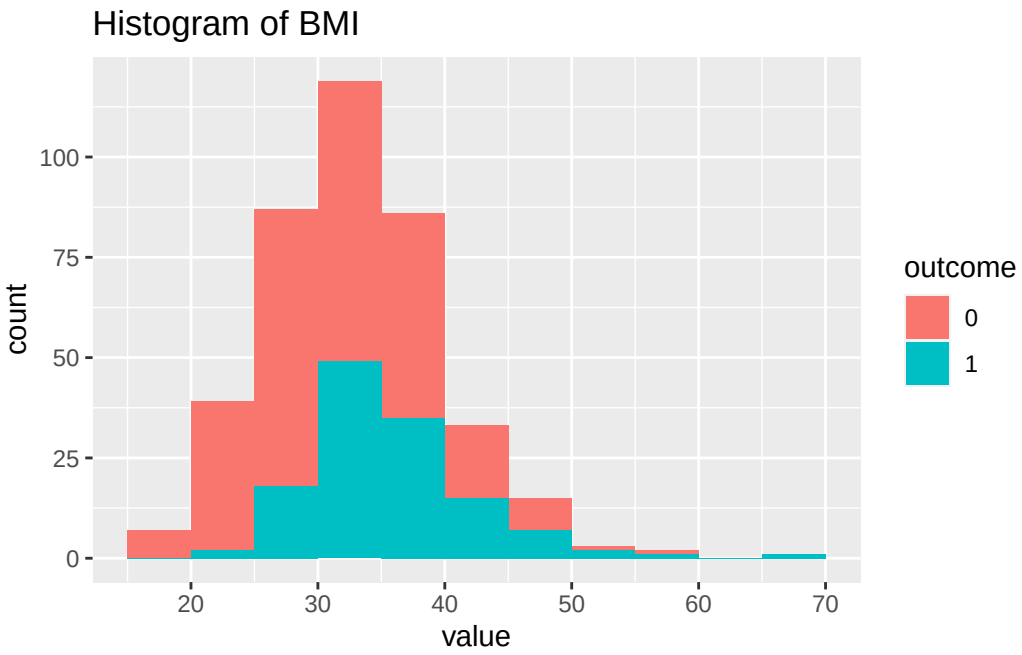
```
[[4]]
```


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

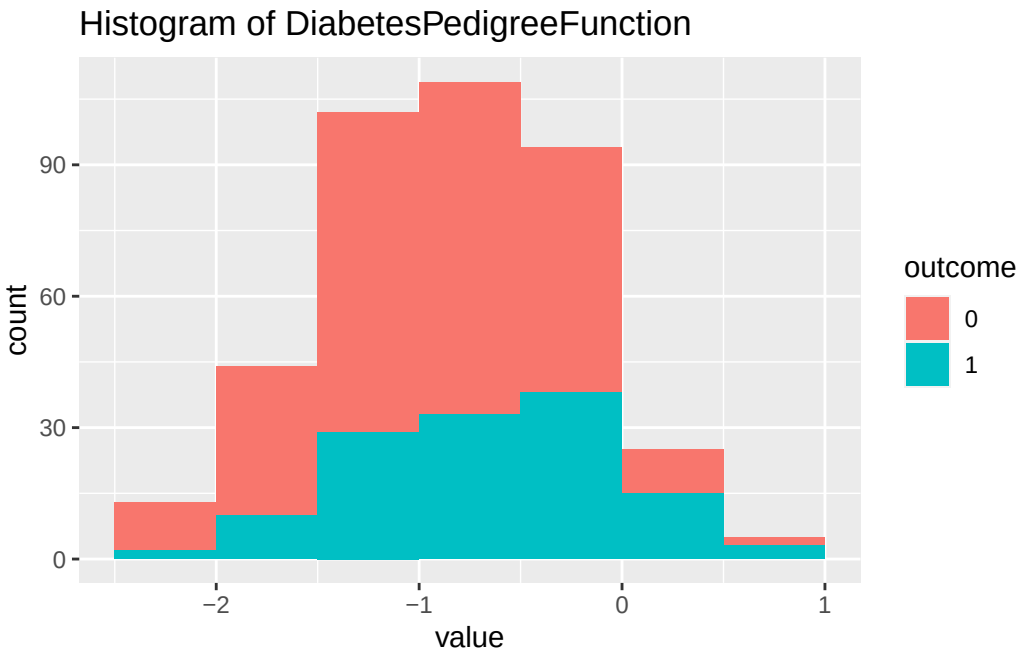
```
[[5]]
```


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

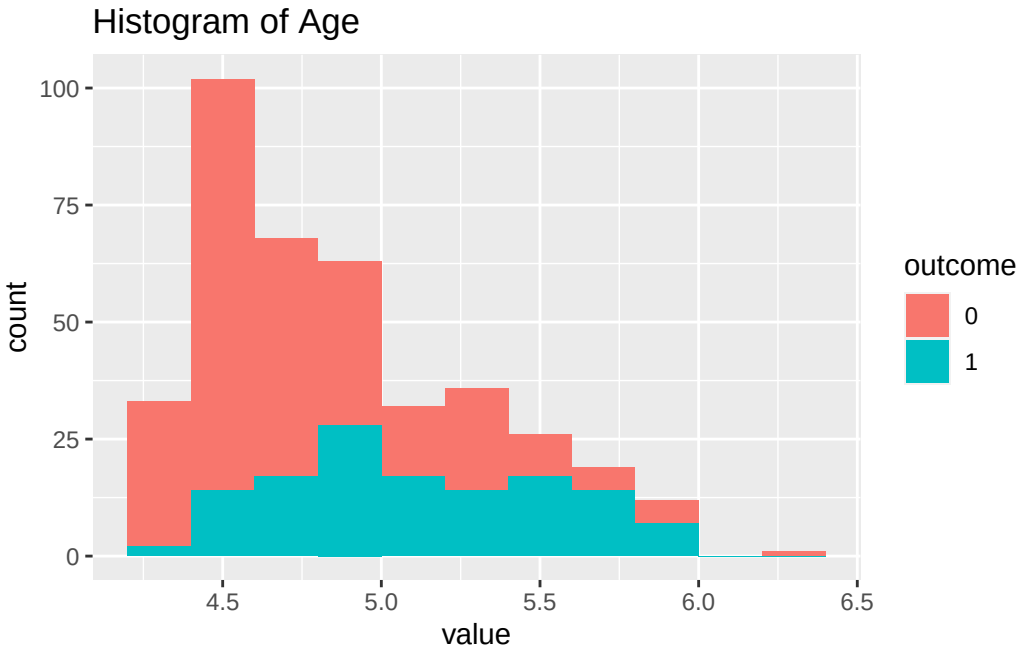
```
[[6]]
```


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

COMENTARIO: `complete.cases` filtra el objeto "datos" para eliminar las filas que contienen valores (`NA`) en alguna columna, es decir se utiliza para determinar si una observación contiene valores completos en todas las columnas. Las filas que cumplen esta condición se mantienen en el objeto "datos".

