Trabajo de evaluación de Machine Learning I sobre la metodologia CART y reduccion de la dimensionalidad.

Álvaro Sánchez Castañeda

3 de mayo de 2017

Ejercicio 1:

Leer el fichero "Crimen.dat", que contiene el total de delitos por cada 100.000 habitantes para cada uno de lo estados de EEUU más el distrito de Columbia (Año 1986). Aplicar y comparar tres técnicas de análisis de conglomerados (una de tipo jerárquico, otra de tipo partición y el método basado en mixturas de normales multivariantes.

Lectura y preprocesado de los datos.

Comenzemos pues cargando una libreria de tecnicas de analisis de conglomerados y los datos a los que vamos a aplicar dichas tecnicas. Comprovemos a su vez si tenemos valores perdidos en estos datos.

```
library(cluster)
datos=read.csv("C:\\Users\\AlvaroSanchez91\\Desktop\\Master Big Data Sevilla\\ML1 Machine Learning I\\E
sum(is.na(datos))
```

```
## [1] 0
```

Vemos que no tenemos ningun valor perdido. El siguiente paso que podemos tomar es tipificar las variables, es recomendable hacer esto con este tipo de tecnicas.

```
print(apply(datos,2,var))
```

```
##      Asesinato         Abusos         Atraco      AgresiÃ³n Robo_domicilio
##       23.20215      212.31228    18993.37020    22004.31294   177912.83373
##          Hurto Robo_vehÃ.culo
##   582812.83843    50007.37490
```

```
print(apply(datos,2,mean))
```

```
##      Asesinato         Abusos         Atraco      AgresiÃ³n Robo_domicilio
##        7.25098       34.21765      154.09804      283.35294     1207.07843
##          Hurto Robo_vehÃ.culo
##     2941.96078      393.84314
```

```
datos2=scale(datos,center = TRUE, scale = TRUE)
print(apply(datos2,2,var))
```

```
##      Asesinato         Abusos         Atraco      AgresiÃ³n Robo_domicilio
##              1              1              1              1              1
##          Hurto Robo_vehÃ.culo
##              1              1
```

```
print(apply(datos2,2,mean))
```

```
##      Asesinato         Abusos         Atraco      AgresiÃ³n Robo_domicilio
##  -5.704817e-18  -5.988171e-17   1.052314e-17  -1.462759e-16   1.663049e-16
##          Hurto Robo_vehÃ.culo
```

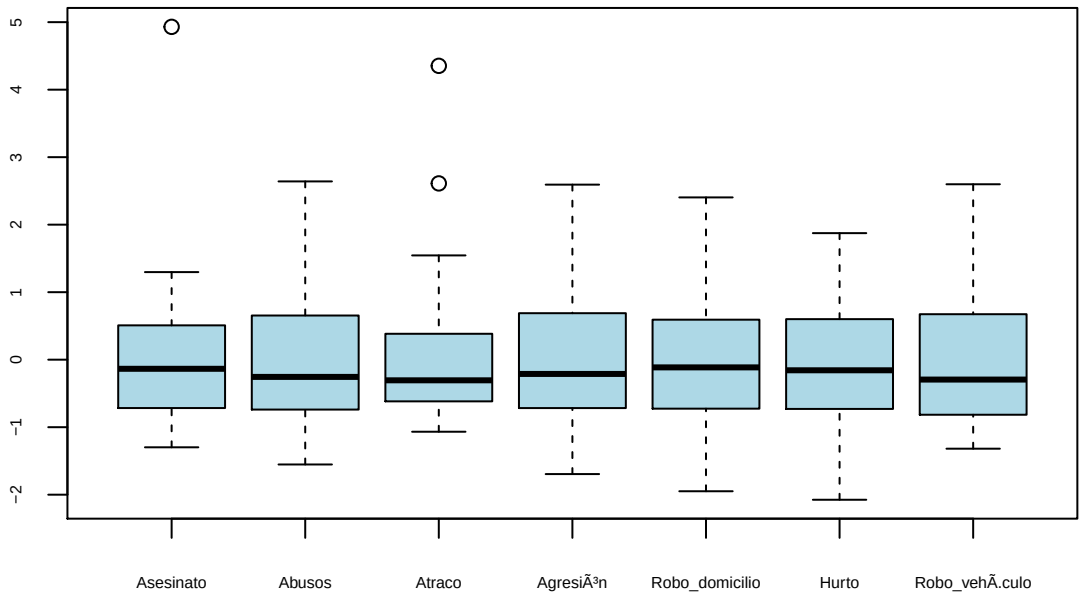
```
##  -3.937992e-17   1.255703e-16
```

En datos2 hemos guardado los datos con varianza uno y media cero.

Breve analisis exploratorio.

Buscaremos valores atipícos en nuestro conjunto de datos, tambibién pueden afectar a algunos metodos.

```
boxplot(datos2, axes=T,cex.axis=0.5,col='lightblue')
```




```
atipicos=data.frame()
outliers=c()
for(i in 1:7){
  out=boxplot(datos2[,i], plot=F)$out
  outliers=c(outliers,names(out))
}
print(outliers)
```

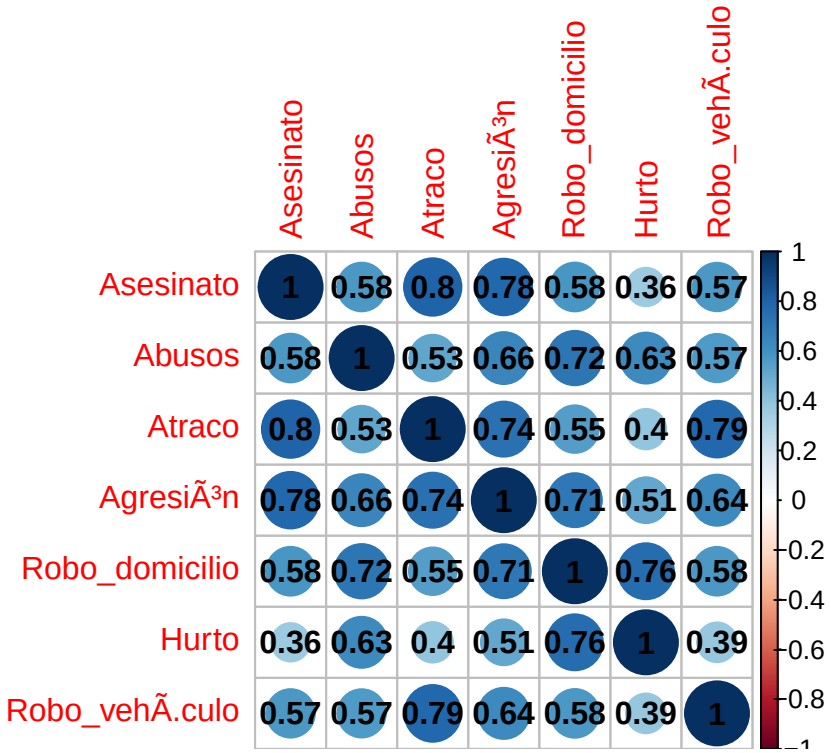
```
## [1] "DC" "NY" "DC"
```

Tenemos que DC y NY tienen valores atipicos, es mas, DC tiene valores atipicos para dos variables distintas. Tal vez sería razonable apartar a DC de nuestro estudio, y considerarlo como un cluster en si mismo. Para esto crearemos una nueva tabla de datos sin DC, y compararemos los resultados de trabajar con y sin el.

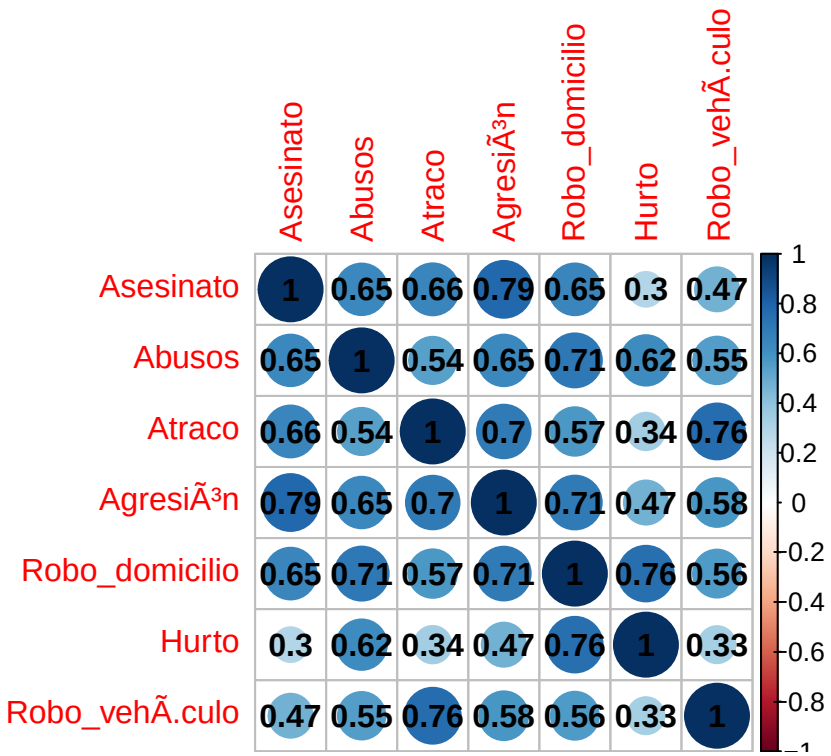
```
datos_sin_out=datos[rownames(datos) != 'DC', ]#Sin tipificar
datos_sin_out2=scale(datos_sin_out,center = TRUE, scale = TRUE)#Tipificando
```

Veamos las correlacciones entre las variables de datos sin tipificar, tanto teniendo o no en cuenta a DC.

```
library(corrplot)
corrplot(cor(datos),addCoef.col="black")
```




```
corrplot(cor(datos_sin_out),addCoef.col="black")
```



Los resultados obtenidos parecen razonables. Observemos a su vez que aunque alguna correlacion parece algo elevada, no lo es tanto como para plantearse eliminar variables directamente.

Veamos a continuación que todas nuestras variables son reales, lo cual nos permitirá posteriormente hacer analisis de conglomerados mediante la distancia euclídea.

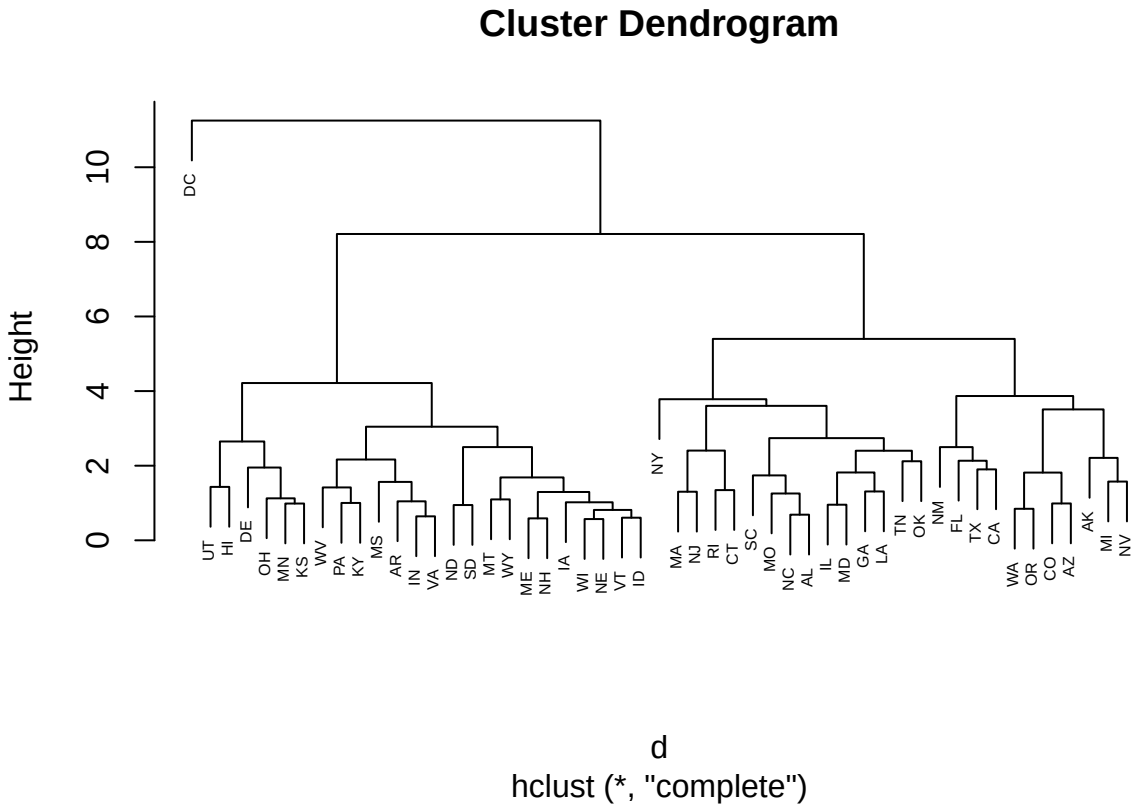
```
head(datos)
```

```
##    Asesinato Abusos Atraco AgresiÃ³n Robo_domicilio Hurto Robo_vehÃ.culo
## ME       2.0   14.8     28       102            803  2347            164
## NH       2.2   21.5     24        92            755  2208            228
## VT       2.0   21.8     22       103            949  2697            181
## MA       3.6   29.7    193       331           1071  2189            906
## RI       3.5   21.4    119       192           1294  2568            705
## CT       4.6   23.8    192       205           1198  2758            447
```

Metodos jerarquicos sobre datos estandarizados.

Metodo aglomerativo.

```
d=dist(datos2)#Calculamos las distancias entre individuos.
jer=hclust(d)#Aplicamos un algoritmo aglomerativo con un metodo de unión simple.
plot(jer, cex=0.5)
```



Lo primero que observamos es que DC (el valor que hemos tomado como atipico) lo separa drasticamente del resto de valores, y solo se agrupa en el ultimo paso. Necesitamos decidir si hay un numero apropiado de clusters para estos datos.

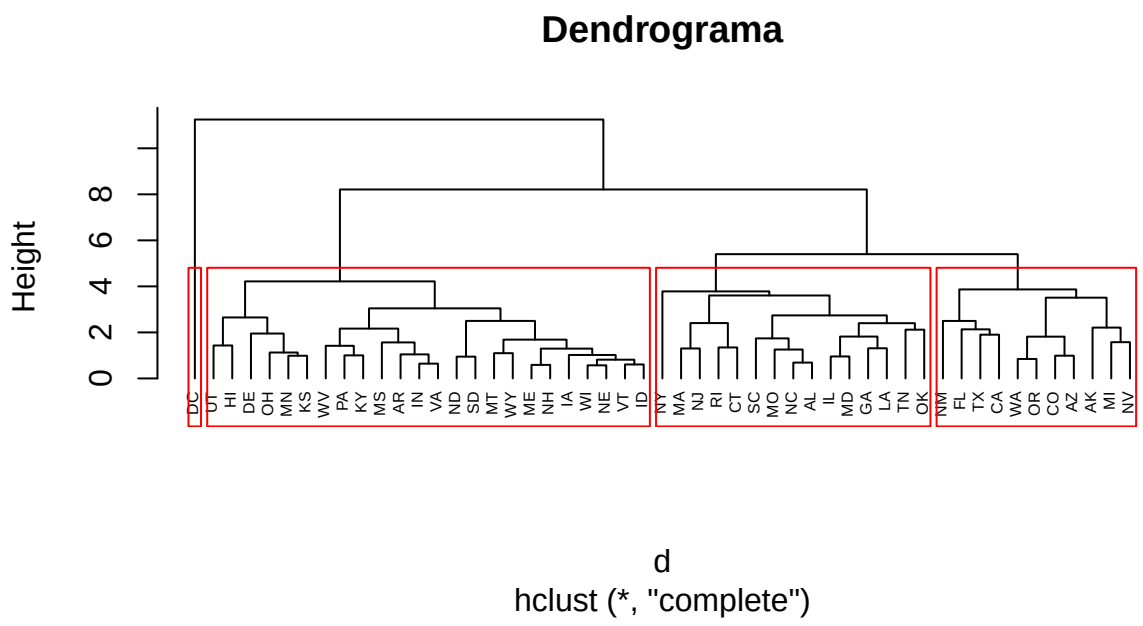
En coeficientes guardaremos para cado paso del metodo jerarquico la altura (height) del paso anterior, dividida entre la altura de el paso actual. Esto puede ayudar a elegir un numero de clusters conveniente.

```
coeficientes=c(0)
for (i in 2:length(jer$height)){
  coeficientes=append(coeficientes,jer$height[i-1]/jer$height[i])
}
print(tail(cbind(jer$merge,jer$height,coeficientes)))
```

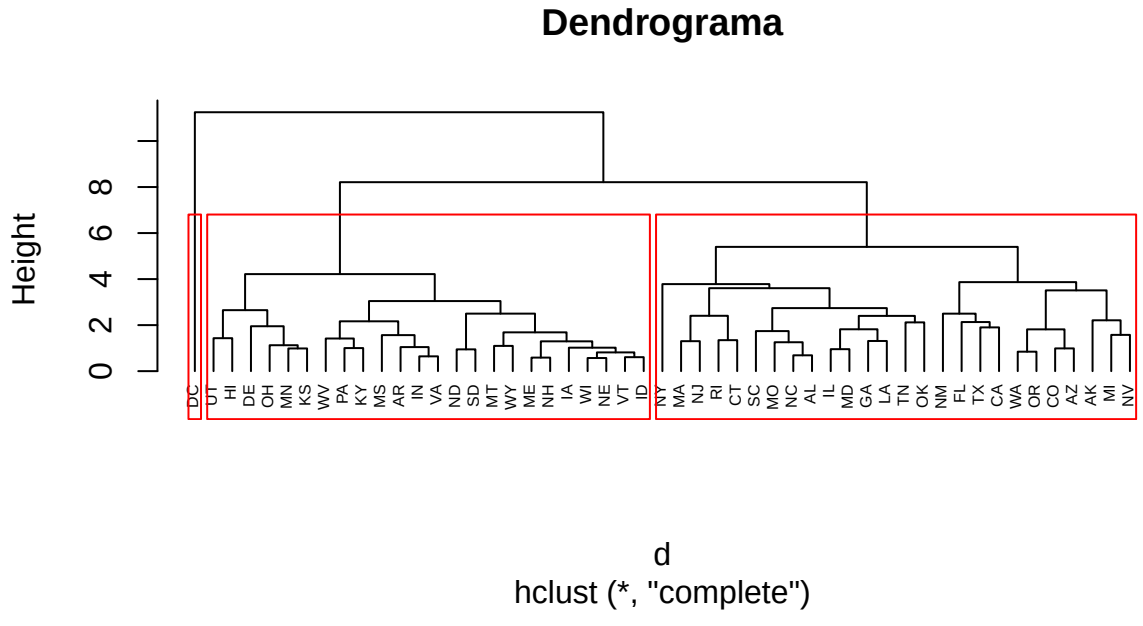
```
##                         coeficientes
## [45,]  -7 44  3.782290    0.9529714
## [46,]  38 43  3.867800    0.9778919
## [47,]  40 42  4.215970    0.9174163
## [48,]  45 46  5.399200    0.7808508
## [49,]  47 48  8.210862    0.6575681
## [50,] -24 49 11.251023    0.7297880
```

En vista del dendograma y de los datos, parece conveniente tomar tres o cuatro clusters.

```
plot(jer,main="Dendrograma", cex=0.5,hang = -1)
rect.hclust(jer, k=4,border="red")
```

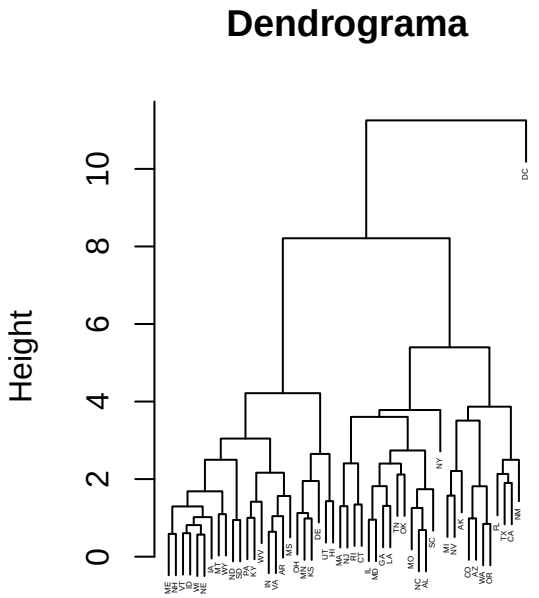


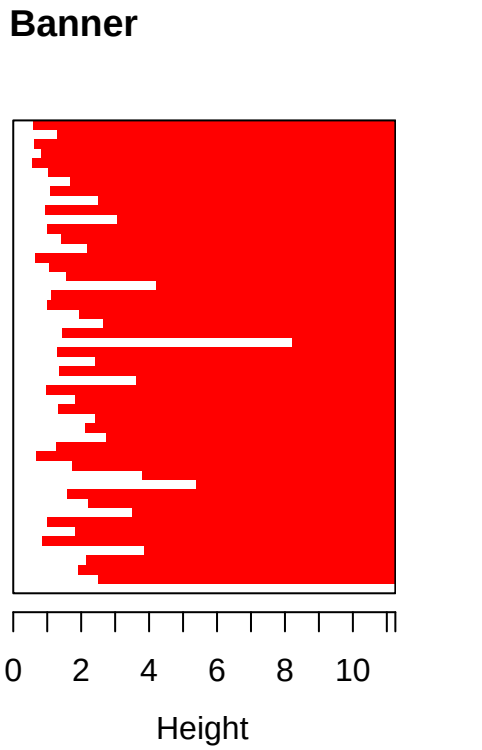

```
plot(jer,main="Dendrograma", cex=0.5,hang = -1)
rect.hclust(jer, k=3,border="red")
```