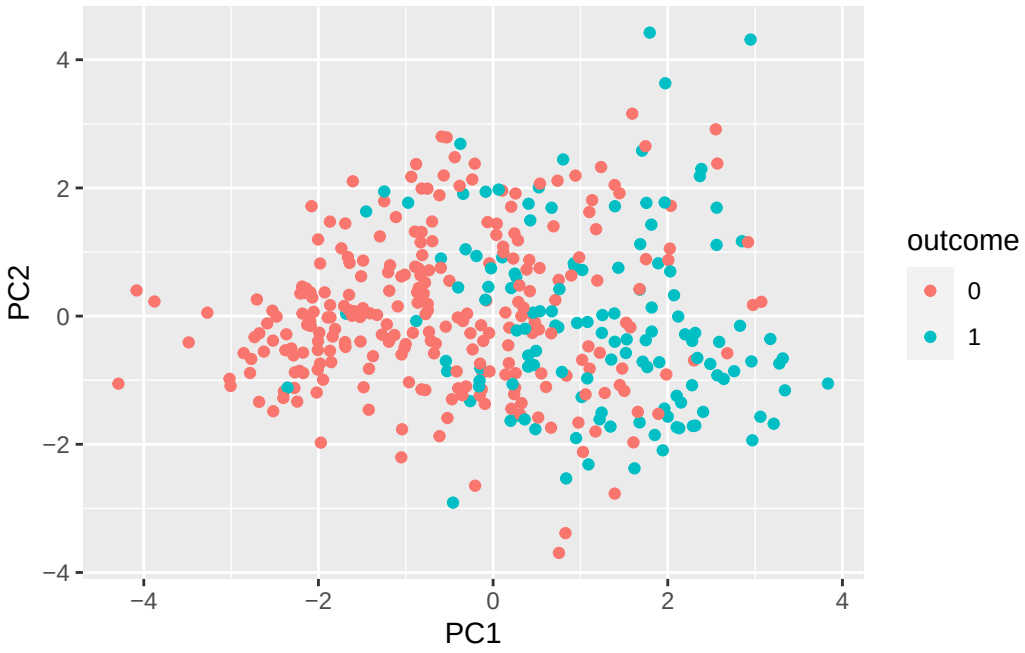
`sqrt` calcula la raíz cuadrada de los valores en la columna "SkinThickness". Puede ser útil para ajustar la distribución de los datos si están sesgados hacia valores más altos.

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

```
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

```
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

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

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

```
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
```

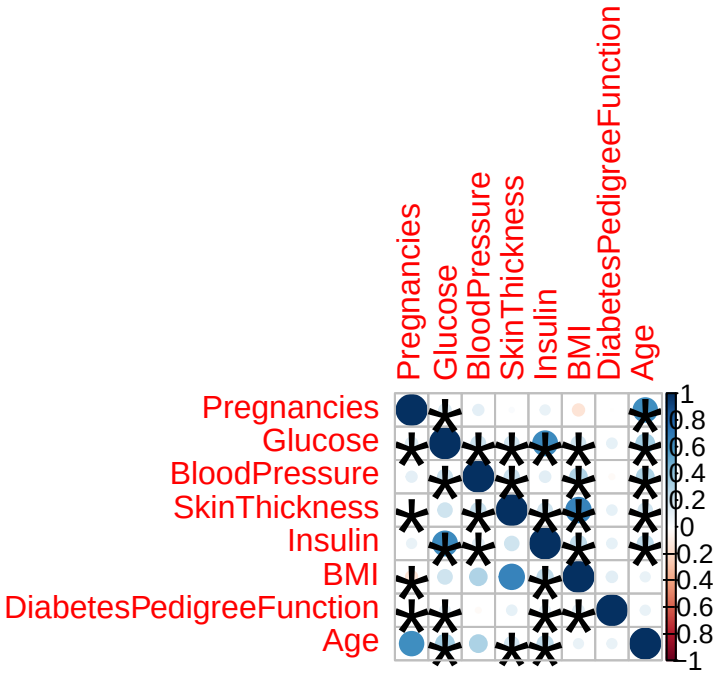
```
DiabetesPedigreeFunction -0.2779529 8.957407e-05
Age                      -0.4005379 1.577456e-14
```

COMENTARIO:

El código divide el conjunto de datos "datos" en subconjuntos basados en la variable "Outcome", calcula las medianas de cada subconjunto y las diferencias entre las medianas, y luego crea un dataframe que combina estas diferencias y los valores de p-valor ajustados.

Ahora Todos los valores son significativos respecto a la obesidad

```
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")
```



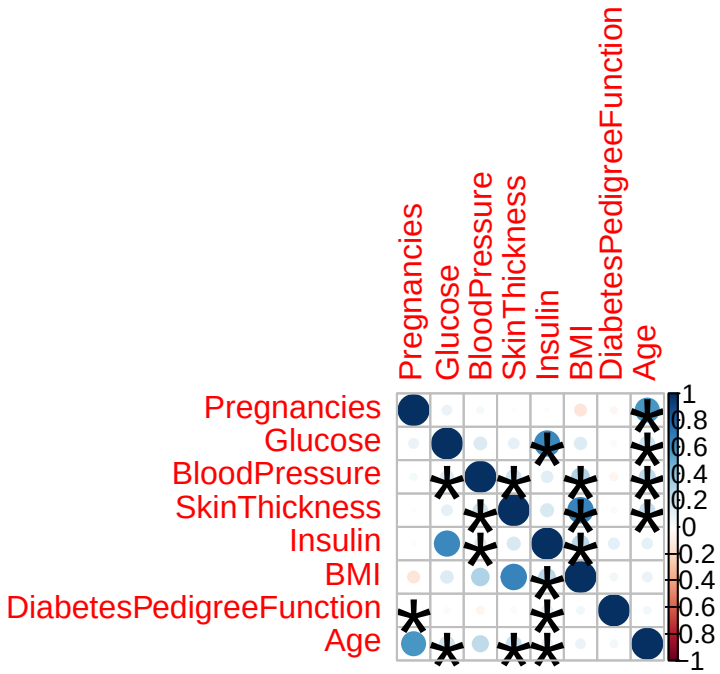
COMENTARIO:

psych::corr.test() es una función que realiza pruebas de correlación en los datos. Toma como entrada una matriz o dataframe y calcula varios coeficientes de correlación, como el

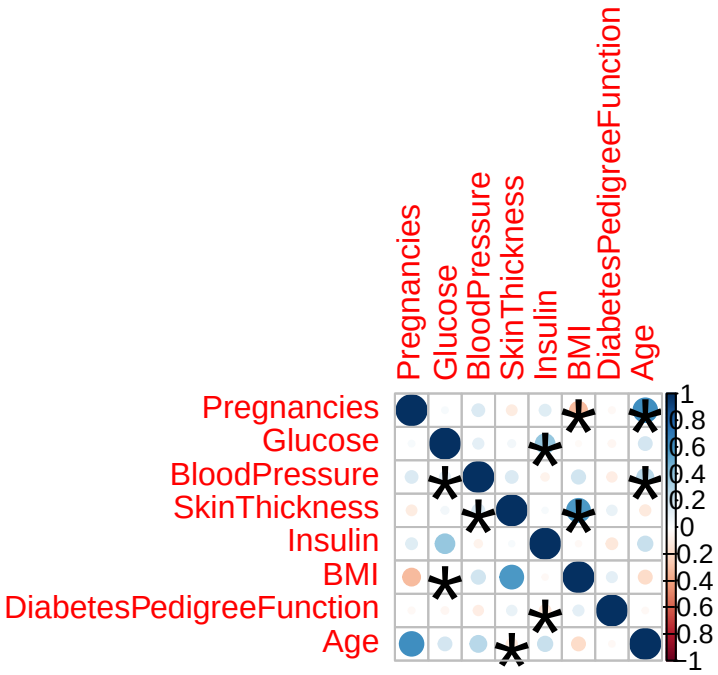
coeficiente de correlación de Pearson y el coeficiente de correlación de Spearman, junto con sus respectivos valores de p-valor.