Hagamos el mismo proceso pero utilizando otra función de R, agnes. Esta a su vez proporciona un grafico de las alturas (banner), bastante util.

```
jer1=agnes(d, diss = T,method = 'complete', stand = F)

par(mfrow=c(1,2))
plot(jer1,which.plots=c(2),main="Dendrograma",cex=0.25)
plot(jer1,which.plots=c(1),main="Banner")
```



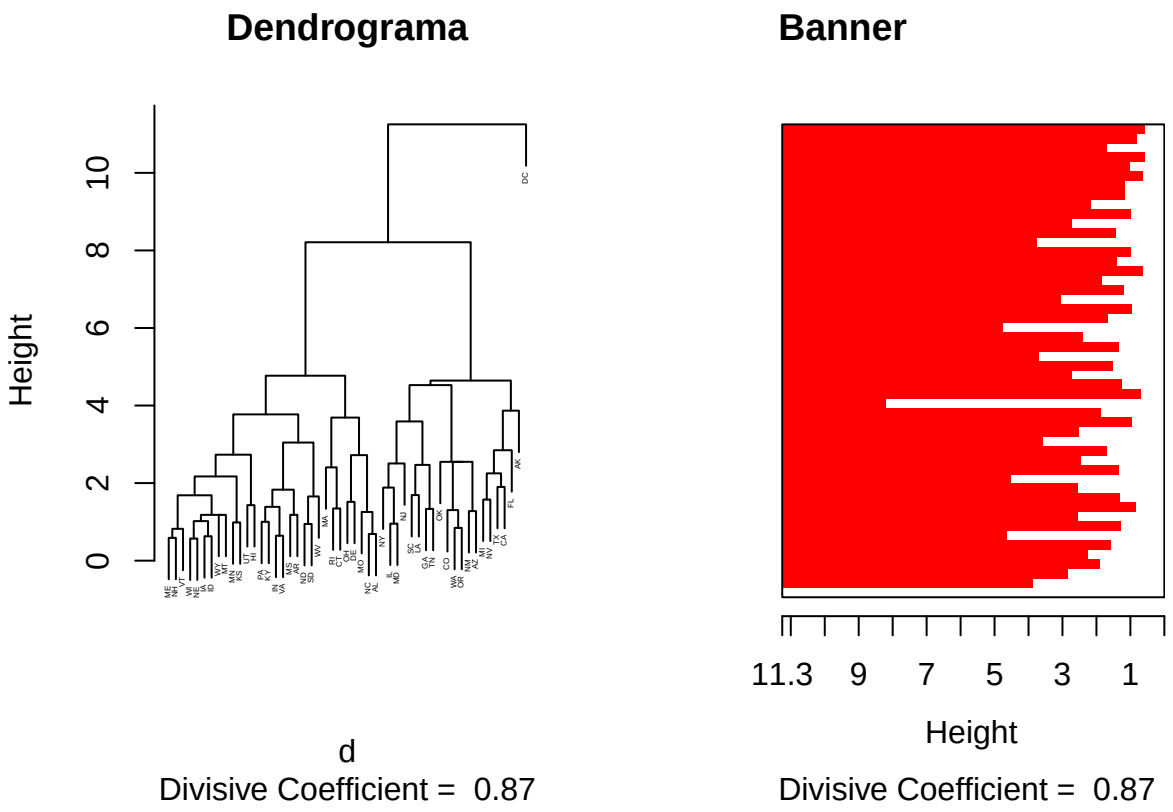



```
par(mfrow=c(1,1))


h=sort(jer1$height)
coeficientes1=c(0)
for (i in 2:length(jer1$height)){
  coeficientes1=append(coeficientes1,h[i-1]/h[i])
}
print(tail(cbind(jer1$merge,h,coeficientes1)))
```

```
##                      h coeficientes1
## [45,] 44  -7  3.782290     0.9529714
## [46,] 43  38  3.867800     0.9778919
## [47,] 42  40  4.215970     0.9174163
## [48,] 45  46  5.399200     0.7808508
## [49,] 47  48  8.210862     0.6575681
## [50,] 49 -24 11.251023     0.7297880
```

Metodo divisivo.

```
jer2=diana(d, diss = T)

par(mfrow=c(1,2))
plot(jer2,which.plots=c(2), cex=0.25,main="Dendrograma")
plot(jer2,which.plots=c(1),main="Banner")
```




```
par(mfrow=c(1,1))

h2=sort(jer2$height)
coeficientes2=c(0)
for (i in 2:length(jer2$height)){
  coeficientes2=append(coeficientes2,h2[i-1]/h2[i])
}
print(tail(cbind(jer2$merge,h2,coeficientes2)))
```

```
##                     h2 coeficientes2
## [45,] 40 -50  3.867800     0.9752058
## [46,] 42  37  4.526868     0.8544097
## [47,] 46  45  4.643887     0.9748015
## [48,] 44  43  4.769863     0.9735891
## [49,] 48  47  8.210862     0.5809211
## [50,] 49 -24 11.251023     0.7297880
```

Hemos repetido los mismos pasos que en el algoritmo aglomerativo y, esta vez, parece que tres clusters es la mejor opción.

Métodos de partición.

Metodo de K-medias.

```
kmedias=kmeans(datos2,3)#inestable ante el valor atipico

#centros de los clusters
kmedias$centers
```

```
##     Asesinato      Abusos      Atraco  AgresiÃ³n Robo_domicilio      Hurto
## 1 -0.60433142 -0.85530840 -0.71760700 -0.8625740    -0.91235428 -0.6229332
## 2 -0.06378228  0.02152249  0.03738277  0.1092741     0.07153099 -0.2489100
## 3  0.87380966  1.11745387  0.91693437  1.0335396     1.14017264  1.0960816
##   Robo_vehÃ.culo
## 1     -0.8543079
## 2      0.3377639
## 3      0.7787957
```

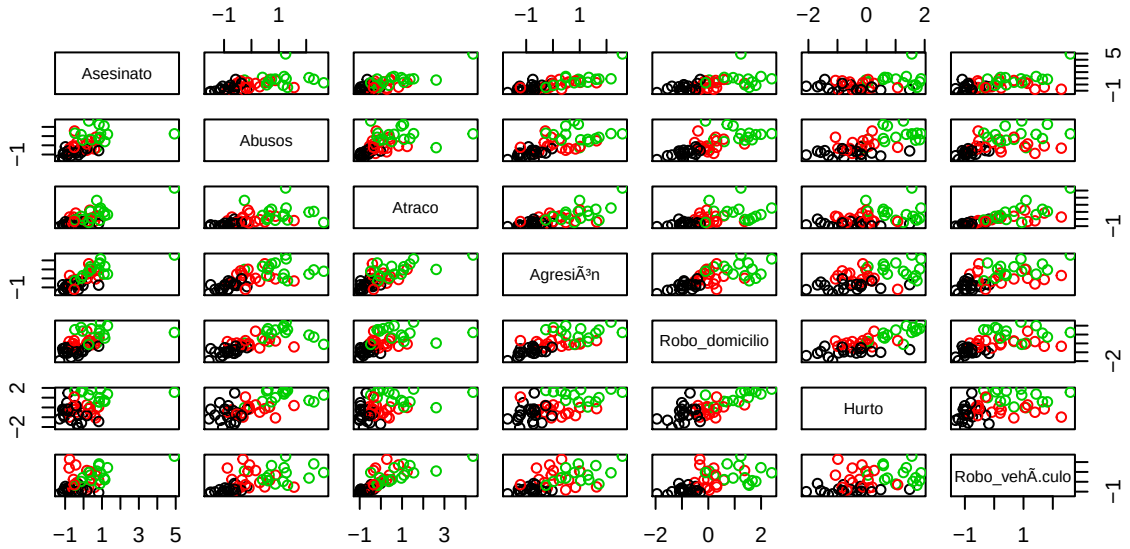
```
#Vector de n componentes indicando el cluster al que se asigna  cada punto
kmedias$cluster
```

```
## ME NH VT MA RI CT NY NJ PA OH IN IL MI WI MN IA MO ND SD NE KS DE MD DC VA
##  1  1  1  2  2  2  3  2  1  2  1  2  3  1  1  1  2  1  1  1  2  2  3  3  1
## WV NC SC GA FL KY TN AL MS AR LA OK TX MT ID WY CO NM AZ UT NV WA OR CA AK
##  1  2  2  2  3  1  2  2  1  1  3  2  3  1  1  1  3  3  3  1  3  3  3  3  3
## HI
##  2
```

```
#Número de puntos en cada cluster
kmedias$size
```

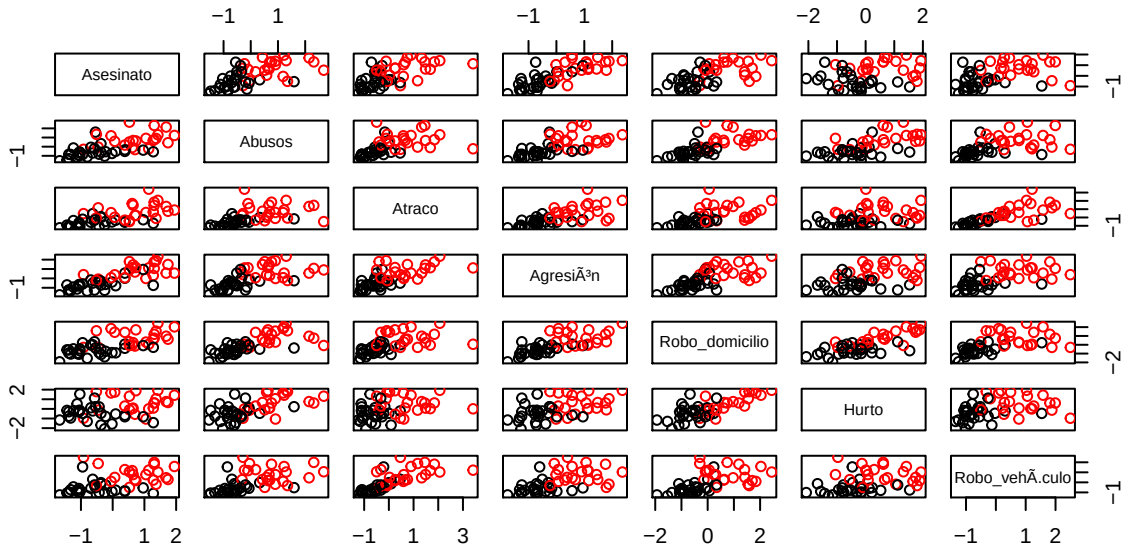
```
## [1] 20 16 15
```

```
plot(as.data.frame(datos2),col=c(1,2,3)[kmedias$cluster])
```



El anterior algoritmo es inestable ante valores atipicos, por lo que sería recomendable utilizar el algoritmo de k-medioides o eliminar DC del análisis. Veamos como quedarian divididos los datos si tomamos k igual a dos y no consideramos DC.

```
kmedias2=kmeans(datos_sin_out2,2)
plot(as.data.frame(datos_sin_out2),col=kmedias2$cluster)
```

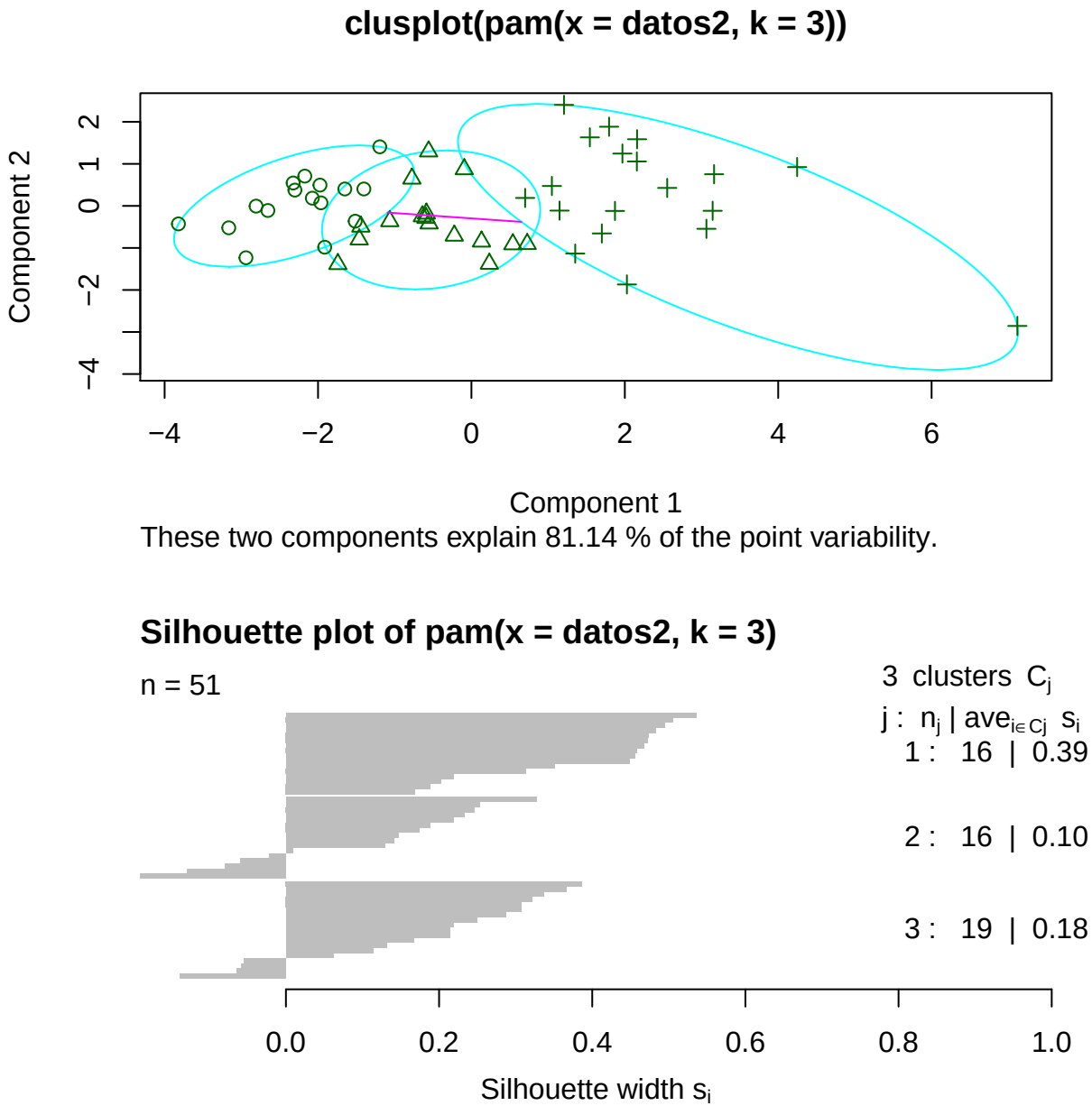


Si se repite varias veces el proceso de k-medias descrito anteriormente (con DC incluido en los datos), llegamos a divisiones muy distintas (algunas de ellas separan a DC). Se muestra mas estable el algoritmo sin considerar DC, pero otra solución pasa por aplicar el algoritmo de k-medioides, que es mas estable ante estas situaciones.

Método de k-medioides.

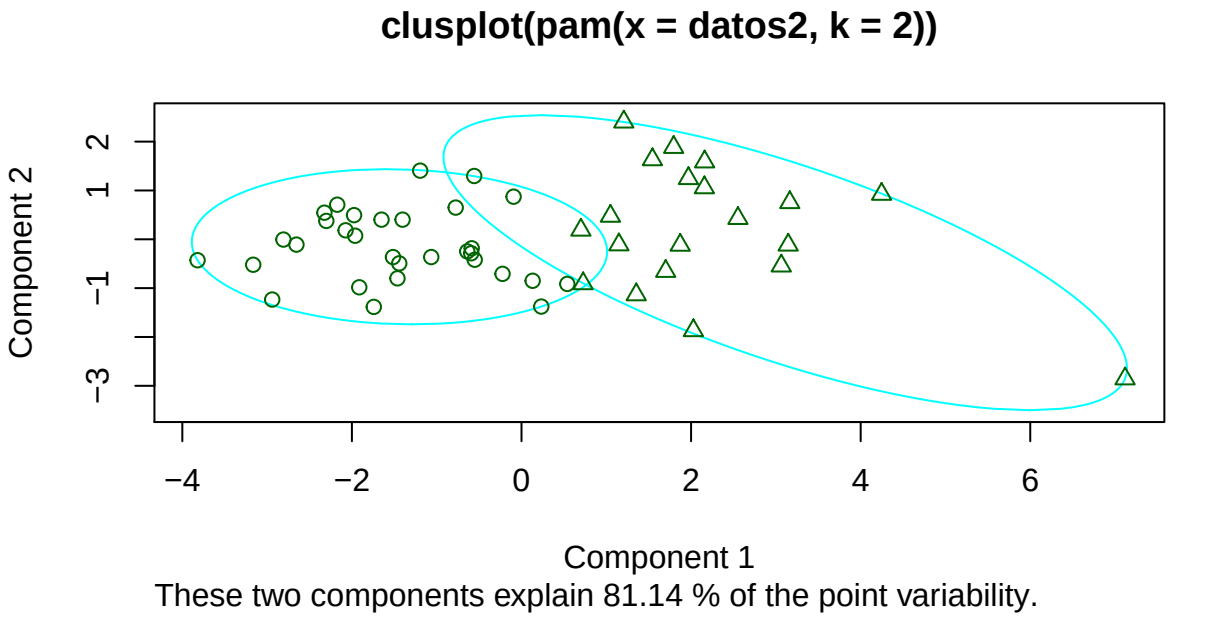
Probemos a construir los clusters mediante este metodo con K igual a tres.

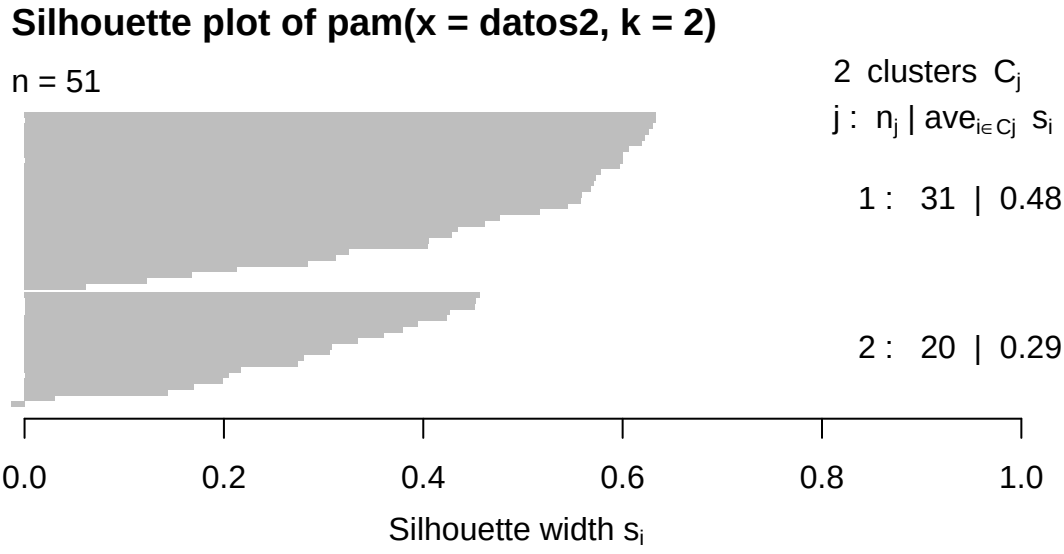
```
kmedioides=pam(datos2,3)
plot(kmedioides)
```



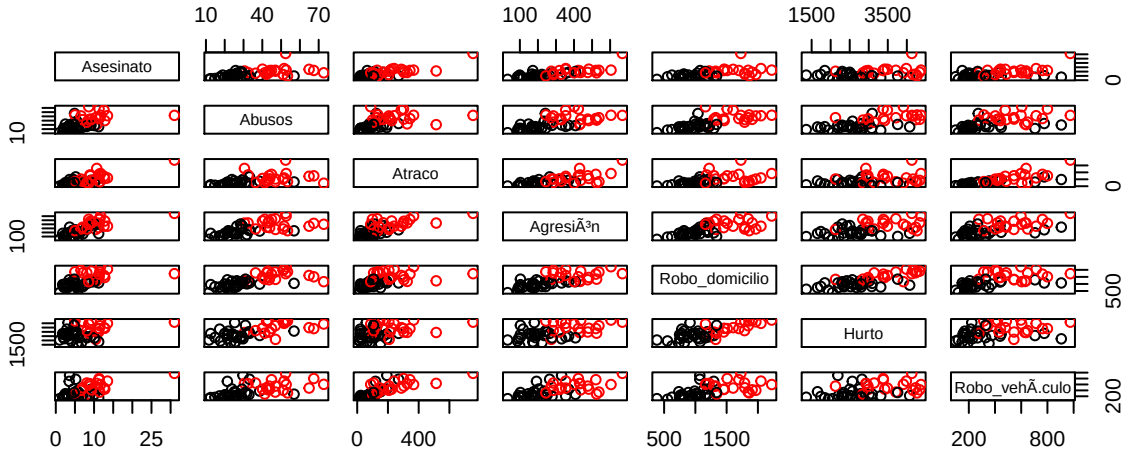
Este metodo proporciona una medida de silueta, la cual es bastante mala para k igual a tres. Tomemos k igual a dos ,lo cual es razonable teniendo en cuenta que en el proceso jerarquico DC quedaba separado en un único cluster.

```
kmedioides=pam(datos2,2)
plot(kmedioides)
```



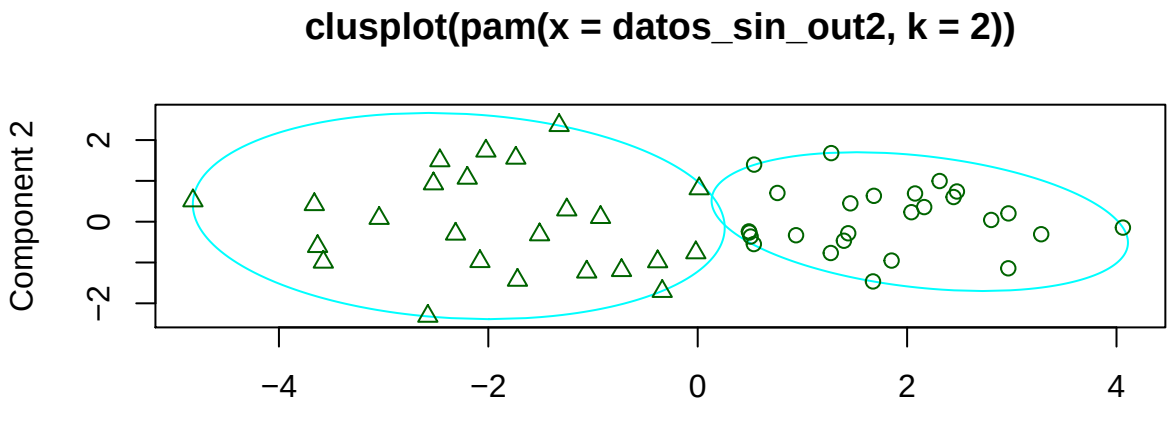


```
plot(datos,col=c(1,2,3,4)[kmedioides$cluster])
```

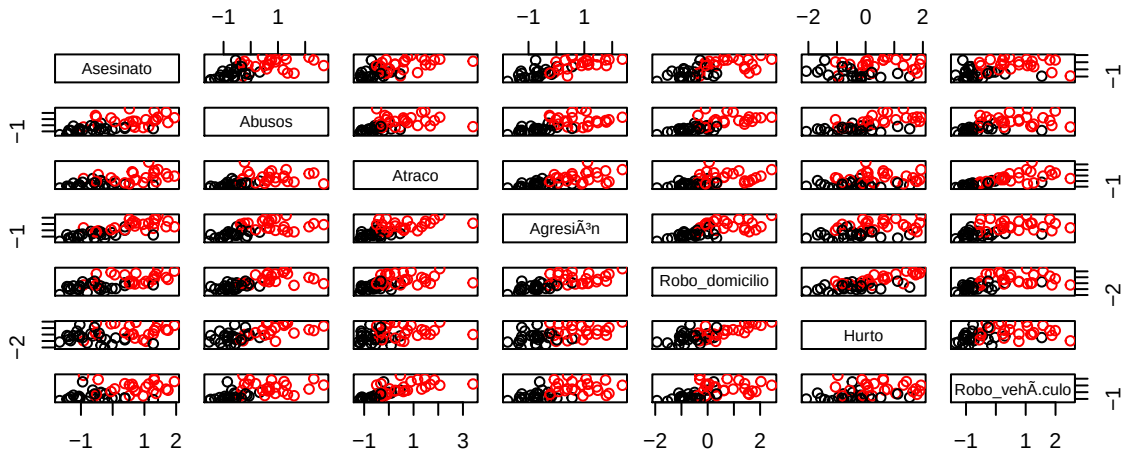


Podemos probar, al igual que hicimos con el metodo de k-medias, a eliminar DC del analisis de conglomerados. Veamos si de este modo mejora la silueta media.

```
kmedioides2=pam(datos_sin_out2,2)
plot(kmedioides2)
```



```
plot(as.data.frame(datos_sin_out2),col=kmedioides2$cluster)
```



Curiosamente, la medida de silueta empeora si no consideramos DC en el analisis mediante k-medioides. Esta vale 0.39, mientras que la anterior, considerando DC, valia 0.4. En cualquier caso son practicamente iguales, aunque no clasifican igual a los elementos distintos de DC.

```
table(cbind(data.frame(kmedioides$clustering[names(kmedioides$clustering)!='DC']),
            data.frame(kmedioides2$clustering)))
```

```
##                                                    kmedioides2.clustering
## kmedioides.clustering.names.kmedioides.clustering......DC..  1  2
##                                                             1 26  5
##                                                             2  0 19
```

Habria que decantarse por una de las dos opciones, ya que DC a aparecido tan diferenciado anteriormente, apesar de tener una peor silueta yo me quedaria con esta ultima división en la que no lo hemos considerado. De este modo podriamos interpretar que tenemos tres grupos, uno formado únicamente por DC, y los otros dos los dados en esta última ejecución de k-medioides.

Método basado en mixturas de normales.

Carguemos una librería que contiene estos procedimientos y apliquemoslos sobre los datos tipificados icluyendo y sin incluir a DC.

```
library(mclust)
```

```
## Warning: package 'mclust' was built under R version 3.3.3
```

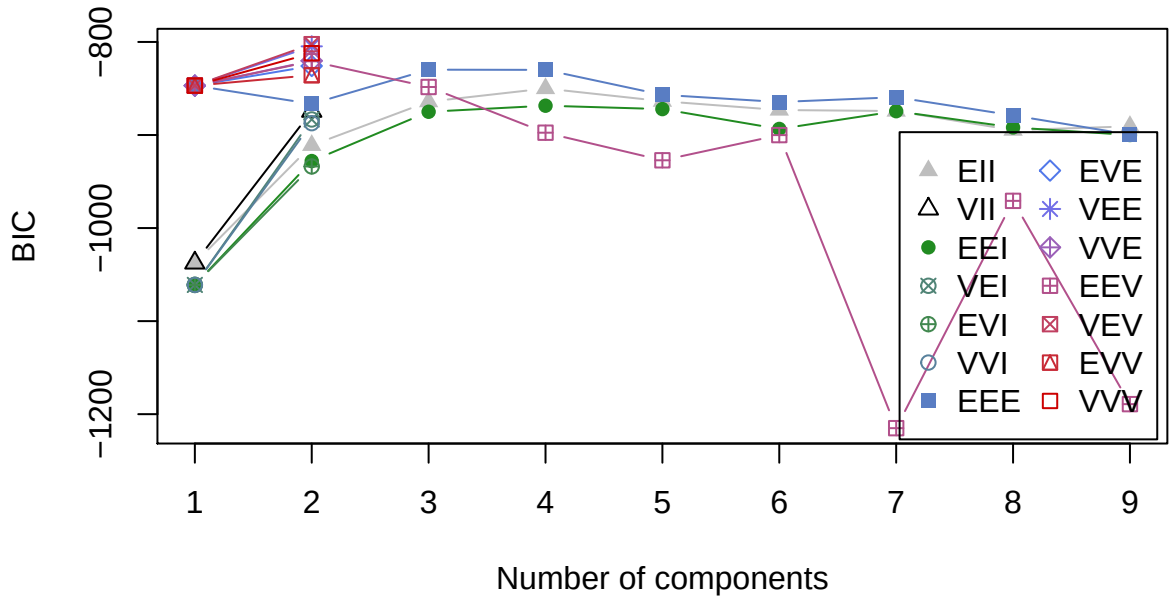
```
## Package 'mclust' version 5.2.3
```

```
## Type 'citation("mclust")' for citing this R package in publications.
```

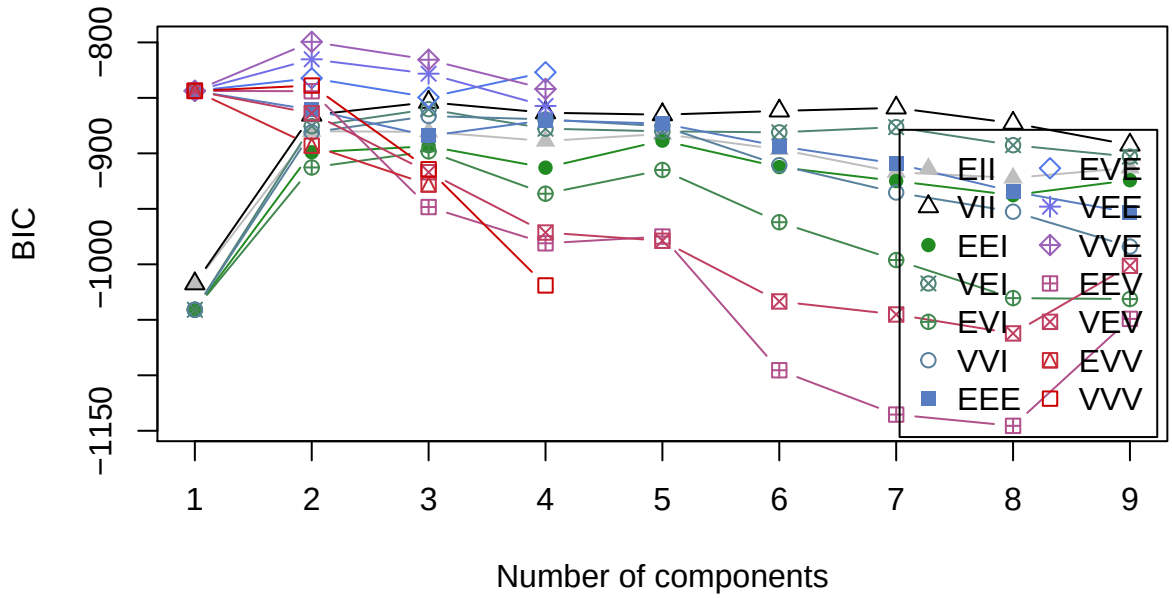
```
clus_Mix=Mclust(datos2)
clus_Mix_sin_out=Mclust(datos_sin_out2)
```

Este comando ejecuta distintos metodos para modelar las distribuciones normales (suponiendo igual tamaño de las normales, igual orientación, etc) y tambien toma varios k distintos. Para cada uno de los resultados obtenidos nos devuelve un criterio de incertidumbre BIC que podemos usar para decidir que partición tomar.

```
plot(clus_Mix,what = "BIC")
```




```
plot(clus_Mix_sin_out,what = "BIC")
```



Vemos que en general, para todos los modelos, parece mejor considerar DC en el estudio (aparentemente, las

escalas son distintas). Si solo comparamos los dos mejores resultados para cada conjunto de datos, obtenemos valores BIC similares.

```
summary(clus_Mix)
```

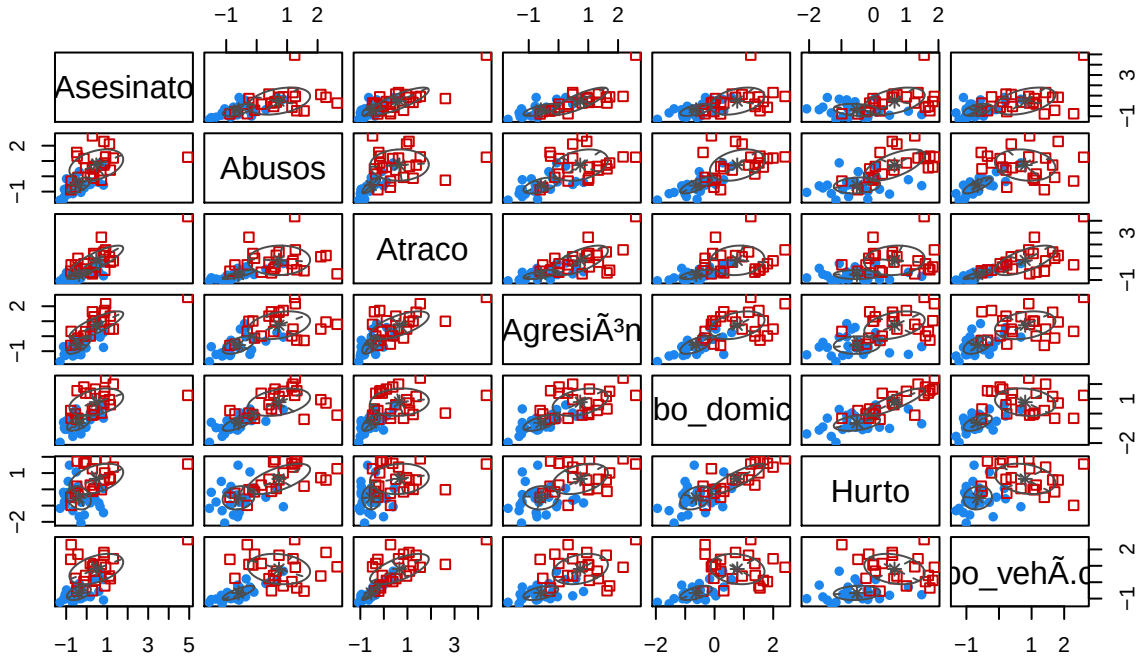
```
## ----------------------------------------------------
## Gaussian finite mixture model fitted by EM algorithm
## ----------------------------------------------------
##
## Mclust VEV (ellipsoidal, equal shape) model with 2 components:
##
##  log.likelihood  n df       BIC      ICL
##       -273.4041 51 65 -802.3769 -803.098
##
## Clustering table:
##  1  2
## 28 23
```

```
summary(clus_Mix_sin_out)
```

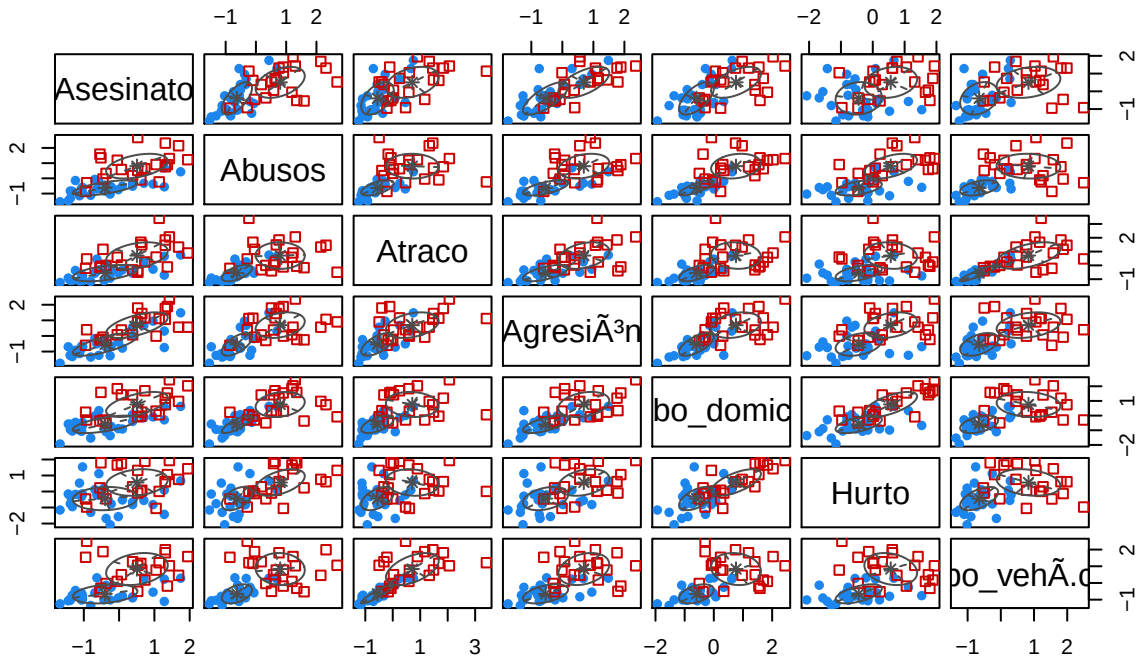
```
## ----------------------------------------------------
## Gaussian finite mixture model fitted by EM algorithm
## ----------------------------------------------------
##
## Mclust VVE (ellipsoidal, equal orientation) model with 2 components:
##
##  log.likelihood  n df       BIC      ICL
##       -301.9503 50 50 -799.5017 -801.383
##
## Clustering table:
##  1  2
## 28 22
```

Sería ligeramente mejor el procedimiento habiendo tomado en cuenta a DC (al igual que pasaba con el metodo de k-medioides). Vemos que en ese caso el mejor metodo usa normales "elipsoidales" de "igual forma". Veamos graficamente como quedarian los datos clasificados.

```
plot(clus_Mix,what = "classification")
```




```
plot(clus_Mix_sin_out,what = "classification")
```



Comparación de distintos metodos.

Por ultimo, compararemos las particiones obtenidas mediante un metodo jerarquico aglomerativo, uno de partición y uno basado en mixturas de normales. Se ha dejado fuera del estudio a DC, aunque vistos los resultados obtenidos anteriormente, aunque sea un valor muy diferenciado no esta claro si devemos apartarlo.

Cuando aplicamos los métodos jerarquicos, no fue necesario ver que pasaba si apartabamos a DC, dado que este quedaba siempre separado y el dendograma restante quedaria igual. Ahora si que haremos esto simplemente para obtener la partición con k igual a dos y sin DC (también se podría eliminar directamente). Tomarémos un método jerarquico aglomerativo.

```
d_sin_out=dist(datos_sin_out2)#distancia entre elementos
jer0_sin_out=hclust(d_sin_out)
```

Comparemos ahora como quedan los datos divididos usando este ultimo metodo con la partición realizada mediante mixturas, y con la partición realizada usando k-medioides. Todo esto sin considerar a DC y tomando dos clusters.

```
table(cbind(data.frame(clus_Mix_sin_out$classification),data.frame(kmedioides2$clustering)))
```

```
##                                kmedioides2.clustering
## clus_Mix_sin_out.classification  1  2
##                               1 24  4
##                               2  2 20
```

Da la casualidad que se clasifican utilizando las mismas etiquetas (es decir, hay muchos elementos en la diagonal de la tabla). Podría ser, que al ser aprendizaje no supervisado, tuviesemos clasificaciones parecidas pero con etiquetas distintas, en cuyo caso tendríamos valores grandes fuera de la diagonal. Esto no sucede, y es mas facil de interpretar la tabla, ademas vemos que se parecen mucho las clasificaciones.

Veamos ahora si se parecen las tres clasificaciones, comparando tambien con la obtenida mediante el método jerarquico.

```
clasificaciones=cbind(data.frame(clus_Mix_sin_out$classification),
                      data.frame(kmedioides2$clustering),cutree(jer0_sin_out, k = 2))
colnames(clasificaciones)=c('Mixturas','Kmedioides','Jerarquico')
table(clasificaciones)
```

```
## , , Jerarquico = 1
##
##         Kmedioides
## Mixturas  1  2
##        1 24  2
##        2  2  3
##
## , , Jerarquico = 2
##
##         Kmedioides
## Mixturas  1  2
##        1  0  2
##        2  0 17
```

Aparentemente tiene buena pinta, contemos cuantos coinciden.

```
clasificaciones_aux=clasificaciones[
  clasificaciones$Mixturas==clasificaciones$Kmedioides,]
clasificaciones_aux=clasificaciones_aux[
  clasificaciones_aux$Mixturas==clasificaciones_aux$Jerarquico,]
```

```r
dim(clasificaciones_aux)
```

```
## [1] 41  3
```

Efectivamente tenemos mucha coincidencia. Esto da cierta confianza, pues, de algun modo da lugar a pensar que los datos son "facilmente agrupables". Pero no es una medida fiable, solo algo intuitivo. A la hora de decantarse por una partición es mejor no usar un metodo jerarquico, lo razonable parece elejir la obtenida mediante mixturas o mediante k-medioides.

Ejercicio 2.

Acceder a los datos gironde la librería PCAmixdata. En los siguientes apartados seleccionar los registros completos si hay valores perdidos.

i) Realizar e interpretar un análisis de componentes principales (matriz de correlaciones) para gironde$employment.

ii) Realizar e interpretar un análisis de componentes principales para datos mixtos sobre la unión de gironde$employment y gironde$services.

iii) Aplicar procedimientos de selección de variables para construir modelos de regresión lineal donde income es la variable dependiente, sobre gironde$employment.

i) Análisis de comonentes principales para gironde$employment.

Lectura y preprocesado de datos.

Comenzemos cargando la librería y los datos gironde$employment para este apartado. Tambien veamos que estructura tienen estos datos.

```
library(PCAmixdata)
data('gironde')
datos_empleo=gironde$employment
head(datos_empleo)
```

```
##                     farmers tradesmen managers workers unemployed
## ABZAC                  1.98      3.68     3.97   38.25      13.60
## AILLAS                 5.23      5.23     1.96   21.57      15.03
## AMBARES-ET-LAGRAVE     0.10      4.38     5.56   35.98      18.23
## AMBES                  0.18      2.29     3.70   42.42      15.11
## ANDERNOS-LES-BAINS     0.30      3.80     8.19   18.65      13.04
## ANGLADE                3.13      5.63     1.25   39.37      16.87
##                     middleempl retired employrate   income
## ABZAC                     9.63   28.90      89.26 17670.60
## AILLAS                   14.38   36.60      90.88 19422.49
## AMBARES-ET-LAGRAVE       15.48   20.28      90.25 21047.07
## AMBES                     8.98   27.33      87.38 18014.52
## ANDERNOS-LES-BAINS       12.07   43.97      89.43 27147.48
## ANGLADE                   5.63   28.12      88.71 15897.99
```

Veamos si hay valores perdidos.

```
which(apply(apply(datos_empleo,2,is.na),1,sum)!=0)
```

```
##              BOSSUGAN SAINT-AVIT-DE-SOULEGE
##                    63                   369
```

```
datos_empleo2=datos_empleo[-which(apply(apply(datos_empleo,2,is.na),1,sum)!=0),]
dim(datos_empleo)
```

```
## [1] 542   9
```

```
dim(datos_empleo2)
```

```
## [1] 540   9
```

Hemos quitado los dos elementos que presentan valores perdidos.

Calculo de componentes principales.