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

```
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

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

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

```r
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 74 20
         N  5 19

               Accuracy : 0.7881
                 95% CI : (0.7033, 0.858)
    No Information Rate : 0.6695
    P-Value [Acc > NIR] : 0.003185

                  Kappa : 0.4696

 Mcnemar's Test P-Value : 0.005110

            Sensitivity : 0.9367
            Specificity : 0.4872
         Pos Pred Value : 0.7872
         Neg Pred Value : 0.7917
             Prevalence : 0.6695
         Detection Rate : 0.6271
   Detection Prevalence : 0.7966
      Balanced Accuracy : 0.7119

       'Positive' Class : D
```

COMENTARIO:

Aquí 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.

Finalmente se calcula una matriz de confusión para evaluar el rendimiento de las predicciones en comparación con los valores reales.

RIDGE

```
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 74 21
         N  5 18

               Accuracy : 0.7797
                 95% CI : (0.6941, 0.8507)
    No Information Rate : 0.6695
    P-Value [Acc > NIR] : 0.005924

                  Kappa : 0.4444

 Mcnemar's Test P-Value : 0.003264

            Sensitivity : 0.9367
            Specificity : 0.4615
         Pos Pred Value : 0.7789
         Neg Pred Value : 0.7826
             Prevalence : 0.6695
         Detection Rate : 0.6271
   Detection Prevalence : 0.8051
```

```
      Balanced Accuracy : 0.6991

       'Positive' Class : D
```

COMENTARIO:

`ecpand.grid` crea una cuadrícula de sintonización para el ajuste del modelo. En este caso, 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. Esto significa que el modelo se entrenará y evaluará en 10 subconjuntos de datos diferentes, repitiendo el proceso 3 veces. La opción **`classProbs = T`** indica que se deben calcular las probabilidades de clase durante el entrenamiento.

LASSO

```r
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 74 19
         N  5 20
```

```
               Accuracy : 0.7966
                 95% CI : (0.7127, 0.8651)
    No Information Rate : 0.6695
    P-Value [Acc > NIR] : 0.001634

                  Kappa : 0.4945

 Mcnemar's Test P-Value : 0.007963

            Sensitivity : 0.9367
            Specificity : 0.5128
         Pos Pred Value : 0.7957
         Neg Pred Value : 0.8000
             Prevalence : 0.6695
         Detection Rate : 0.6271
   Detection Prevalence : 0.7881
      Balanced Accuracy : 0.7248

       'Positive' Class : D
```

COMENTARIO:

En esta parte se aplica la misma metdología de ridge con la única diferencia que el valor de alpha cambia a 1.

NAIVE BAYES

```r
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 58 17
         N 14 29

               Accuracy : 0.7373
                 95% CI : (0.6483, 0.814)
    No Information Rate : 0.6102
    P-Value [Acc > NIR] : 0.002563

                  Kappa : 0.4412

 Mcnemar's Test P-Value : 0.719438

            Sensitivity : 0.8056
            Specificity : 0.6304
         Pos Pred Value : 0.7733
         Neg Pred Value : 0.6744
             Prevalence : 0.6102
         Detection Rate : 0.4915
   Detection Prevalence : 0.6356
      Balanced Accuracy : 0.7180

       'Positive' Class : D
```

COMENTARIO: `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.

```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])
```

COMENTARIO:

En la primera línea se busca el valor de lambda más pequeño en el modelo ajustado. Este debe ser mayor o igual al valor de lambda óptimo seleccionado durante la sintonización (**`model$bestTune$lambda`**).

Este código extrae los coeficientes correspondientes al valor de lambda utilizado en el modelo final, y los almacena en un dataframe llamado **`featsele`**. Esto para analizar y visualizar los coeficientes de las variables seleccionada.

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

```
[1] "(Intercept)"              "Glucose"
[3] "SkinThickness"            "BMI"
[5] "DiabetesPedigreeFunction" "Age"
```

COMENTARIO: Este código devuelve los nombres de las filas que representan las variables en el dataframe **featsele** donde los coeficientes del modelo final son diferentes de cero.