Lo primero que podemos hacer en nuestro analisis de componentes principales, es ver que pinta tiene nuestra matriz de correlaciones.

```
library(corrplot)
corrplot(cor(datos_empleo2),addCoef.col="black")
```

En lugar de abordar directamente el problema diagonalizando esta matriz, podemos recurrir a una función de PCAmixdata para facilitar los calculos. Veamos a su vez un resumen de los resultados obtenidos.

```
acp<- princomp(datos_empleo2, cor = TRUE)#con cor=TRUE trabajamos con la matriz de correlaciónes
#esto daría igual si tipificamos los datos.
summary(acp)
```

```
## Importance of components:
##                           Comp.1    Comp.2    Comp.3    Comp.4    Comp.5
## Standard deviation     1.5488296 1.2567129 1.1840160 1.0228749 0.9676437
## Proportion of Variance 0.2665414 0.1754808 0.1557660 0.1162526 0.1040371
## Cumulative Proportion  0.2665414 0.4420222 0.5977882 0.7140408 0.8180779
##                           Comp.6     Comp.7     Comp.8       Comp.9
## Standard deviation     0.79952880 0.71169174 0.70110395 5.951540e-04
## Proportion of Variance 0.07102737 0.05627835 0.05461631 3.935647e-08
## Cumulative Proportion  0.88910531 0.94538366 0.99999996 1.000000e+00
```

Obtenemos las componentes principales (combinaciones lineales de las variables originales) ordenadas directamente en orden descendiente de varianza (cuando transformemos los datos, primero los tipificará y despues hará la transformación dada por la diagonalización pues hemos indicado cor = TRUE). El resumen también proporciona la proporción acumulada de la varianza total, de este modo podemos observar, por ejemplo, que solo nos bastan cinco componentes principales para explicar algo mas del 80% de la varianza total.

Podemos tratar de crear una tabla que reuna los información anterior, pero que sea algo mas facil de leer.

```
resumen<- matrix(NA,nrow=length(acp$sdev),ncol=3)
resumen[,1]<-  acp$sdev^2
resumen[,2]<- 100*resumen[,1]/sum(resumen[,1])
resumen[,3]<- cumsum(resumen[,2])
colnames(resumen)<- c("Autovalor","Porcentaje","Porcentaje acumulado")
resumen
```

```
##          Autovalor   Porcentaje Porcentaje acumulado
##  [1,] 2.398873e+00 2.665414e+01             26.65414
##  [2,] 1.579327e+00 1.754808e+01             44.20222
##  [3,] 1.401894e+00 1.557660e+01             59.77882
##  [4,] 1.046273e+00 1.162526e+01             71.40408
##  [5,] 9.363343e-01 1.040371e+01             81.80779
##  [6,] 6.392463e-01 7.102737e+00             88.91053
##  [7,] 5.065051e-01 5.627835e+00             94.53837
##  [8,] 4.915467e-01 5.461631e+00            100.00000
##  [9,] 3.542083e-07 3.935647e-06            100.00000
```

Aunque no es necesario, por curiosidad podemos diagonalizar nosotros mismos mediante una funcion de R la matriz de correlaciones.

```
R=cor(datos_empleo2)#matriz de correlaciones
eigen(R)$values#automalores de R
```

```
## [1] 2.398873e+00 1.579327e+00 1.401894e+00 1.046273e+00 9.363343e-01
## [6] 6.392463e-01 5.065051e-01 4.915467e-01 3.542083e-07
```

Vemos que obtenemos los mismos autovalores (varianza de las componentes principales). Veamos los coeficientes de las componentes principales (matriz de cambio de base).

```
loadings(acp)#Coeficientes
```

```
##
## Loadings:
##            Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6 Comp.7 Comp.8 Comp.9
## farmers     0.137  0.341  0.502 -0.410  0.348  0.334  0.203 -0.276 -0.309
## tradesmen          0.189         0.835  0.466                       -0.177
## managers   -0.489  0.140 -0.140 -0.114        -0.511 -0.182 -0.587 -0.262
## workers     0.170 -0.657  0.354               -0.295  0.108        -0.549
## unemployed        -0.180 -0.574 -0.309  0.588                0.302 -0.297
## middleempl -0.457 -0.144         0.106 -0.398  0.671               -0.364
## retired     0.332  0.549 -0.209        -0.378 -0.178         0.291 -0.531
## employrate -0.382  0.135  0.465         0.112        -0.576  0.514
## income     -0.492  0.157                      -0.184  0.756  0.347
##
##                Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6 Comp.7 Comp.8
## SS loadings     1.000  1.000  1.000  1.000  1.000  1.000  1.000  1.000
## Proportion Var  0.111  0.111  0.111  0.111  0.111  0.111  0.111  0.111
## Cumulative Var  0.111  0.222  0.333  0.444  0.556  0.667  0.778  0.889
##                Comp.9
```

```
## SS loadings     1.000
## Proportion Var  0.111
## Cumulative Var  1.000
```

```
round(eigen(R)$vectors,2)  #haciendo el calculo a "mano"
```

```
##        [,1]  [,2]  [,3]  [,4]  [,5]  [,6]  [,7]  [,8]  [,9]
##  [1,]  0.14  0.34 -0.50 -0.41  0.35  0.33 -0.20 -0.28 -0.31
##  [2,]  0.01  0.19 -0.02  0.84  0.47  0.09  0.04 -0.10 -0.18
##  [3,] -0.49  0.14  0.14 -0.11  0.02 -0.51  0.18 -0.59 -0.26
##  [4,]  0.17 -0.66 -0.35  0.09 -0.04 -0.29 -0.11  0.07 -0.55
##  [5,] -0.07 -0.18  0.57 -0.31  0.59  0.09  0.06  0.30 -0.30
##  [6,] -0.46 -0.14  0.10  0.11 -0.40  0.67  0.06 -0.07 -0.36
##  [7,]  0.33  0.55  0.21  0.03 -0.38 -0.18  0.05  0.29 -0.53
##  [8,] -0.38  0.13 -0.46 -0.06  0.11 -0.09  0.58  0.51  0.00
##  [9,] -0.49  0.16 -0.07  0.05  0.02 -0.18 -0.76  0.35  0.00
```

Aquí debemos tener cuidado al transformar de forma manual. Si tenemos un dato y queremos escribirlo en función de sus componentes principales no podemos multiplicar directamente por la matriz de autovectores. Antes de ese paso debemos tipificar, pensemos ademas, que en otro caso, esta discusión de la varianza no tendría sentido. Veamos que efectivamente esto es lo que hacemos si transformamos un dato.

```
m=matrix(loadings(acp),9,9)
datos_empleo3=scale(datos_empleo2)
t(as.matrix(datos_empleo3[1,])) %*% m
```

```
##           [,1]       [,2]       [,3]        [,4]       [,5]       [,6]
## [1,] 0.9727654 -0.6573566 -0.1036945 -0.03992037 -0.1093044 -0.2975457
##            [,7]      [,8]          [,9]
## [1,] -0.2744413 0.0427731 -0.0007691971
```

```
predict(acp, datos_empleo2[1,])
```

```
##          Comp.1     Comp.2     Comp.3      Comp.4     Comp.5     Comp.6
## ABZAC 0.9736674 -0.6579661 -0.1037907 -0.03995738 -0.1094058 -0.2978216
##           Comp.7     Comp.8        Comp.9
## ABZAC -0.2746957 0.04281276 -0.0007699103
```

Selección de componentes principales.

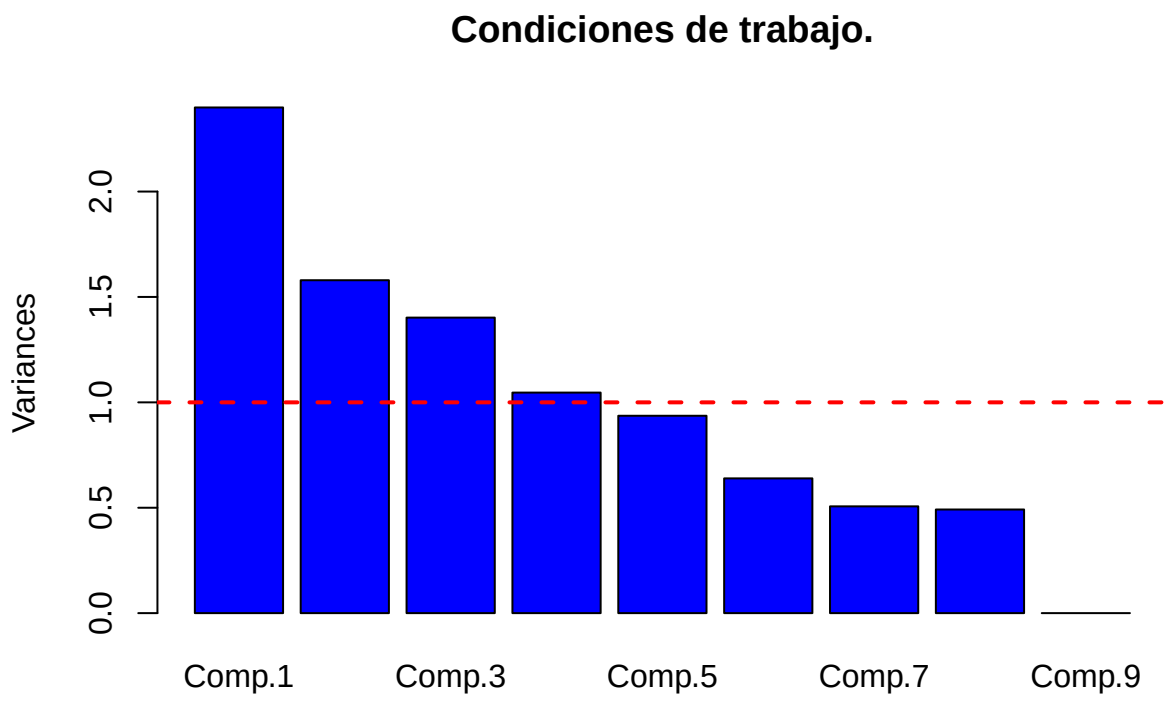
Podríamos tratar de usar procedimiento inferencial con el objetivo de seleccionar un numero de componentes principales. Los datos deben seguir una normal multivariante para aplicar este procedimiento. Veamos si las marginales siguen normales mediante el test de Shapiro-Wilk.

```
for (i in 1:length(colnames(datos_empleo2))){
  print(shapiro.test(datos_empleo2[,i])$p.value)
}
```

```
## [1] 3.040529e-27
## [1] 6.5186e-15
## [1] 1.846067e-14
## [1] 0.7882344
## [1] 3.516802e-06
## [1] 0.0437067
## [1] 1.256592e-06
## [1] 1.043093e-07
## [1] 2.923677e-24
```

Tenemos algunos p-valores muy pequeños, de modo que descartamos la normalidad en algunas marginales y por tanto en la distribución conjunta. Elijamos entonces el numero de componentes principales con el que vamos a trabajar mediante un procedimiento gráfico.

```
plot(acp,col="blue",main="Condiciones de trabajo.")
abline(h=mean(resumen[,1]),lwd=2,lty=2,col="red")
```



El grafico muestra las varianzas de cada una de las componentes principales, recordemos que al haber trabajado con la matriz de correlaciones, la media de estas varianzas es uno. De este modo, la linea roja separa las componentes principales cuya varianza supera a la media. Podriamos quedarnos con cuatro componentes principales.

Interpretación de las componentes principales.

Veamos las correlaciones de las componentes principales elejidas con las variables originales.

```
correlaciones<-loadings(acp)%*%diag(acp$sdev)
correlaciones[,1:4]
```

```
##                  [,1]       [,2]        [,3]        [,4]
## farmers     0.21263934  0.4282127  0.59400245 -0.41959794
## tradesmen   0.01528016  0.2369167  0.02824815  0.85421667
## managers   -0.75776637  0.1753706 -0.16602761 -0.11638916
## workers     0.26386775 -0.8255312  0.41856171  0.09015892
## unemployed -0.10338934 -0.2267579 -0.67974893 -0.31642883
## middleempl -0.70856531 -0.1807011 -0.11472756  0.10800076
## retired     0.51496776  0.6898408 -0.24715421  0.02786421
## employrate -0.59183564  0.1695443  0.55013007 -0.05835600
```

```
## income     -0.76247793  0.1971733  0.08092396  0.05362971
```

Mediante las correlaciones que acabamos de calcular, se definen las comunidades:para cada variable, es la suma de correlaciones cuadrado con las c.p. seleccionadas. Calculemoslas.

```
cbind(apply(correlaciones[,1:4]^2,1,sum))
```

```
##                 [,1]
## farmers    0.7574830
## tradesmen  0.7868471
## managers   0.6460763
## workers    0.9344505
## unemployed 0.6242943
## middleempl 0.5595442
## retired    0.8029337
## employrate 0.6850632
## income     0.6296747
```

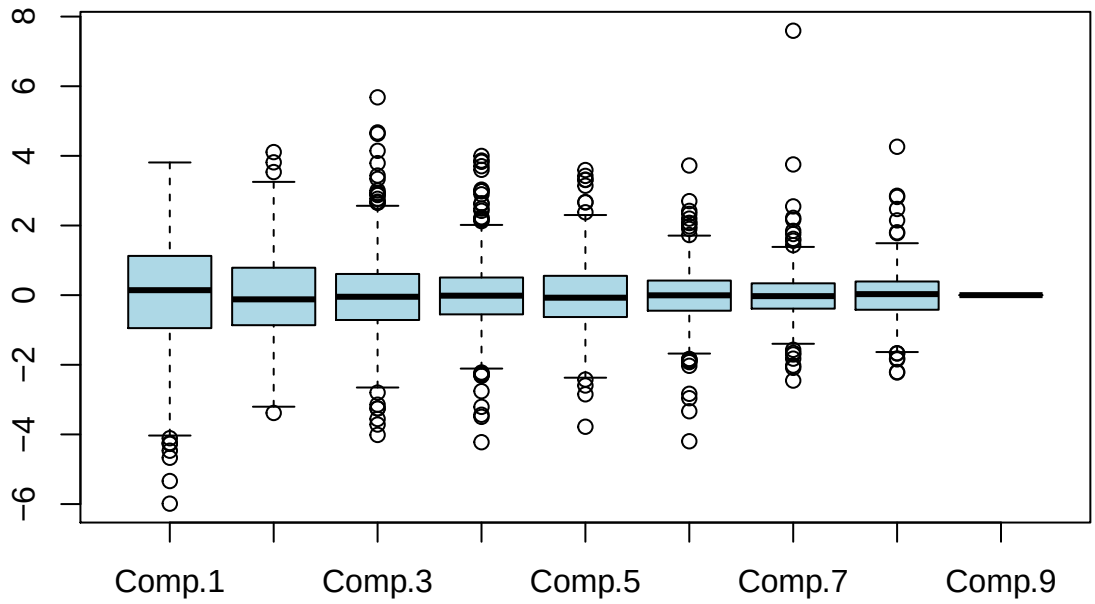
Presentemos las correlaciones en una grafica que a primera vista sea mas facil de entender.

```
corrplot(correlaciones[,1:4],method="ellipse",addCoef.col="black")
```

Viendo esto, podriamos pensar, por ejemplo, que si tenemos un elemento con un valor alto para la cuarta componente principal, es facil que tenga un valor alto para la variable tradesmen. Otra posible interpretación, algo mas arriesgada, es ver si las distintas variables se parecen en sus correlaciones a las componentes principales, y pensar por lo tanto, que se parecen entre si. Es decir, buscar "grupos" de variables parecidas.

Podemos usar graficos de caja y bigote para ver la variabilidad de cada componente principal.

```
boxplot(acp$scores,col="lightblue",notched=TRUE)
```



Vemos que hay componentes con bastantes valores perdidos.

Correlaciones reproducidas con cuatro componentes principales.

Podemos reproducir las correlaciones entre las variables originales usando las cuatro componentes principales que hemos elegido.

```
descompespec<-eigen(R)
autovalores<- descompespec$values
autovectores<- descompespec$vectors
Raprox2<- autovectores[,1:4]%*%diag(autovalores[1:4])%*%t(autovectores[,1:4])
Raprox2  #Matriz de correlaciones reproducidas
```

```
##              [,1]         [,2]       [,3]        [,4]        [,5]
##  [1,]  0.75748297 -0.236948154 -0.1358192 -0.08659811 -0.39008492
##  [2,] -0.23694815  0.786847106 -0.0741421 -0.10271137 -0.34480299
##  [3,] -0.13581917 -0.074142097  0.6460763 -0.42471034  0.18826427
##  [4,] -0.08659811 -0.102711373 -0.4247103  0.93445047 -0.15313113
##  [5,] -0.39008492 -0.344802988  0.1882643 -0.15313113  0.62429433
##  [6,] -0.34151270  0.035377109  0.5117151 -0.07607650  0.15804488
##  [7,]  0.24639904  0.188124054 -0.2314561 -0.53453879 -0.05048328
##  [8,]  0.29801815 -0.003183978  0.3936615 -0.07112836 -0.33274082
##  [9,] -0.05213459  0.083160213  0.5926810 -0.32525916 -0.03785648
```

```
##              [,6]        [,7]         [,8]        [,9]
##  [1,] -0.34151270  0.24639904  0.298018148 -0.05213459
##  [2,]  0.03537711  0.18812405 -0.003183978  0.08316021
##  [3,]  0.51171513 -0.23145612  0.393661452  0.59268100
##  [4,] -0.07607650 -0.53453879 -0.071128362 -0.32525916
##  [5,]  0.15804488 -0.05048328 -0.332740821 -0.03785648
##  [6,]  0.55954425 -0.45817851  0.319299797  0.50114383
##  [7,] -0.45817851  0.80293374 -0.325410720 -0.27513973
##  [8,]  0.31929980 -0.32541072  0.685063207  0.52608030
##  [9,]  0.50114383 -0.27513973  0.526080300  0.62967473
```

Veamos si se parece mucho a la matriz de correlaciones original mediante las correlaciones residuales.

```
Resid2=R-Raprox2#CORRELACIONES RESIDUALES
mean((Resid2)^2)
```

```
## [1] 0.02201881
```

```
corrplot(Resid2)
```

Parece que la aproximación es buena, los resuduos mas altos se encuentran en la diagonal, esto se debe a que quedaba relativamente bastante varivilidad por explicar.

ii) Realizar e interpretar un análisis de componentes principales para datos mixtos sobre la unión de gironde$employment y gironde$services.

Lectura y preprocesamiento de los datos.

```
datos_servicios=gironde$services
head(datos_servicios)
```

```
##                     butcher  baker postoffice dentist grocery nursery
## ABZAC                     0 2 or +     1 or +       0       0       0
## AILLAS                    0      0          0       0  1 or +       0
## AMBARES-ET-LAGRAVE        1 2 or +     1 or +  3 or +  1 or +  1 or +
## AMBES                     0      1     1 or +  1 to 2  1 or +       0
## ANDERNOS-LES-BAINS  2 or + 2 or +     1 or +  3 or +  1 or +       0
## ANGLADE                   0      1          0       0  1 or +       0
##                     doctor chemist restaurant
## ABZAC                    0       1          1
## AILLAS              3 or +       0          1
## AMBARES-ET-LAGRAVE  3 or +  2 or +     3 or +
## AMBES               3 or +       1     3 or +
## ANDERNOS-LES-BAINS  3 or +  2 or +     3 or +
## ANGLADE                  0       0          2
```

Vemos que tenemos variables de tipo factor, seria combeniente ver las etiquetas posible de cada variable.

```
for (i in c(1:9)){
  print(levels(datos_servicios[,i]))
}
```

```
## [1] "0"      "1"      "2 or +"
## [1] "0"      "1"      "2 or +"
## [1] "0"      "1 or +"
## [1] "0"      "1 to 2" "3 or +"
## [1] "0"      "1 or +"
## [1] "0"      "1 or +"
## [1] "0"      "1 to 2" "3 or +"
## [1] "0"      "1"      "2 or +"
## [1] "0"      "1"      "2"      "3 or +"
```

Construyamos nuestros datos mixtos uniendo las variables cuantitativas y las cualitativas. Una vez echo esto, podemos eliminar los datos con valores perdidos.

```
datos_mixtos=cbind(datos_empleo,datos_servicios)#unimos las variables
which(apply(apply(datos_mixtos,2,is.na),1,sum)!=0)#buscamos valores perdidos
```

```
##              BOSSUGAN SAINT-AVIT-DE-SOULEGE
##                    63                   369
```

```
datos_mixtos2=datos_mixtos[-which(apply(apply(datos_mixtos,2,is.na),1,sum)!=0),]
dim(datos_mixtos)
```

```
## [1] 542  18
```

```
dim(datos_mixtos2)
```

```
## [1] 540  18
```

Análisis de componentes principales.

Para este tipo de datos mixtos habrá que volver a separar las variables en cualitativas y cuantitativas, antes los juntamos para buscar los valores perdidos. Esta separación es necesaria, pues se construye una matriz (que definirá una métrica) apartir de dos matrizes, cada una obtenida de los datos con variables cualitativas y cuantitativas respectivamente. Cuando finalmente tengamos la matriz, lo que se hará será descomponer en valores singulares (tal como hacíamos con las componentes principales para datos cuantitativos).

Todo este proceso lo realiza la función PCAmix, solo tenemos que introducirle las dos tablas de datos cuantitativos y cualitativos.

```
split<-splitmix(datos_mixtos2)#dividimos en datos cualitativos y cuantitativos.

X1<-split$X.quanti#guardamos datos cuantitativos
X2<-split$X.quali#guardamos datos cualitativos

res.pcamix<-PCAmix(X.quanti=X1, X.quali=X2, #guardamos la información del analisis
                   rename.level=TRUE, graph=FALSE,ndim=25)
#Con ndim=25 se guarda información en el objeto de 25 dimensiones
```

Hemos generado el objeto y guardado la información de las 25 dimensiones que genera, podemos comenzar viendo los autovalores y la proporción de inercia explicada por cada dimension.

```
res.pcamix$eig
```

```
##          Eigenvalue   Proportion Cumulative
## dim 1  6.310191e+00 2.524076e+01   25.24076
## dim 2  2.697311e+00 1.078924e+01   36.03001
## dim 3  2.337837e+00 9.351347e+00   45.38135
## dim 4  1.560155e+00 6.240620e+00   51.62197
## dim 5  1.179731e+00 4.718922e+00   56.34090
## dim 6  1.050822e+00 4.203289e+00   60.54418
## dim 7  1.024263e+00 4.097051e+00   64.64124
## dim 8  9.791605e-01 3.916642e+00   68.55788
## dim 9  9.391575e-01 3.756630e+00   72.31451
## dim 10 8.660783e-01 3.464313e+00   75.77882
## dim 11 7.734111e-01 3.093644e+00   78.87247
## dim 12 7.289646e-01 2.915859e+00   81.78832
## dim 13 6.871324e-01 2.748530e+00   84.53685
## dim 14 6.030891e-01 2.412356e+00   86.94921
## dim 15 5.577486e-01 2.230994e+00   89.18020
## dim 16 4.835794e-01 1.934318e+00   91.11452
## dim 17 4.750533e-01 1.900213e+00   93.01474
## dim 18 3.966466e-01 1.586586e+00   94.60132
## dim 19 3.446142e-01 1.378457e+00   95.97978
## dim 20 3.109675e-01 1.243870e+00   97.22365
## dim 21 2.622891e-01 1.049157e+00   98.27280
## dim 22 2.163303e-01 8.653213e-01   99.13813
## dim 23 1.320954e-01 5.283815e-01   99.66651
## dim 24 8.337272e-02 3.334909e-01  100.00000
## dim 25 3.459173e-07 1.383669e-06  100.00000
```

```
sum(res.pcamix$eig[,1])#inercia total
```

```
## [1] 25
```

Vemos que con 12 dimensiones tenemos explicada el ochenta por ciento de la inercia, que en total suma 25.

Pasemos a ver la proporción que explica cada dimensíon sobre cada variable.

```
#Suma de las proporciones, a veces vale mas de uno debido a las variables dummie.
apply(res.pcamix$sqload,1,sum)
```

```
##    farmers  tradesmen   managers    workers unemployed middleempl
##          1          1          1          1          1          1
##    retired employrate     income    butcher      baker postoffice
##          1          1          1          2          2          1
##    dentist    grocery    nursery     doctor    chemist restaurant
##          2          1          1          2          2          3
```

Tenemos venticinco dimensiones, lo cual puede ser algo engorroso. Podemos volver a generar el objeto con la información de las componentes principales, pero en esta ocasión guardar solo la inofrmación de cinco dimensiones.

```
res.pcamix<-PCAmix(X.quanti=X1, X.quali=X2,
                   rename.level=TRUE, graph=FALSE,ndim=5)#con ndim=5 guardamos 5 dimensiones
```

Ahora si, veamos la proporción que explica cada dimensíon sobre cada variable.

```
res.pcamix$sqload
```

```
##                   dim1         dim2         dim3         dim4         dim5
## farmers    0.221332548 0.0213480663 7.406745e-03 0.0921912423 0.0676294211
## tradesmen  0.008552195 0.0022176098 9.356754e-04 0.0733190612 0.0520025702
## managers   0.107488536 0.0970735852 3.659378e-01 0.0223013485 0.0150251498
## workers    0.028092562 0.1210757157 8.806367e-03 0.5764082077 0.1574710058
## unemployed 0.086720359 0.0047544027 2.972809e-03 0.0641193770 0.2451916212
## middleempl 0.070369927 0.0129569414 4.105238e-01 0.0212297982 0.0068941894
## retired    0.000903663 0.0004093083 3.243865e-01 0.4709492067 0.0083627628
## employrate 0.056126955 0.0437326890 4.763605e-01 0.0168493356 0.0603922792
## income     0.048900282 0.0777582070 4.622209e-01 0.0304464724 0.0000427087
## butcher    0.622010935 0.1334729361 3.198844e-02 0.0010302467 0.0125142376
## baker      0.764622022 0.3515319079 1.086986e-02 0.0030645086 0.0793984524
## postoffice 0.667786490 0.0793274627 7.386514e-05 0.0005191622 0.0059682165
## dentist    0.806968351 0.3880883028 4.466690e-02 0.0372969052 0.0552199888
## grocery    0.188365580 0.0123467307 4.267250e-02 0.0116091051 0.0605424340
## nursery    0.231805367 0.1482730405 5.259283e-03 0.0372243936 0.0260171376
## doctor     0.843660628 0.4249093927 2.182137e-02 0.0143299204 0.0366668101
## chemist    0.873770912 0.5207297706 7.353566e-02 0.0639193474 0.0141600889
## restaurant 0.682713498 0.2573046964 4.739779e-02 0.0233474477 0.2762314840
```

Podemos ver también como se reparte la inercia de cada dimensión entre las variables (lo expresaremos en porcentajes).

```
#unimos los porcentajes de las variables cualitativas y cuantitativas
A=rbind(100*res.pcamix$quali$contrib.pct,
        res.pcamix$quanti$contrib.pct)
A
```

```
##                   dim1        dim2         dim3        dim4         dim5
## butcher     9.85724448  4.94837072  1.368292104  0.06603489  1.060770830
## baker      12.11725676 13.03268101  0.464953923  0.19642333  6.730219189
## postoffice 10.58266715  2.94098343  0.003159551  0.03327632  0.505896573
## dentist    12.78833518 14.38797145  1.910608270  2.39058960  4.680728871
## grocery     2.98510117  0.45774224  1.825298444  0.74409943  5.131886561
## nursery     3.67350805  5.49706924  0.224963648  2.38594188  2.205345741
```

```
## doctor     13.36981169 15.75307518  0.933400179  0.91849333  3.108066486
## chemist    13.84698083 19.30551634  3.145457683  4.09698677  1.200281610
## restaurant 10.81922114  9.53930484  2.027420750  1.49648249 23.414794345
## farmers     3.50754128  0.79145742  0.316820441  5.90910757  5.732615862
## tradesmen   0.13552989  0.08221558  0.040023129  4.69947262  4.408003999
## managers    1.70341181  3.59890252 15.652837935  1.42943151  1.273608593
## workers     0.44519354  4.48875663  0.376688707 36.94557116 13.348048392
## unemployed  1.37429059  0.17626455  0.127160677  4.10980790 20.783696706
## middleempl  1.11517906  0.48036517 17.559986441  1.36074922  0.584386781
## retired     0.01432069  0.01517468 13.875496820 30.18605078  0.708870578
## employrate  0.88946526  1.62134410 20.376124893  1.07997825  5.119158677
## income      0.77494142  2.88280490 19.771306405  1.95150294  0.003620208
```

```
apply(A,2,sum) #vemos que suman 100 para cada dimensión
```

```
## dim1 dim2 dim3 dim4 dim5
##  100  100  100  100  100
```

Ya hemos visto por encima como se reparte la inercia, veamos como serían algunos de los coeficientes de la transformación lineal para expresar nuestros datos en terminos de las dimensiones (componentes principales). A su vez, veamos algunos resultados de estas transformaciones.

```
head(data.frame(res.pcamix$coef))
```

```
##                    dim1        dim2          dim3        dim4        dim5
## const       1.844672083 -3.89708621 -13.902582428 -2.62436882  5.31565501
## farmers    -0.045087603  0.02141754  -0.013550719  0.05852167 -0.05764109
## tradesmen  -0.015529780 -0.01209553   0.008439239  0.09144760  0.08856635
## managers    0.037119613  0.05395462   0.112522705  0.03400363  0.03209679
## workers    -0.009047638 -0.02872922  -0.008322467 -0.08242178 -0.04954156
## unemployed  0.029352087  0.01051192  -0.008928451 -0.05075870  0.11414610
```

```
head(res.pcamix$ind$coord)
```

```
##                          dim 1      dim 2      dim 3      dim 4       dim 5
## ABZAC                0.3089595 -1.3275558 -0.3797857 -0.3256275  0.08540901
## AILLAS              -0.5151541  0.4860533 -0.6130975  1.1286307  1.83588594
## AMBARES-ET-LAGRAVE   5.4067580  2.1560126 -0.5016042 -2.2518391  0.77006102
## AMBES                2.4031163 -2.7811727 -0.8398837 -0.6566105 -0.55942986
## ANDERNOS-LES-BAINS   5.0613694  2.5346005 -1.1770582  2.2782578  0.24678929
## ANGLADE             -1.1175075 -1.6720510 -1.8199727 -0.9576146  0.43372770
```

Del mismo modo, podemos ver las coordenadas de las categorías de las variables cualitativas, al igual que sucedía con el analisis de correspondencias. Veamos pues algunas de estas coordenadas.

```
head(res.pcamix$levels$coord[,1:4])
```

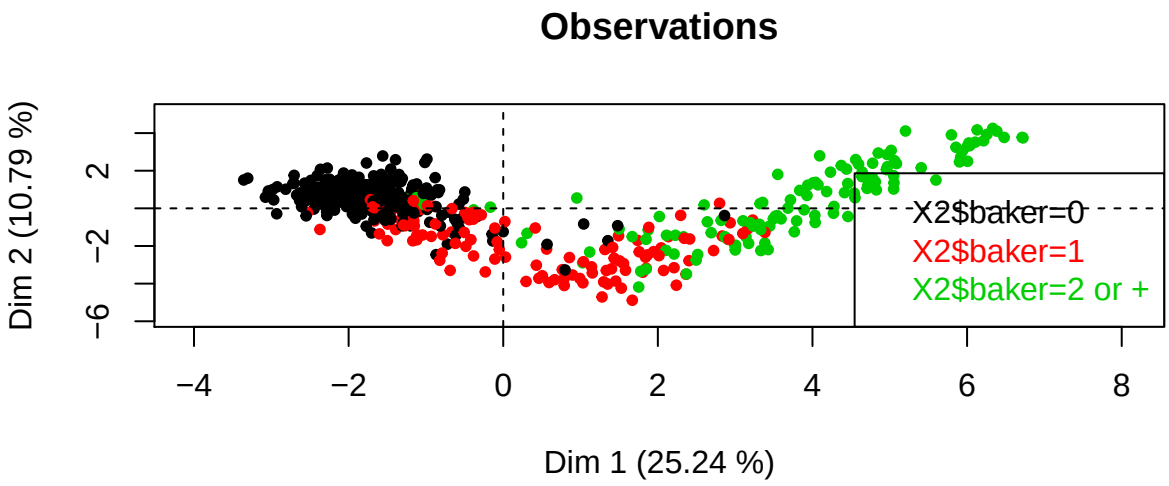
```
##                      dim1        dim2         dim3         dim4
## butcher=0      -0.4819660  0.05013436  0.066880538 -0.016613015
## butcher=1       0.4877141 -0.67694879  0.093225025  0.068987487
## butcher=2 or +  1.7304291  0.60277049 -0.441253891 -0.005573798
## baker=0        -0.6840912  0.33685588 -0.002428494 -0.016954762
## baker=1         0.1044372 -1.06360512  0.152655538 -0.055652622
## baker=2 or +    1.4986535  0.31536672 -0.153155074  0.097751722
```

Interpretación.

Podemos tratar de ayudarnos de este analisis de componentes principales para interpretar los datos, aunque con las dos primeras dimensiones solo explicamos el 36% de la inercia. Veamos de todas formas algunas graficas para ver que tenemos.

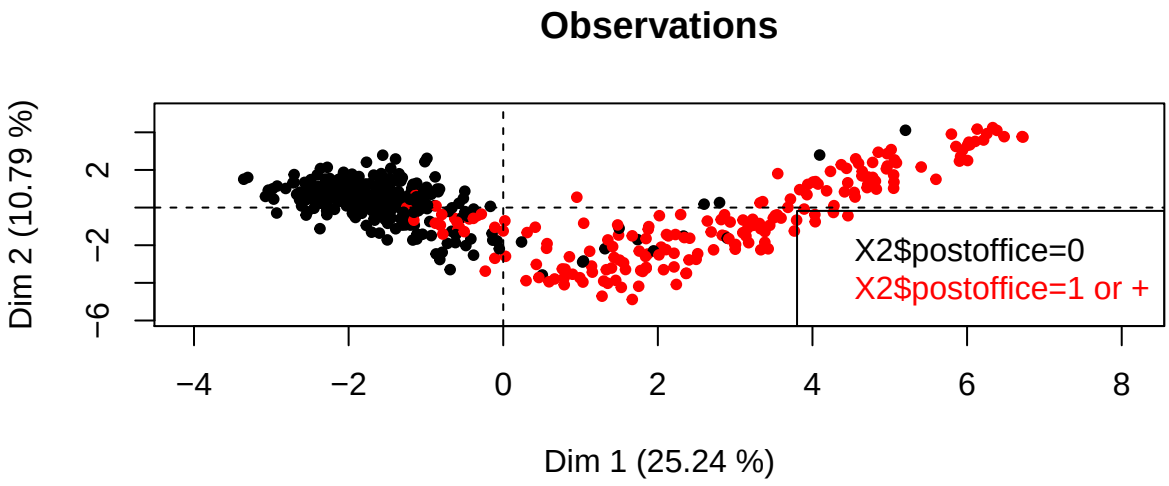
Comencemos viendo como son algunas nuves de puntos en las dos primeras dimensiones.

```
plot(res.pcamix,choice="ind",axes=c(1,2),
     coloring.ind=X2$baker ,label=FALSE,
     posleg="bottomright", main="Observations")
```



Hemos pintado los puntos de distintos colores en función de la variable baker. Vemos que estas dos dimensiones separan bastante bien estos casos. Parece que las categorías de esta variable realmente vienen de una de conteo, y que dicha variable de conteo esta directamente relaccionada con la dimensión 1.

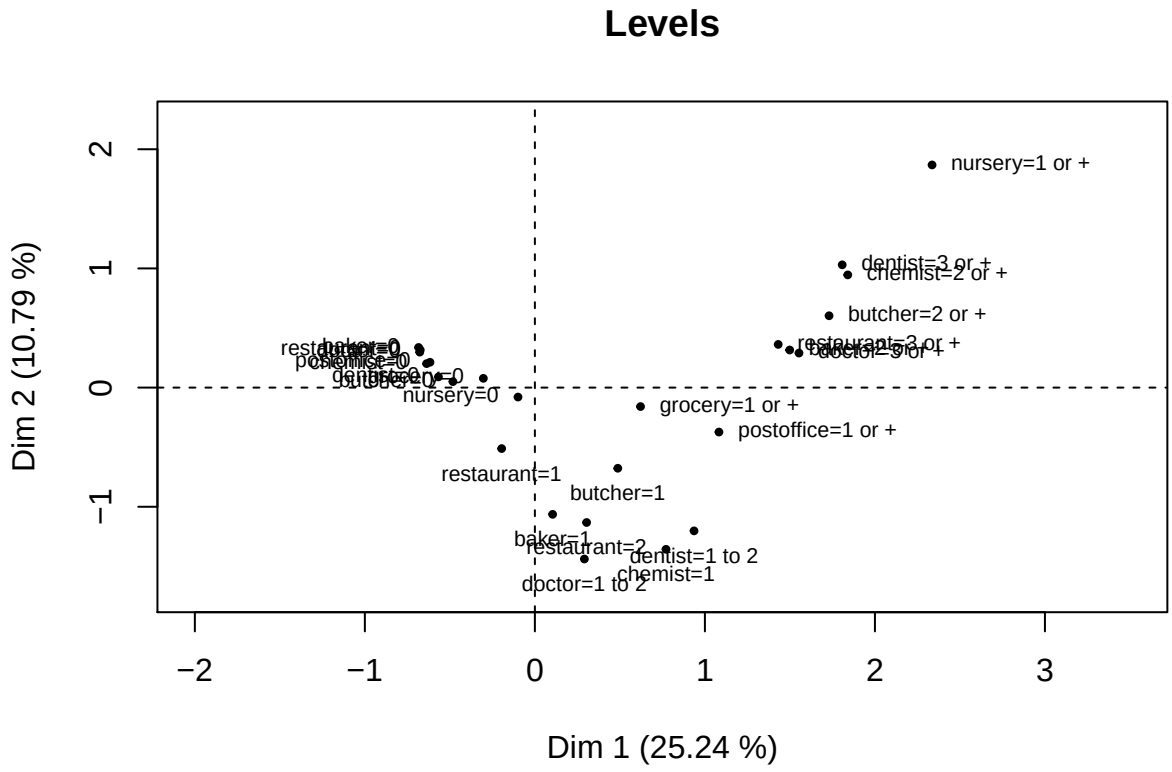
```
plot(res.pcamix,choice="ind",axes=c(1,2),
     coloring.ind=X2$postoffice ,label=FALSE,
     posleg="bottomright", main="Observations")
```



Es la misma nuve de puntos pero identificando los puntos en función de la variable post ofice. Parece que si la primera dimension es positiva o negativa determina con cierta fuerza a que categoría pertenecen los datos.

Veamos ahora si podemos sacar conclusiones de como quedarían las distintas etiquetas de las variables cualitativas distribuidas en estas dos primeras dimensiones.

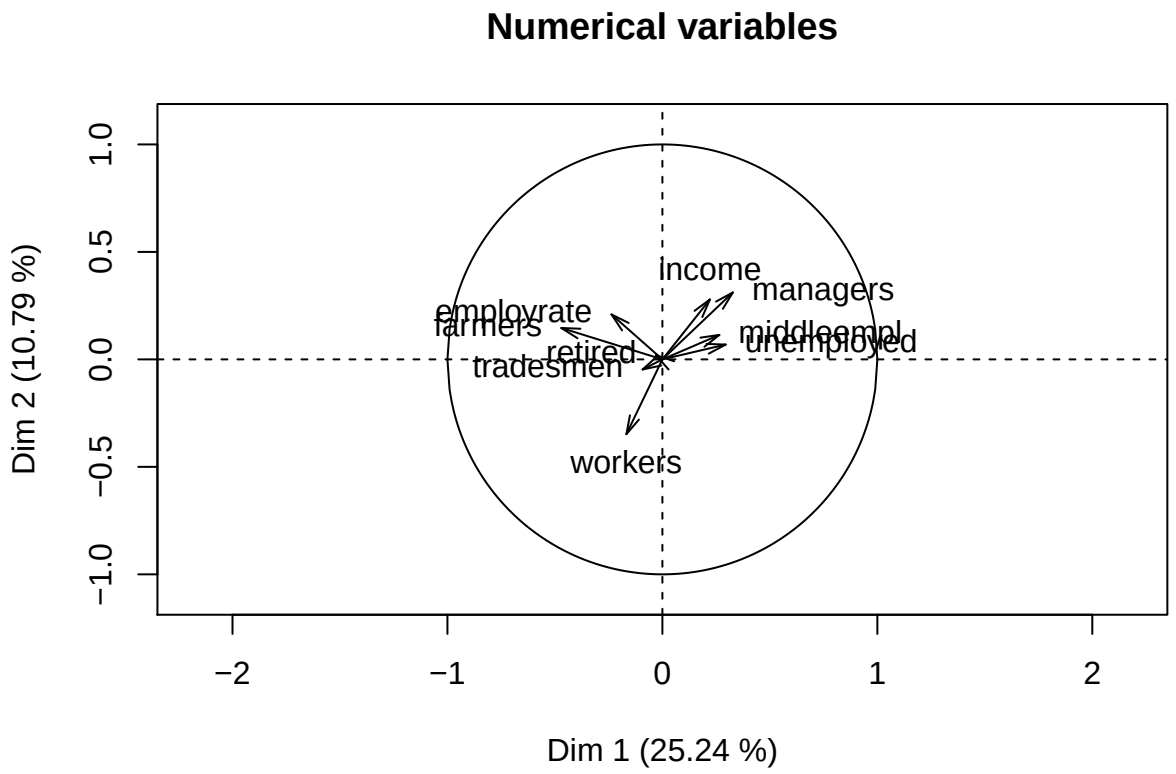
```
plot(res.pcamix,choice="levels",
     axes=c(1,2),xlim=c(-2,3.5),cex=0.7,
     main="Levels")
```



Se aprecia rapidamente como las etiquetas correspondientes a los valores nulos se agrupan a la izquierda de la grafica. Además, el resto de etiquetas parece que tambien se agrupan en valores medios y valores altos, siendo ambas dimensiones mayores para etiquetas correspondientes a valores mas altos (si dejamos de lado a las correspondientes a valores nulos).

Veamos ahora una representación de las 'correlaciones' entre las variables cuantitativas.

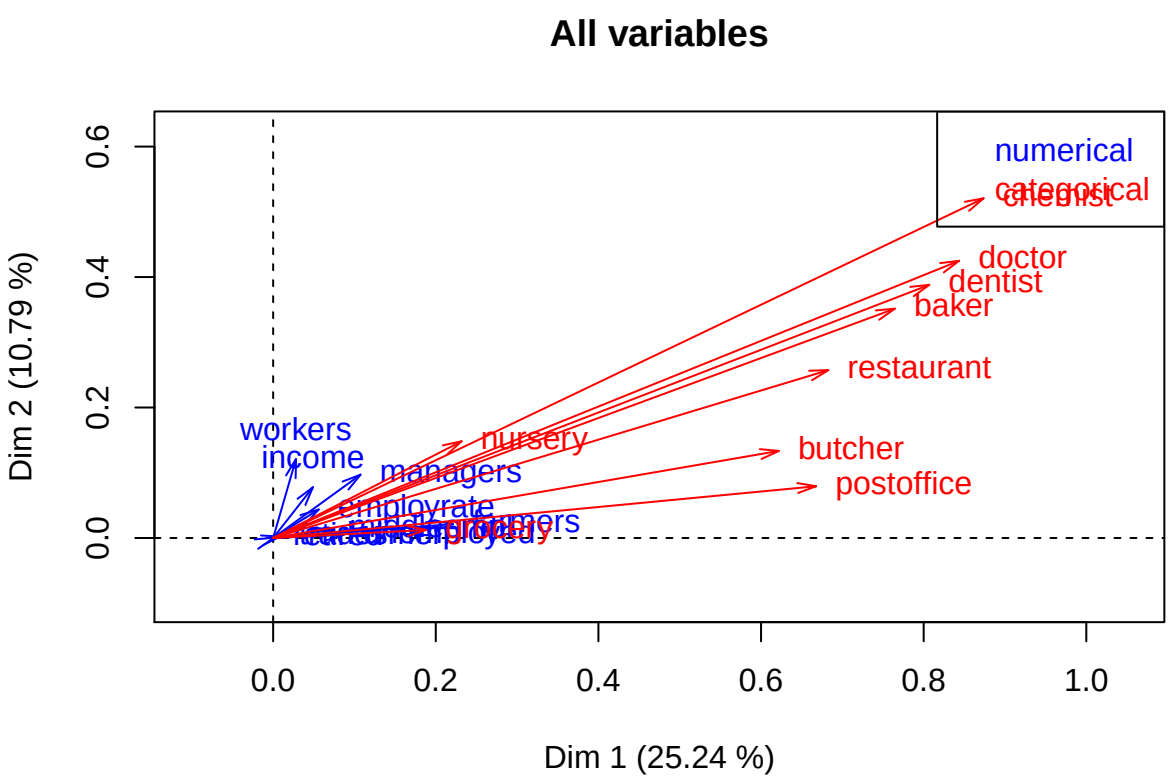
```
plot(res.pcamix,choice="cor",axes=c(1,2),
     main="Numerical variables")
```



Se aprecia como algunas variables se parecen, como income y managers, las cuales a su vez parecen tener una relacción inversa con la variable workers. De hecho, siendo menos exigentes, podemos apreciar tres grupos de variables que se agrupan en los tres primeros cuadrantes respectivamente.