```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 58 15
         N 14 31

               Accuracy : 0.7542
                 95% CI : (0.6665, 0.8288)
    No Information Rate : 0.6102
    P-Value [Acc > NIR] : 0.0006837

                  Kappa : 0.4814

 Mcnemar's Test P-Value : 1.0000000

            Sensitivity : 0.8056
            Specificity : 0.6739
         Pos Pred Value : 0.7945
         Neg Pred Value : 0.6889
             Prevalence : 0.6102
         Detection Rate : 0.4915
```

```
   Detection Prevalence : 0.6186
      Balanced Accuracy : 0.7397

       'Positive' Class : D
```

COMENTARIO:

Se puede observar que este modelo tiene una accuracy del 85%

```r
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: 247, 246, 246, 247, 246, 247, ...
Resampling results across tuning parameters:

  k    Accuracy   Kappa
    5  0.7285714  0.3183171
    7  0.7361993  0.3290307
    9  0.7130952  0.2597265
   11  0.7276455  0.2855763
   13  0.7360670  0.3140850
   15  0.7423280  0.3309352
   17  0.7423280  0.3329307
   19  0.7496032  0.3474711
   21  0.7567901  0.3587690
   23  0.7578924  0.3560087
```

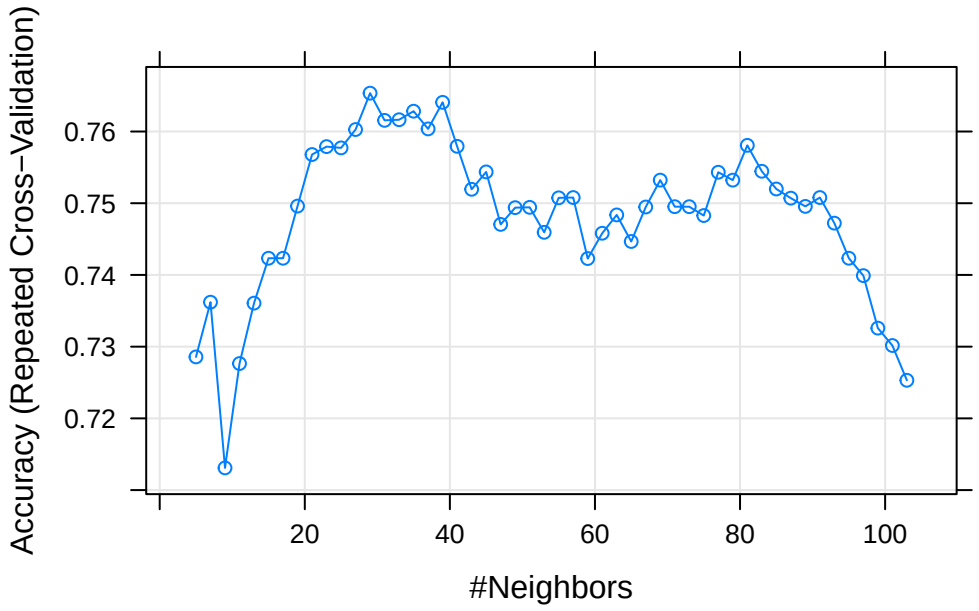
```
  25  0.7577160  0.3540226
  27  0.7602734  0.3596515
  29  0.7653439  0.3737017
  31  0.7615520  0.3651775
  33  0.7616402  0.3639096
  35  0.7628307  0.3677007
  37  0.7603616  0.3635632
  39  0.7640653  0.3727438
  41  0.7579365  0.3558416
  43  0.7519400  0.3361629
  45  0.7543651  0.3430265
  47  0.7470459  0.3190840
  49  0.7493827  0.3286858
  51  0.7494268  0.3279204
  53  0.7459436  0.3163010
  55  0.7507496  0.3261657
  57  0.7507937  0.3259157
  59  0.7422840  0.2990622
  61  0.7458113  0.3048454
  63  0.7483686  0.3030758
  65  0.7446649  0.2951321
  67  0.7494709  0.3054904
  69  0.7532187  0.3166839
  71  0.7495150  0.3021275
  73  0.7495150  0.3021275
  75  0.7482804  0.2998636
  77  0.7543210  0.3101913
  79  0.7532187  0.3088780
  81  0.7580688  0.3184812
  83  0.7544533  0.3050653
  85  0.7519841  0.2920728
  87  0.7507055  0.2875717
  89  0.7495591  0.2829499
  91  0.7507937  0.2833727
  93  0.7472222  0.2718725
  95  0.7423280  0.2472452
  97  0.7399030  0.2293441
  99  0.7325838  0.1986528
 101  0.7301587  0.1846034
 103  0.7253086  0.1585248