Tratemos por ultimo de reprensentar conjuntamente las variables cualitativas y cuantitativas.

```
plot(res.pcamix,choice="sqload",axes=c(1,2),
     coloring.var="type", leg=TRUE,
     xlim=c(-0.1,1.05),posleg="topright",
     main="All variables")
```



Aquí se podría apreciar como se relaccionan algunas variables con las dimensiones del análisis de componentes principales y deducir de esto relacciones entre ellas. Lo primero que salta a la vista, es que las variables categoricas toman valores mas altos y, por lo tanto, están mas relaccionadas con estas dos dimensiones (mirar tabla de porcentajes de inercia repartida entre las variables para cada dimensión). Por otro lado podemos tratar de ver algunas de estas relacciones, vemos por ejemplo que la variable workers se relacciona bastante con la segunda dimensión, mientras que farmers lo hace con la primera (en la grafica anterior, al igual que en esta, se observa como son perpendiculares, de lo que se deduciría que son mas o menos independientes), esto podríamos hacerlo con mas variables. Por último, se puede apreciar como en general, las variables categoricas parecen estar mas relaccionadas entre sí que las numericas, aunque evidentemente se aprecian grupos como el formado por doctor dentist y baker.

iii) Aplicar procedimientos de selección de variables para construir modelos de regresión lineal donde income es la variable dependiente, sobre gironde$employment.

Preprocesamiento de los datos.

Ya tenemos los datos cargados y sin valores perdidos, dividamoslos en conjuntos de entrenamiento y test para evaluar que tal funciona la selección de variables.

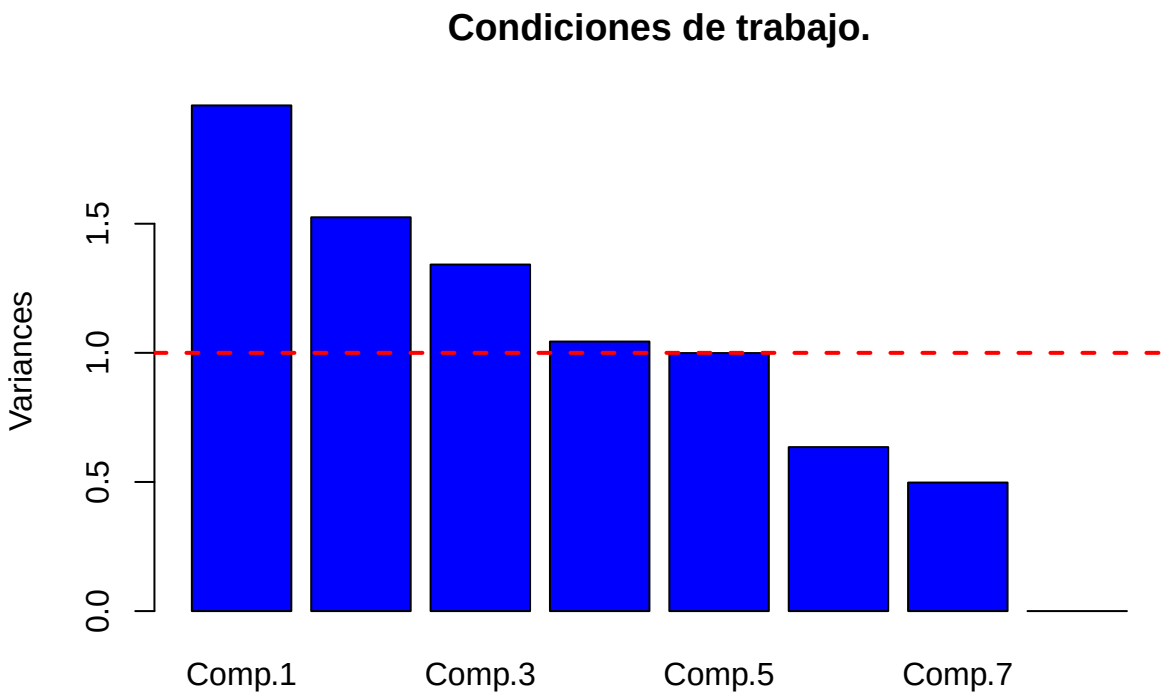
```
set.seed(1)
n=nrow(datos_empleo2)#numero de filas

inditest<- sample(1:n,trunc(n*0.25)+1)#indices de test
inditrain<-setdiff(1:n,inditest)#indices de entrenamiento

#tomamos train y test tipificados
train_empleo=data.frame(scale(datos_empleo2[inditrain,]))
#escalamos respecto a los datos train
test_empleo=data.frame(scale(datos_empleo2[inditest,], center =
                               apply(datos_empleo2[inditrain,],2,mean),
            scale = apply(datos_empleo2[inditrain,],2,sd)))
```

Ya que hemos estado calculando componentes principales en las partes anteriores del ejercicio, podríamos probar a usarlas para ver si obtenemos buenos resultados con el modelo de regresión. Las calcularemos usando solo los datos de entrenamiento y las variables predictoras.

```
#calculamos componentes principales
acp_train<- princomp(train_empleo[,-c(9)], cor = TRUE)
#transformacion de datos de entrenamiento
train_empleo_acp=data.frame(cbind(predict(acp_train,train_empleo[,-c(9)]),
                                  train_empleo[,c('income')]))
colnames(train_empleo_acp)[9]='income'
#transformacion de datos test
test_empleo_acp=data.frame(cbind(predict(acp_train,test_empleo[,-c(9)]),
                                 test_empleo[,c('income')]))
colnames(test_empleo_acp)[9]='income'
```

Hagamos simplemente una grafica de la varianza explicada por estas componentes principales para despues seguir con la selección de variables.

```
plot(acp_train,col="blue",main="Condiciones de trabajo.")
abline(h=1,lwd=2,lty=2,col="red")
```



En esta ocasión tenemos evidentemente una componente principal menos.

Modelos de regresión lineal con todas las variables.

Ya que vamos a crear distintos modelos, podemos crear una función que evalue que tal se comportan sobre el conjunto test.

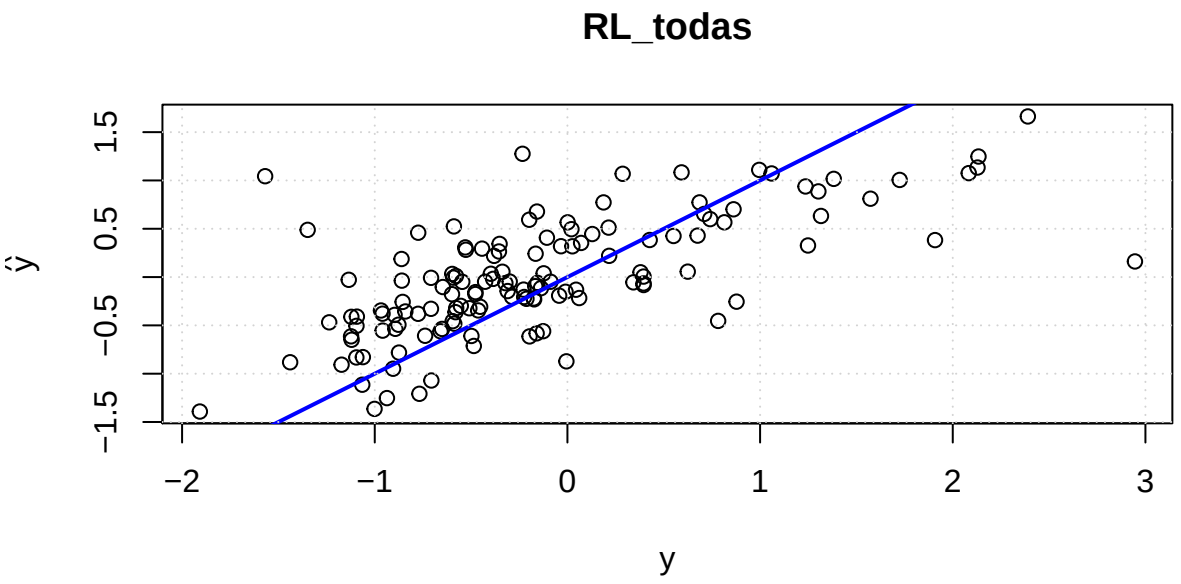
```
Ajuste<- function(y,pred,titulo)
{
  residuos=y-pred
  plot(y,pred,main=titulo,ylab=expression(hat(y)))
  abline(a=0,b=1,col="blue",lwd=2)
  grid()
  MSE= mean(residuos^2)
  RMSE= sqrt(MSE)
  R2= cor(y,pred)^2
  return(list(MSE=MSE,RMSE=RMSE,R2=R2))
}
```

Comencemos definiendo un modelo con todas las variables.

```
modeloRL=lm(income~.,data=train_empleo)
summary(modeloRL)
```

```
##
## Call:
## lm(formula = income ~ ., data = train_empleo)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.2587 -0.3805 -0.0704  0.2797  9.9593
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.985e-14  4.122e-02   0.000    1.000
## farmers      2.006e+01  2.023e+01   0.992    0.322
## tradesmen    1.195e+01  1.197e+01   0.998    0.319
## managers     1.768e+01  1.752e+01   1.009    0.314
## workers      3.731e+01  3.771e+01   0.990    0.323
## unemployed   1.900e+01  1.913e+01   0.993    0.321
## middleempl   2.438e+01  2.443e+01   0.998    0.319
## retired      3.464e+01  3.500e+01   0.990    0.323
## employrate   2.319e-01  4.845e-02   4.787  2.4e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8285 on 395 degrees of freedom
## Multiple R-squared:  0.3272, Adjusted R-squared:  0.3136
## F-statistic: 24.01 on 8 and 395 DF,  p-value: < 2.2e-16
```

```
predRL_test=predict(modeloRL,newdata=test_empleo)
Ajuste(test_empleo$income ,predRL_test,"RL_todas")
```



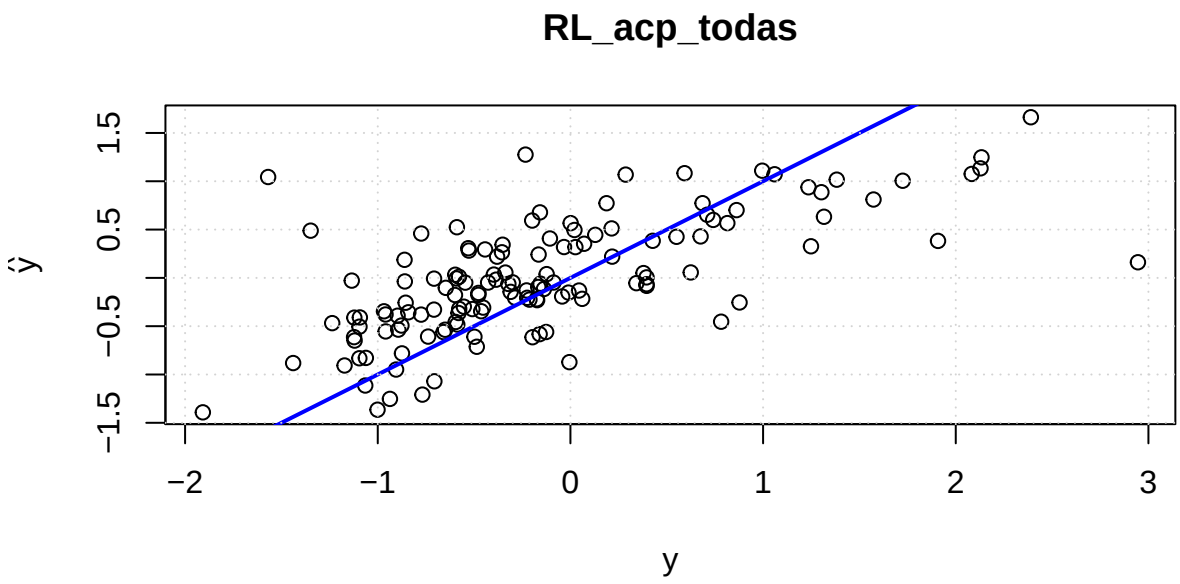
```
## $MSE
## [1] 0.418813
##
## $RMSE
## [1] 0.6471577
##
## $R2
## [1] 0.4584485
```

Hacemos lo mismo con los datos transformados mediante el uso de componentes principales.

```
modeloRL_acp=lm(income~.,data=train_empleo_acp)
summary(modeloRL_acp)
```

```
##
## Call:
## lm(formula = income ~ ., data = train_empleo_acp)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.2587 -0.3805 -0.0704  0.2797  9.9593
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.147e-15  4.122e-02   0.000  1.00000
## Comp.1      -3.827e-01  2.946e-02 -12.993  < 2e-16 ***
## Comp.2      -9.211e-02  3.338e-02  -2.759  0.00606 **
## Comp.3       9.823e-02  3.558e-02   2.760  0.00604 **
## Comp.4       5.878e-02  4.035e-02   1.457  0.14593
## Comp.5       5.403e-02  4.124e-02   1.310  0.19088
## Comp.6       9.273e-02  5.173e-02   1.793  0.07381 .
## Comp.7      -4.862e-03  5.843e-02  -0.083  0.93373
## Comp.8      -6.629e+01  6.677e+01  -0.993  0.32142
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8285 on 395 degrees of freedom
## Multiple R-squared:  0.3272, Adjusted R-squared:  0.3136
## F-statistic: 24.01 on 8 and 395 DF,  p-value: < 2.2e-16
```

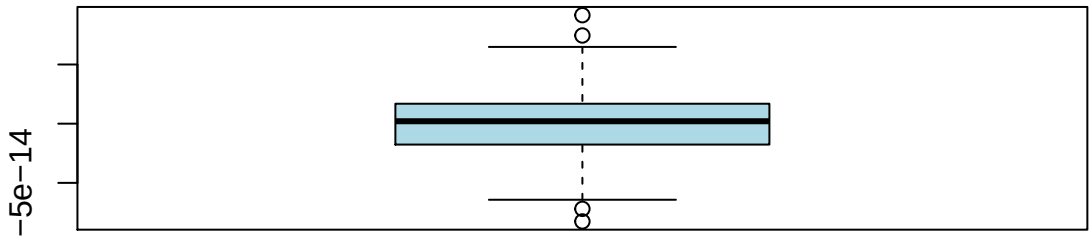
```
predRL_test_acp=predict(modeloRL_acp,newdata=test_empleo_acp)
Ajuste(test_empleo_acp$income ,predRL_test_acp,"RL_acp_todas")
```




```
## $MSE
## [1] 0.418813
##
## $RMSE
## [1] 0.6471577
##
## $R2
## [1] 0.4584485
```

Se observa que, aunque sea totalmente razonable, resulta curioso. Hemos obtenido los mismos resultados usando las componentes principales y las variables originales (ambos parecen bastante malos).

```
boxplot(predRL_test_acp-predRL_test,col='lightblue')
```



Esto se debe a que al fin y al cabo las componentes principales son combinaciones lineales, las cuales las

puede hacer el modelo lineal internamente. Pero puede merecer la pena usar estas componentes principales mas adelante, cuando seleccionemos variables. De todas formas hay un procedimiento que usa algo similar, pero pensado para aplicarlo en los modelos de regresión, regresión PLS (esto se hace por que al crear las componentes principales, no estamos teniendo en cuenta a la variable respuesta).

Exploración completa.

Tomemos todos los subconjuntos posibles de variables, y veamos que resultados se obtienen con ellos.

```
library(leaps)
#exploración completa con variables originales
modeloRL_mejorsub=regsubsets(income~.,data=train_empleo,nvmax=8)
#exploración completa con componentes principales
modeloRL_mejorsub_acp=regsubsets(income~.,data=train_empleo_acp,nvmax=8)
summary(modeloRL_mejorsub)
```

```
## Subset selection object
## Call: regsubsets.formula(income ~ ., data = train_empleo, nvmax = 8)
## 8 Variables  (and intercept)
##            Forced in Forced out
## farmers        FALSE      FALSE
## tradesmen      FALSE      FALSE
## managers       FALSE      FALSE
## workers        FALSE      FALSE
## unemployed     FALSE      FALSE
## middleempl     FALSE      FALSE
## retired        FALSE      FALSE
## employrate     FALSE      FALSE
## 1 subsets of each size up to 8
## Selection Algorithm: exhaustive
##          farmers tradesmen managers workers unemployed middleempl retired
## 1  ( 1 ) " "     " "       "*"      " "     " "         " "         " "
## 2  ( 1 ) " "     " "       "*"      " "     " "         " "         " "
## 3  ( 1 ) " "     " "       "*"      " "     " "         "*"         " "
## 4  ( 1 ) " "     "*"       "*"      " "     " "         "*"         " "
## 5  ( 1 ) " "     "*"       "*"      " "     "*"         "*"         " "
## 6  ( 1 ) "*"     "*"       "*"      " "     "*"         "*"         " "
## 7  ( 1 ) "*"     "*"       "*"      " "     "*"         "*"         "*"
## 8  ( 1 ) "*"     "*"       "*"      "*"     "*"         "*"         "*"
##          employrate
## 1  ( 1 ) " "
## 2  ( 1 ) "*"
## 3  ( 1 ) "*"
## 4  ( 1 ) "*"
## 5  ( 1 ) "*"
## 6  ( 1 ) "*"
## 7  ( 1 ) "*"
## 8  ( 1 ) "*"
```

```
summary(modeloRL_mejorsub_acp)
```

```
## Subset selection object
## Call: regsubsets.formula(income ~ ., data = train_empleo_acp, nvmax = 8)
## 8 Variables  (and intercept)
##        Forced in Forced out
```

```
## Comp.1     FALSE      FALSE
## Comp.2     FALSE      FALSE
## Comp.3     FALSE      FALSE
## Comp.4     FALSE      FALSE
## Comp.5     FALSE      FALSE
## Comp.6     FALSE      FALSE
## Comp.7     FALSE      FALSE
## Comp.8     FALSE      FALSE
## 1 subsets of each size up to 8
## Selection Algorithm: exhaustive
##          Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6 Comp.7 Comp.8
## 1  ( 1 ) "*"    " "    " "    " "    " "    " "    " "    " "
## 2  ( 1 ) "*"    " "    "*"    " "    " "    " "    " "    " "
## 3  ( 1 ) "*"    "*"    "*"    " "    " "    " "    " "    " "
## 4  ( 1 ) "*"    "*"    "*"    " "    " "    "*"    " "    " "
## 5  ( 1 ) "*"    "*"    "*"    "*"    " "    "*"    " "    " "
## 6  ( 1 ) "*"    "*"    "*"    "*"    "*"    "*"    " "    " "
## 7  ( 1 ) "*"    "*"    "*"    "*"    "*"    "*"    " "    "*"
## 8  ( 1 ) "*"    "*"    "*"    "*"    "*"    "*"    "*"    "*"
```

Para cada numero de variables posibles, toma el mejor conjunto. Usando tanto las variables originales como las componentes principales, cuando una variable esta entre las mejores para un numero determinado de variables, también lo está en los siguientes. Debido a esto, cuando hagamos la regresión hacia delante o hacia atrás, obtendremos los mismos resultados que con este método.