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

COMENTARIO: Se muestran los resultados del ajuste del modelo para diferentes valores de k (número de vecinos más cercanos considerados). Se informa la precisión y el coeficiente kappa para cada valor de k probado.

La métrica utilizada para seleccionar el modelo óptimo fue la precisión. El modelo con la precisión más alta se seleccionó como el modelo final.

El valor de k seleccionado para el modelo final fue 19, lo que significa que se consideraron los 19 vecinos más cercanos para realizar las predicciones.

```
plot(knnFit)
```




```
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 65 22
          N  7 24

                Accuracy : 0.7542
                  95% CI : (0.6665, 0.8288)
     No Information Rate : 0.6102
     P-Value [Acc > NIR] : 0.0006837

                   Kappa : 0.4511

  Mcnemar's Test P-Value : 0.0093296

             Sensitivity : 0.9028
             Specificity : 0.5217
          Pos Pred Value : 0.7471
          Neg Pred Value : 0.7742
              Prevalence : 0.6102
          Detection Rate : 0.5508
    Detection Prevalence : 0.7373
       Balanced Accuracy : 0.7123

        'Positive' Class : D
```

COMENTARIO: Un valor de kappa más cercano a 1 indica un acuerdo sustancial entre las predicciones del modelo y las clases reales. En este caso el valor está alejado a 1.

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
library(caret)
datos <- read.csv("./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
```

```r
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
```

COMENTARIO:

Antes de ajustar el modelo PLS, los predictores se preprocesan mediante centrado y escalado. El centrado consiste en restar la media de cada variable predictora, mientras que el escalado consiste en dividir por la desviación estándar. Este paso de estandarización garantiza que todas las variables estén en una escala similar, lo que evita que una variable en particular domine el análisis.

El rendimiento del modelo se evalúa en función de distintos parámetros de ajuste, concretamente el número de componentes. Las métricas de evaluación utilizadas son ROC, sensibilidad y especificidad. El ROC se utiliza para problemas de clasificación binaria y proporciona una medida de la capacidad del modelo para discriminar entre las dos clases. El modelo óptimo se selecciona en función del mayor valor ROC, y el número correspondiente de componentes se elige como valor final del modelo.

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).

Si tuneamos lambda

```
datos <- read.csv("./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
```

COMENTARIO: Se crea un vector llamado "lambda" que contiene valores secuenciales de 0 a 50 con incrementos de 0.1.

Este código 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("./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,]
```

```r
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
```

COMENTARIO:

Se seleccionan las columnas 2, 5, 7 y 8 de los datos de entrenamiento como variables predictoras, y la columna "Outcome" como la variable objetivo. Se utiliza "pls" como el algoritmo de PLS-DA y se especifica que se realicen 10 ajustes para seleccionar el número óptimo de componentes utilizando la métrica ROC. Finalmente evalúa el rendimiento del modelo utilizando la matriz de confusión y se obtienen otras estadísticas relacionadas con la clasificación binaria.

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.

Finalmente podríamos hacer un análisis de la varianza multivariante

```
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.