Veamos los coeficientes R^2 para los mejores subconjuntos obtenidos.

```
resumen=summary(modeloRL_mejorsub)
resumen$rsq #R2 aumenta con el número de predictores
```

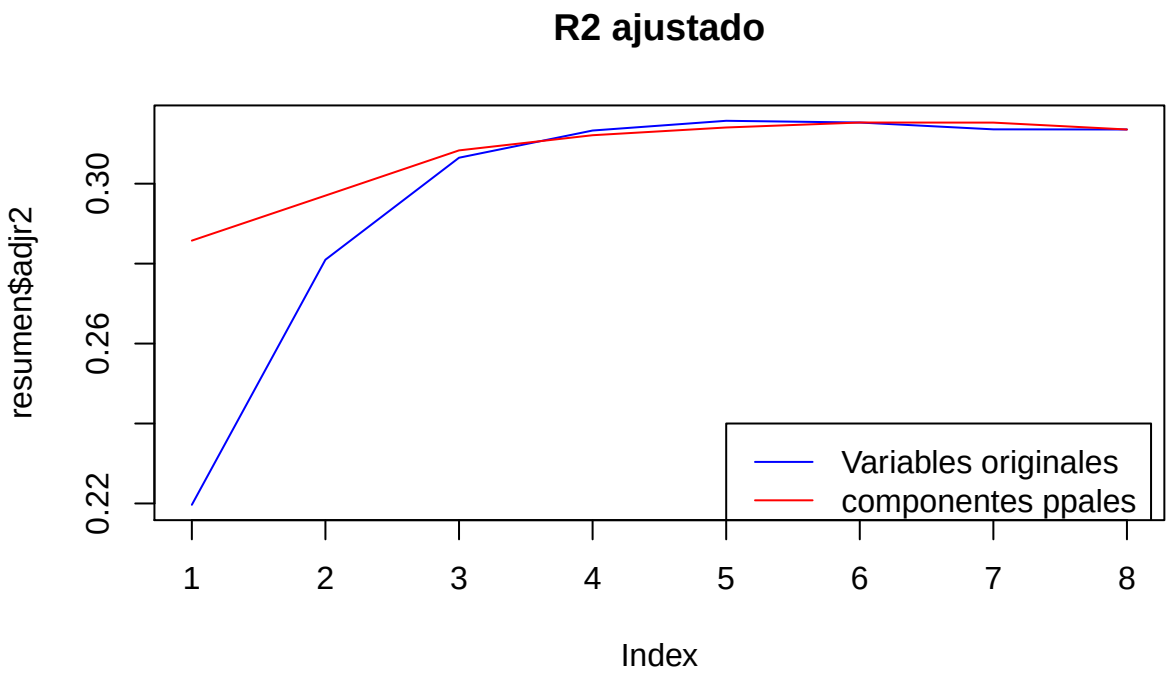
```
## [1] 0.2215996 0.2845510 0.3116588 0.3201114 0.3242236 0.3254951 0.3255152
## [8] 0.3271830
```

```
resumen_acp=summary(modeloRL_mejorsub_acp)
resumen_acp$rsq #R2 aumenta con el número de predictores
```

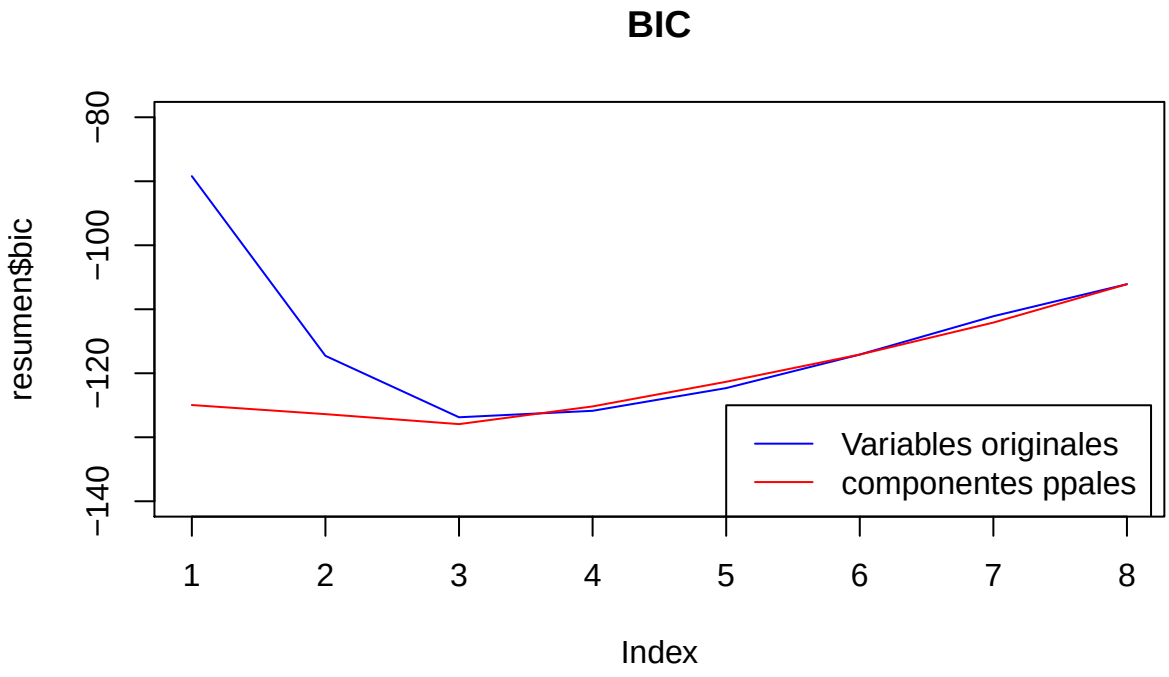
```
## [1] 0.2875322 0.3005116 0.3134795 0.3189526 0.3225682 0.3254924 0.3271712
## [8] 0.3271830
```

Representemos graficamente algunas medidas que penalicen la complejidad para distintos tamaños subconjuntos de variables.

```
plot(resumen$adjr2,type="l",col='blue',main='R2 ajustado')
lines(resumen_acp$adjr2,type="l",col='red')
legend(x=5,y=0.24,legend=c('Variables originales','componentes ppales'),
       lty=c(1,1),col=c('blue','red'))
```




```
plot(resumen$bic,type="l",col='blue',main='BIC',ylim=c(-140,-80))
lines(resumen_acp$bic,type="l",col='red')
legend(x=5,y=-125,legend=c('Variables originales','componentes ppales'),
       lty=c(1,1),col=c('blue','red'))
```



Viendo estos graficos, ya si que parece que merece la pena usar componentes principales. Veamos cuales son las mejores variables según el criterio BIC.

```
which.min(resumen$bic)
```

```
## [1] 3
```

```
which.min(resumen_acp$bic)
```

```
## [1] 3
```

```
compos<- which.min(resumen$bic)
compos_acp<- which.min(resumen_acp$bic)
vsel<- colnames(resumen$which)[resumen$which[compos,]]
vsel_acp<- colnames(resumen_acp$which)[resumen_acp$which[compos_acp,]]
vsel
```

```
## [1] "(Intercept)" "managers"    "middleempl"  "employrate"
```

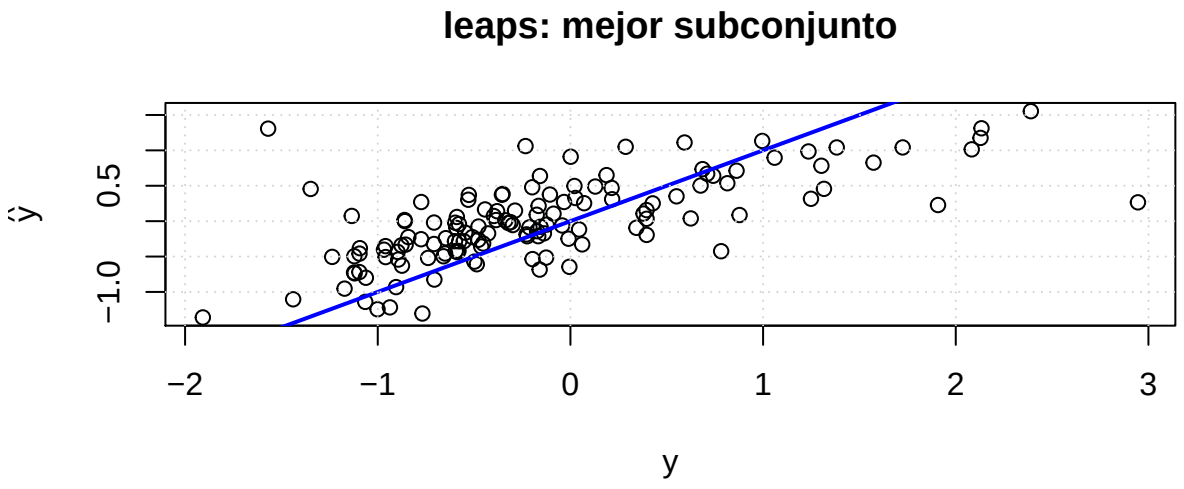
```
vsel_acp
```

```
## [1] "(Intercept)" "Comp.1"      "Comp.2"      "Comp.3"
```

Si usamos la transformación a componentes principales escogemos las tres primeras (mediante el criterio BIC), que además son las que mas varianza explican. Construyamos estos dos modelos para evaluarlos sobre el conjunto test.

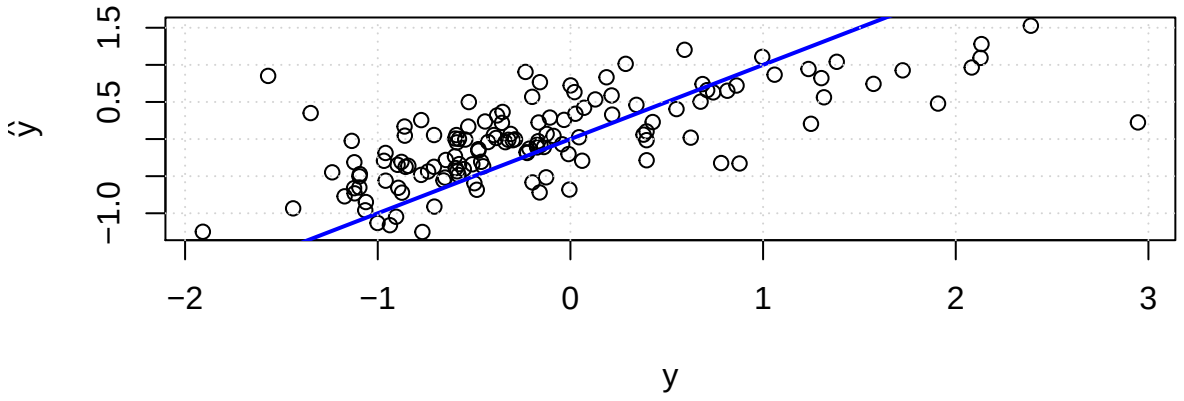
```
#tomamos las variables seleccionadas
vsel=vsel[-1]#quitamos (Intercept)
vsel_acp=vsel_acp[-1]
#construimos las formulas de los modelos
fmla <- as.formula(paste("income ~ ", paste(vsel, collapse= "+")))
fmla_acp <- as.formula(paste("income ~ ", paste(vsel_acp, collapse= "+")))
#construimos los modelos
modeloRL_mej<- lm(fmla,data=train_empleo)
modeloRL_mej_acp<- lm(fmla_acp,data=train_empleo_acp)
#evaluamos sobre conjunto test
predRLmej_test=predict(modeloRL_mej,newdata=test_empleo)
predRLmej_test_acp=predict(modeloRL_mej_acp,newdata=test_empleo_acp)
```

```
t(Ajuste(test_empleo$income,predRLmej_test,"leaps: mejor subconjunto"))
```




```
##       MSE      RMSE      R2
## [1,] 0.408528 0.6391619 0.473628
```

```
t(Ajuste(test_empleo_acp$income,predRLmej_test_acp,"leaps: mejor subconjunto sobre cp"))
```




```
##       MSE       RMSE      R2
## [1,] 0.4028893 0.6347356 0.4843665
```

Apesar de que no tenemos buenos resultados en ninguno de los dos modelos, estos parecen algo mejores usando componentes principales, y en cualquier caso, ambos mejoran los resultados de los modelos que utilizán todas las variables.

Regresión lineal: "forward".

Mediante este algoritmo en cada paso aumentamos en una variable nuestro modelo, eligiendo aquella que mejor resultados da al incluirse junto con las anteriores.

```
modeloRL_fw=regsubsets(income~.,data=train_empleo,
                       nvmax=8,method="forward")
modeloRL_fw_acp=regsubsets(income~.,data=train_empleo_acp,
                       nvmax=8,method="forward")
summary(modeloRL_fw)
```

```
## Subset selection object
## Call: regsubsets.formula(income ~ ., data = train_empleo, nvmax = 8,
##     method = "forward")
## 8 Variables  (and intercept)
##            Forced in Forced out
## farmers        FALSE      FALSE
## tradesmen      FALSE      FALSE
## managers       FALSE      FALSE
## workers        FALSE      FALSE
## unemployed     FALSE      FALSE
## middleempl     FALSE      FALSE
## retired        FALSE      FALSE
## employrate     FALSE      FALSE
## 1 subsets of each size up to 8
## Selection Algorithm: forward
##          farmers tradesmen managers workers unemployed middleempl retired
## 1  ( 1 ) " "     " "       "*"      " "     " "         " "         " "
## 2  ( 1 ) " "     " "       "*"      " "     " "         " "         " "
## 3  ( 1 ) " "     " "       "*"      " "     " "         "*"         " "
## 4  ( 1 ) " "     "*"       "*"      " "     " "         "*"         " "
## 5  ( 1 ) " "     "*"       "*"      " "     "*"         "*"         " "
## 6  ( 1 ) "*"     "*"       "*"      " "     "*"         "*"         " "
## 7  ( 1 ) "*"     "*"       "*"      " "     "*"         "*"         "*"
## 8  ( 1 ) "*"     "*"       "*"      "*"     "*"         "*"         "*"
##          employrate
## 1  ( 1 ) " "
## 2  ( 1 ) "*"
## 3  ( 1 ) "*"
## 4  ( 1 ) "*"
## 5  ( 1 ) "*"
## 6  ( 1 ) "*"
## 7  ( 1 ) "*"
## 8  ( 1 ) "*"
```

```
summary(modeloRL_fw_acp)
```

```
## Subset selection object
## Call: regsubsets.formula(income ~ ., data = train_empleo_acp, nvmax = 8,
##     method = "forward")
## 8 Variables  (and intercept)
##        Forced in Forced out
## Comp.1     FALSE      FALSE
## Comp.2     FALSE      FALSE
## Comp.3     FALSE      FALSE
```

```
## Comp.4     FALSE      FALSE
## Comp.5     FALSE      FALSE
## Comp.6     FALSE      FALSE
## Comp.7     FALSE      FALSE
## Comp.8     FALSE      FALSE
## 1 subsets of each size up to 8
## Selection Algorithm: forward
##          Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6 Comp.7 Comp.8
## 1  ( 1 ) "*"    " "    " "    " "    " "    " "    " "    " "
## 2  ( 1 ) "*"    " "    "*"    " "    " "    " "    " "    " "
## 3  ( 1 ) "*"    "*"    "*"    " "    " "    " "    " "    " "
## 4  ( 1 ) "*"    "*"    "*"    " "    " "    "*"    " "    " "
## 5  ( 1 ) "*"    "*"    "*"    "*"    " "    "*"    " "    " "
## 6  ( 1 ) "*"    "*"    "*"    "*"    "*"    "*"    " "    " "
## 7  ( 1 ) "*"    "*"    "*"    "*"    "*"    "*"    " "    "*"
## 8  ( 1 ) "*"    "*"    "*"    "*"    "*"    "*"    "*"    "*"
```

```
resumen=summary(modeloRL_fw)
resumen_acp=summary(modeloRL_fw_acp)
```

Tal como dijimos anteriormente, obtenemos los mismos resultados que con la busqueda exhaustiva de combinaciones de variables. De modo que no hace falta que veamos que medidas de bondad tenemos con dichas variables, pues son las mismas que las calculadas en el paso anterior.

Regresión lineal: "backward".

Cabe esperar que volvamos a obtener los mismos resultados que con los dos metodos anteriores de busqueda, comprovemoslo.

```
modeloRL_bw=regsubsets(income~.,data=train_empleo,
                       nvmax=8,method="backward")
modeloRL_bw_acp=regsubsets(income~.,data=train_empleo_acp,
                       nvmax=8,method="backward")
summary(modeloRL_bw)
```

```
## Subset selection object
## Call: regsubsets.formula(income ~ ., data = train_empleo, nvmax = 8,
##     method = "backward")
## 8 Variables  (and intercept)
##            Forced in Forced out
## farmers        FALSE      FALSE
## tradesmen      FALSE      FALSE
## managers       FALSE      FALSE
## workers        FALSE      FALSE
## unemployed     FALSE      FALSE
## middleempl     FALSE      FALSE
## retired        FALSE      FALSE
## employrate     FALSE      FALSE
## 1 subsets of each size up to 8
## Selection Algorithm: backward
##          farmers tradesmen managers workers unemployed middleempl retired
## 1  ( 1 ) " "     " "       "*"      " "     " "        " "        " "
## 2  ( 1 ) " "     " "       "*"      " "     " "        " "        " "
## 3  ( 1 ) " "     " "       "*"      " "     " "        "*"        " "
## 4  ( 1 ) " "     "*"       "*"      " "     " "        "*"        " "
```

```
## 5  ( 1 ) " "     "*"       "*"       " "     "*"        "*"        " "
## 6  ( 1 ) "*"     "*"       "*"       " "     "*"        "*"        " "
## 7  ( 1 ) "*"     "*"       "*"       " "     "*"        "*"        "*"
## 8  ( 1 ) "*"     "*"       "*"       "*"     "*"        "*"        "*"
##          employrate
## 1  ( 1 ) " "
## 2  ( 1 ) "*"
## 3  ( 1 ) "*"
## 4  ( 1 ) "*"
## 5  ( 1 ) "*"
## 6  ( 1 ) "*"
## 7  ( 1 ) "*"
## 8  ( 1 ) "*"
```

```
summary(modeloRL_bw_acp)
```

```
## Subset selection object
## Call: regsubsets.formula(income ~ ., data = train_empleo_acp, nvmax = 8,
##     method = "backward")
## 8 Variables  (and intercept)
##        Forced in Forced out
## Comp.1     FALSE      FALSE
## Comp.2     FALSE      FALSE
## Comp.3     FALSE      FALSE
## Comp.4     FALSE      FALSE
## Comp.5     FALSE      FALSE
## Comp.6     FALSE      FALSE
## Comp.7     FALSE      FALSE
## Comp.8     FALSE      FALSE
## 1 subsets of each size up to 8
## Selection Algorithm: backward
##          Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6 Comp.7 Comp.8
## 1  ( 1 ) "*"    " "    " "    " "    " "    " "    " "    " "
## 2  ( 1 ) "*"    " "    "*"    " "    " "    " "    " "    " "
## 3  ( 1 ) "*"    "*"    "*"    " "    " "    " "    " "    " "
## 4  ( 1 ) "*"    "*"    "*"    " "    " "    "*"    " "    " "
## 5  ( 1 ) "*"    "*"    "*"    "*"    " "    "*"    " "    " "
## 6  ( 1 ) "*"    "*"    "*"    "*"    "*"    "*"    " "    " "
## 7  ( 1 ) "*"    "*"    "*"    "*"    "*"    "*"    " "    "*"
## 8  ( 1 ) "*"    "*"    "*"    "*"    "*"    "*"    "*"    "*"
```

```
resumen=summary(modeloRL_bw)
resumen_acp=summary(modeloRL_bw_acp)
```

Efectivamente, los subconjuntos de variables son los mismos, de modo que no hace falta comprobar medidas de bondad de ajuste, esto ya se hizo en la busqueda exhaustiva sobre subconjuntos de variables.

Regresión lineal: "seqrep".

Veamos que subconjuntos se escogen con el método hacia delante y hacia atrás.

```
modeloRL_seq=regsubsets(income~.,data=train_empleo,
                        nvmax=8,method="seqrep")
modeloRL_seq_acp=regsubsets(income~.,data=train_empleo_acp,
                        nvmax=8,method="seqrep")
```

```
summary(modeloRL_seq)
```

```
## Subset selection object
## Call: regsubsets.formula(income ~ ., data = train_empleo, nvmax = 8,
##     method = "seqrep")
## 8 Variables  (and intercept)
##            Forced in Forced out
## farmers        FALSE      FALSE
## tradesmen      FALSE      FALSE
## managers       FALSE      FALSE
## workers        FALSE      FALSE
## unemployed     FALSE      FALSE
## middleempl     FALSE      FALSE
## retired        FALSE      FALSE
## employrate     FALSE      FALSE
## 1 subsets of each size up to 8
## Selection Algorithm: 'sequential replacement'
##          farmers tradesmen managers workers unemployed middleempl retired
## 1  ( 1 ) " "     " "       "*"      " "     " "        " "        " "
## 2  ( 1 ) " "     " "       "*"      " "     " "        " "        " "
## 3  ( 1 ) " "     " "       "*"      " "     " "        "*"        " "
## 4  ( 1 ) " "     "*"       "*"      " "     " "        "*"        " "
## 5  ( 1 ) " "     "*"       "*"      " "     "*"        "*"        " "
## 6  ( 1 ) "*"     "*"       "*"      "*"     "*"        "*"        " "
## 7  ( 1 ) "*"     "*"       "*"      " "     "*"        "*"        "*"
## 8  ( 1 ) "*"     "*"       "*"      "*"     "*"        "*"        "*"
##          employrate
## 1  ( 1 ) " "
## 2  ( 1 ) "*"
## 3  ( 1 ) "*"
## 4  ( 1 ) "*"
## 5  ( 1 ) "*"
## 6  ( 1 ) " "
## 7  ( 1 ) "*"
## 8  ( 1 ) "*"
```

```
summary(modeloRL_seq_acp)
```

```
## Subset selection object
## Call: regsubsets.formula(income ~ ., data = train_empleo_acp, nvmax = 8,
##     method = "seqrep")
## 8 Variables  (and intercept)
##        Forced in Forced out
## Comp.1     FALSE      FALSE
## Comp.2     FALSE      FALSE
## Comp.3     FALSE      FALSE
## Comp.4     FALSE      FALSE
## Comp.5     FALSE      FALSE
## Comp.6     FALSE      FALSE
## Comp.7     FALSE      FALSE
## Comp.8     FALSE      FALSE
## 1 subsets of each size up to 8
## Selection Algorithm: 'sequential replacement'
##          Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6 Comp.7 Comp.8
```

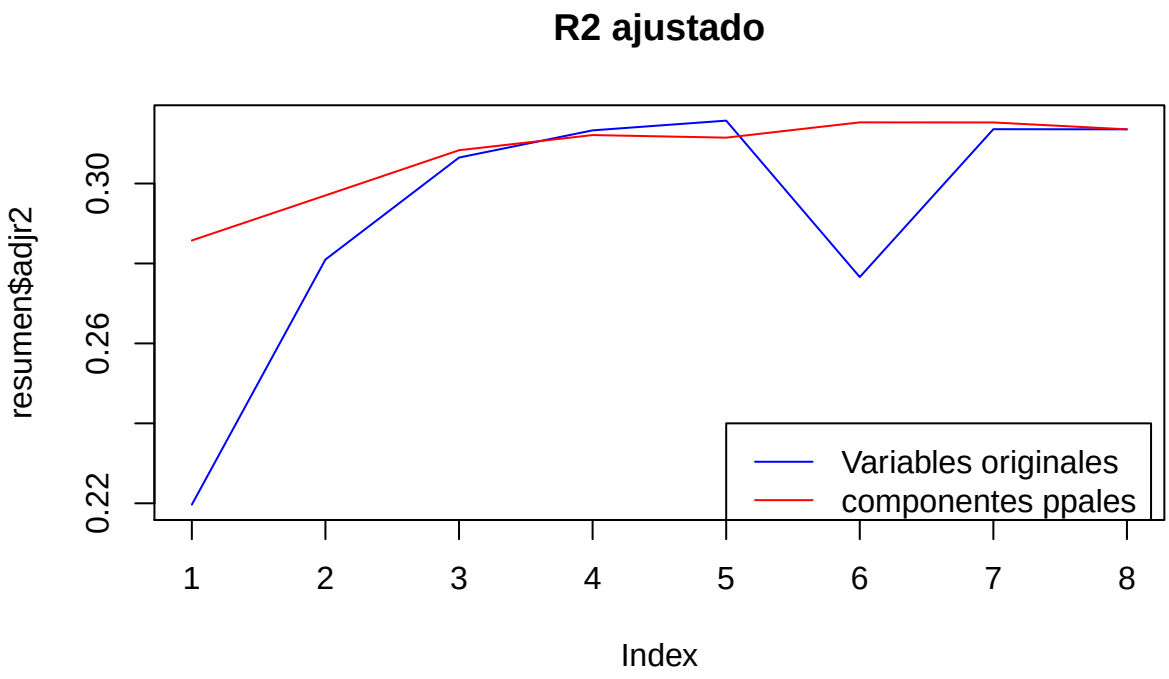
```
## 1  ( 1 ) "*"    " "    " "    " "    " "    " "    " "    " "
## 2  ( 1 ) "*"    " "    "*"    " "    " "    " "    " "    " "
## 3  ( 1 ) "*"    "*"    "*"    " "    " "    " "    " "    " "
## 4  ( 1 ) "*"    "*"    "*"    " "    " "    "*"    " "    " "
## 5  ( 1 ) "*"    "*"    "*"    "*"    "*"    " "    " "    " "
## 6  ( 1 ) "*"    "*"    "*"    "*"    "*"    "*"    " "    " "
## 7  ( 1 ) "*"    "*"    "*"    "*"    "*"    "*"    " "    "*"
## 8  ( 1 ) "*"    "*"    "*"    "*"    "*"    "*"    "*"    "*"
```

```
resumen=summary(modeloRL_seq)
resumen_acp=summary(modeloRL_seq_acp)
```

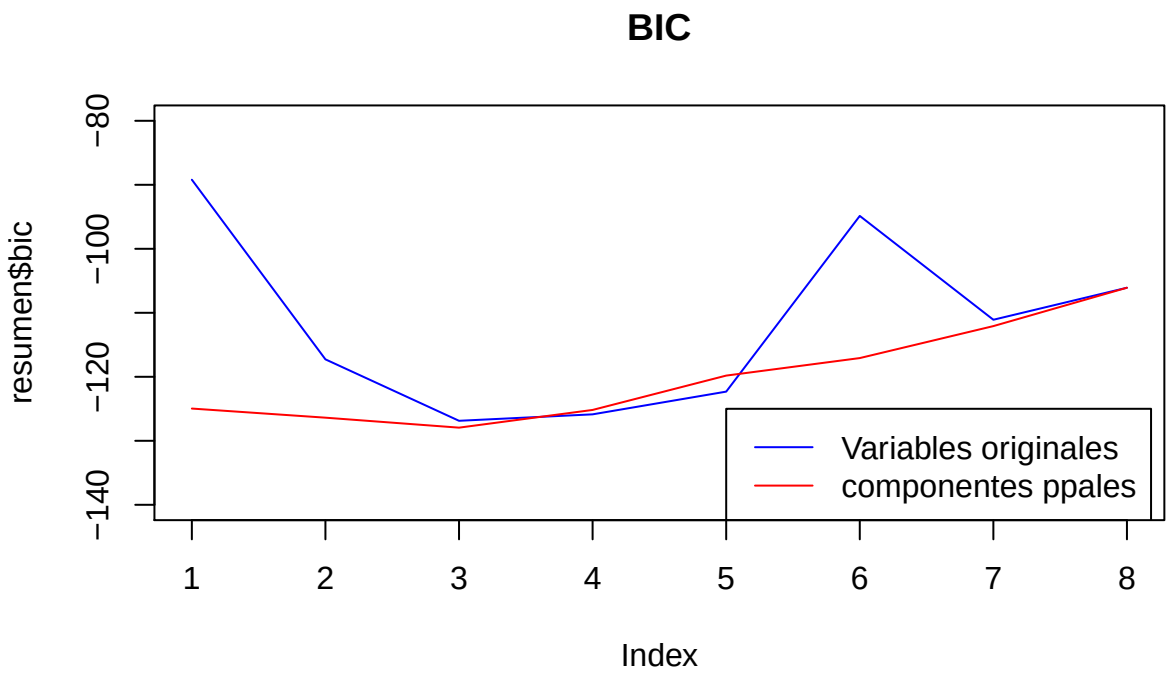
Sorprendentemente, esta vez no obtenemos los mismos resultados que con los métodos anteriores. Veamos entonces diversas medidas que penalicen la complejidad usando los resultados obtenidos.

tamaños subconjuntos de variables.

```
plot(resumen$adjr2,type="l",col='blue',main='R2 ajustado')
lines(resumen_acp$adjr2,type="l",col='red')
legend(x=5,y=0.24,legend=c('Variables originales','componentes ppales'),
       lty=c(1,1),col=c('blue','red'))
```




```
plot(resumen$bic,type="l",col='blue',main='BIC',ylim=c(-140,-80))
lines(resumen_acp$bic,type="l",col='red')
legend(x=5,y=-125,legend=c('Variables originales','componentes ppales'),
       lty=c(1,1),col=c('blue','red'))
```



Ademas de haber obtenido unos subconjuntos inesperados, vemos que la curva produce tambien un pico un tanto extraño para los modelos en los que no usamos componentes principales.

Basemonos en el criterio BIC para ver que subconjunto parece apropiado para crear un modelo lineal.

```
which.min(resumen$bic)
```

```
## [1] 3
```

```
which.min(resumen_acp$bic)
```

```
## [1] 3
```

```
compos<- which.min(resumen$bic)
compos_acp<- which.min(resumen_acp$bic)
vsel<- colnames(resumen$which)[resumen$which[compos,]]
vsel_acp<- colnames(resumen_acp$which)[resumen_acp$which[compos_acp,]]
vsel
```

```
## [1] "(Intercept)" "managers"    "middleempl"  "employrate"
```

```
vsel_acp
```

```
## [1] "(Intercept)" "Comp.1"      "Comp.2"      "Comp.3"
```

Tomaríamos las mismas variables que en la busqueda exhaustiva, luego no hace falta generar de nuevo los modelos.

Selección de variables mediante algoritmos genéticos.

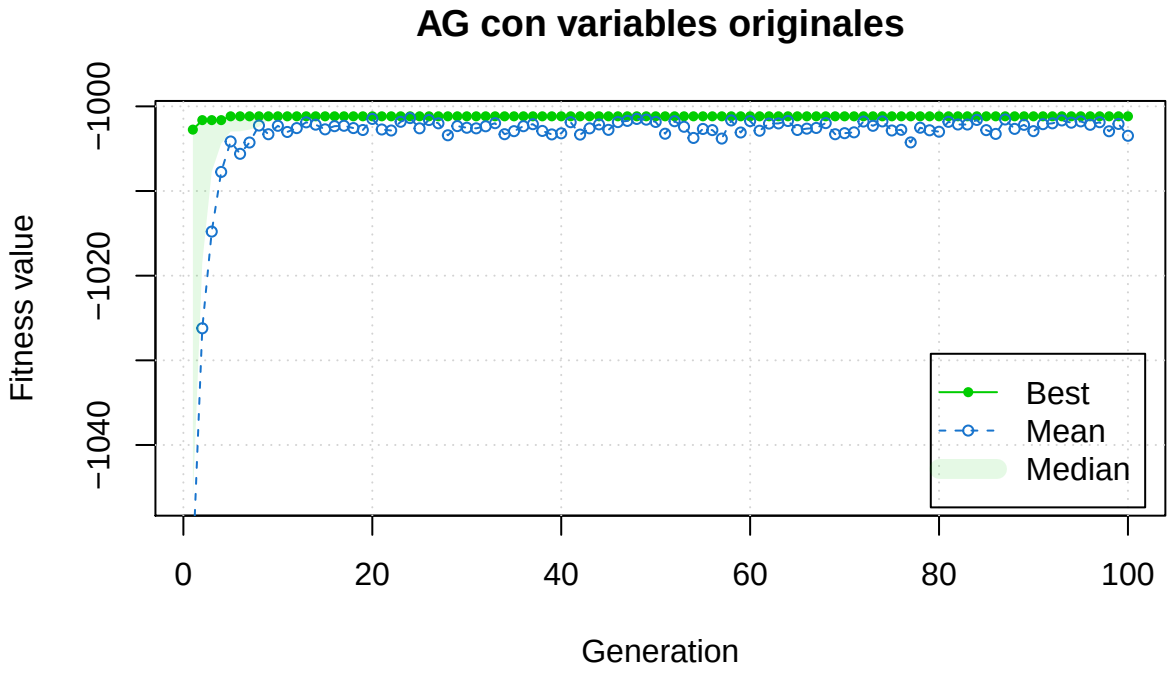
Carguemos una librería que permite el uso de estos algoritmos para seleccionar variables. Necesitamos ademas definir una función (fitnes), que indique los parametros de los modelos a ajustar, esta toma valores 0 y 1 para indicar si una variable se usa o no. A su vez, necesitamos dividir los datos en variables predictoras y en variable respuesta. Hagamos todo esto para los datos usando las variables originales y usando las componentes principales.

```
library(GA)

xent <- model.matrix(modeloRL)[,-1]
yent <- model.response(model.frame(modeloRL))

xent_acp <- model.matrix(modeloRL_acp)[,-1]
yent_acp <- model.response(model.frame(modeloRL_acp))

fitness <- function(string)
{
  inc <- which(string==1)
  X <- cbind(1, xent[,inc])
  mod <- lm.fit(X, yent)
  class(mod) <- "lm"
  -AIC(mod)   #ga es para maximizar
}

fitness_acp <- function(string)
{
  inc <- which(string==1)
  X <- cbind(1, xent_acp[,inc])
  mod <- lm.fit(X, yent_acp)
  class(mod) <- "lm"
  -AIC(mod)   #ga es para maximizar
}
```

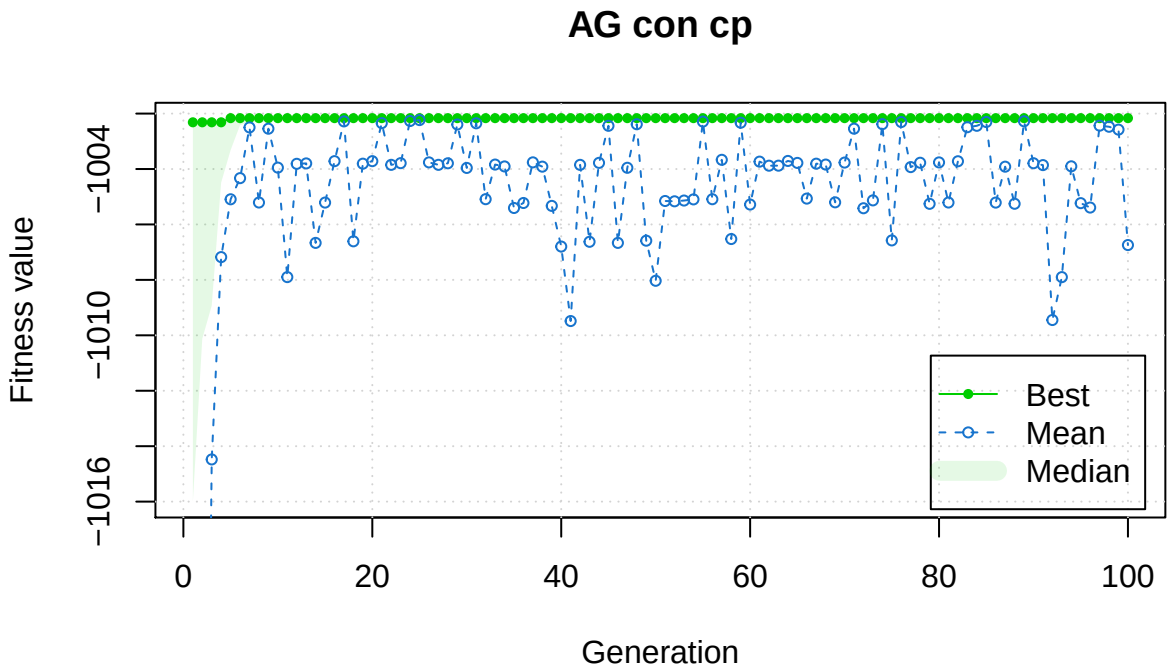
Ya podemos aplicar la busqueda de subconjuntos de variables mediante algoritmos genéticos.

```
AG <- ga("binary", fitness = fitness, nBits = ncol(xent),
         names = colnames(xent), monitor = F,
         popSize=100)
AG_acp <- ga("binary", fitness = fitness_acp, nBits = ncol(xent_acp),
         names = colnames(xent_acp), monitor = F,
         popSize=100)
```

```
plot(AG, main= 'AG con variables originales')
```




```
plot(AG_acp, main= 'AG con cp')
```



```
summary(AG)
```

```
## +-----------------------------------+
## |         Genetic Algorithm         |
## +-----------------------------------+
##
## GA settings:
## Type                  =  binary
## Population size       =  100
## Number of generations =  100
## Elitism               =  5
## Crossover probability =  0.8
## Mutation probability  =  0.1
##
## GA results:
## Iterations             = 100
## Fitness function value = -1001.176
## Solution =
##      farmers tradesmen managers workers unemployed middleempl retired
## [1,]       0         1        1       0          1          1       0
##      employrate
## [1,]          1
```

```
summary(AG_acp)
```

```
## +-----------------------------------+
## |         Genetic Algorithm         |
## +-----------------------------------+
##
## GA settings:
## Type                  =  binary
## Population size       =  100
## Number of generations =  100
## Elitism               =  5
## Crossover probability =  0.8
## Mutation probability  =  0.1
##
## GA results:
## Iterations             = 100
## Fitness function value = -1002.165
## Solution =
##      Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6 Comp.7 Comp.8
## [1,]      1      1      1      1      0      1      0      0
```

Tomemos las variables elegidas, y evaluemos los modelos resultantes de usarlas para regresiones lineales

```
#Modelo con las variables seleccionadas
vsel=colnames(AG@solution)[AG@solution==1]
vsel_acp=colnames(AG_acp@solution)[AG_acp@solution==1]

fmla <- as.formula(paste("income ~ ", paste(vsel, collapse= "+")))
fmla_acp <- as.formula(paste("income ~ ", paste(vsel_acp, collapse= "+")))
fmla
```

```
## income ~ tradesmen + managers + unemployed + middleempl + employrate
```

```
fmla_acp
```

```
## income ~ Comp.1 + Comp.2 + Comp.3 + Comp.4 + Comp.6
```

```
modeloRL_AG<- lm(fmla,data=train_empleo)
modeloRL_AG_acp<- lm(fmla_acp,data=train_empleo_acp)
summary(modeloRL_AG)
```

```
##
## Call:
## lm(formula = fmla, data = train_empleo)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.2873 -0.3850 -0.0887  0.2885  9.9636
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 3.606e-17  4.115e-02   0.000   1.0000
## tradesmen   9.499e-02  4.159e-02   2.284   0.0229 *
## managers    3.348e-01  4.573e-02   7.322 1.37e-12 ***
## unemployed  6.604e-02  4.243e-02   1.556   0.1204
## middleempl  1.872e-01  4.427e-02   4.229 2.91e-05 ***
## employrate  2.404e-01  4.557e-02   5.276 2.18e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8272 on 398 degrees of freedom
## Multiple R-squared:  0.3242, Adjusted R-squared:  0.3157
## F-statistic: 38.19 on 5 and 398 DF,  p-value: < 2.2e-16
```

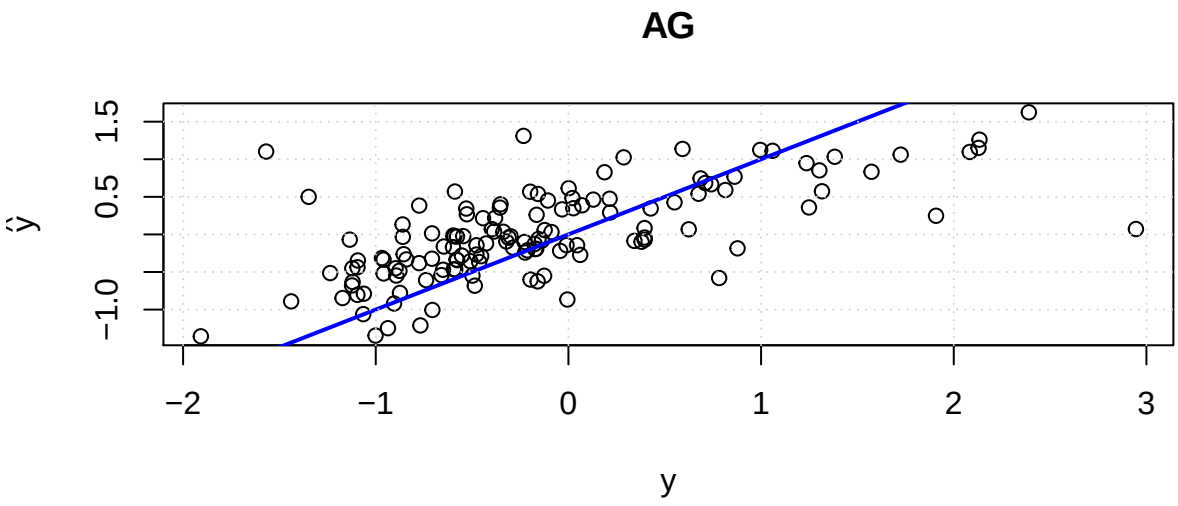
```
summary(modeloRL_AG_acp)
```

```
##
## Call:
## lm(formula = fmla_acp, data = train_empleo_acp)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.1384 -0.3892 -0.0873  0.2775 10.0798
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.414e-16  4.121e-02   0.000  1.00000
## Comp.1      -3.827e-01  2.945e-02 -12.997  < 2e-16 ***
## Comp.2      -9.211e-02  3.337e-02  -2.760  0.00604 **
## Comp.3       9.823e-02  3.557e-02   2.761  0.00602 **
## Comp.4       5.878e-02  4.033e-02   1.457  0.14578
## Comp.6       9.273e-02  5.171e-02   1.793  0.07370 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8282 on 398 degrees of freedom
## Multiple R-squared:  0.3226, Adjusted R-squared:  0.3141
## F-statistic:  37.9 on 5 and 398 DF,  p-value: < 2.2e-16
```

```
predRLAG_test=predict(modeloRL_AG,newdata=test_empleo)
predRLAG_test_acp=predict(modeloRL_AG_acp,newdata=test_empleo_acp)
```

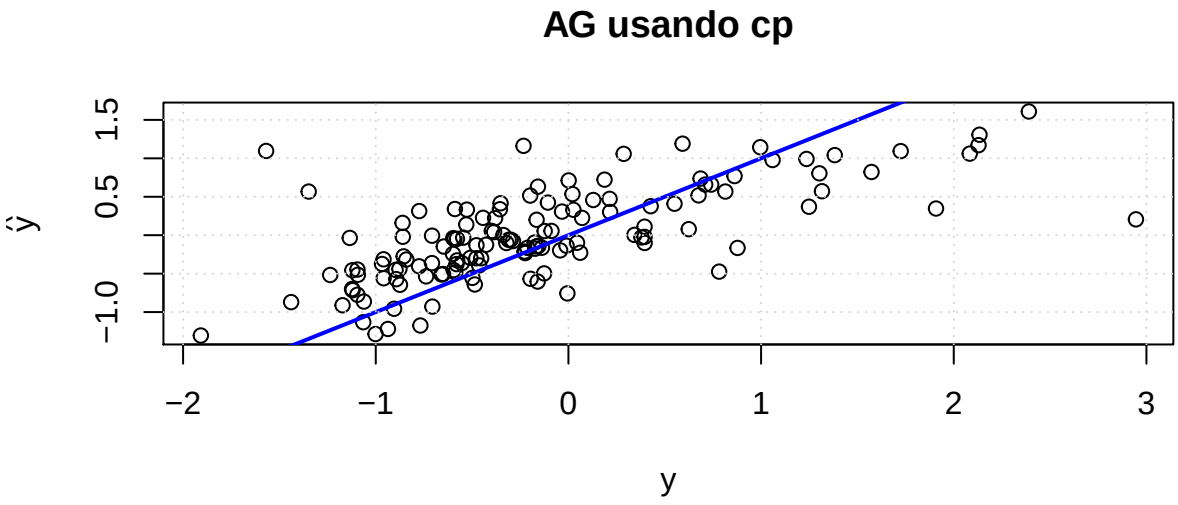
Veamos que bondad de ajuste tenemos usando los modelos que acabamos de construir sobre los datos test.

```
t(Ajuste(test_empleo$income,predRLAG_test,"AG"))
```




```
##      MSE       RMSE      R2
## [1,] 0.4286738 0.6547319 0.4477575
```

```
t(Ajuste(test_empleo_acp$income,predRLAG_test_acp,"AG usando cp"))
```




```
##      MSE       RMSE      R2
## [1,] 0.4031733 0.6349593 0.480607
```

Esta seleccion es algo peor que la obtenida mediante una busqueda exhaustiva. Aun así, usando componentes principales, el resultado es practicamente idéntico.

Ejercicio 3.

Completar la construcción de un árbol de clasificación correspondiente al fichero de instrucciones "EjemploLABrpart_default.r", dentro del material correspondiente a Árboles de Clasificación y Regresión.

Variables del conjunto de datos:

default (No/Yes): el cliente presenta números rojos en la tarjeta de crédito

student (No/Yes)

balance:saldo medio tras el pago mensual

income: ingresos

Objetivos:

El objetivo es construir un clasificador que prediga si un cliente presenta numeros rojos en la tarjeta de crédito. El banco prefiere evitar tarjetas "deudoras", se va a conisderar una matriz de costes. Coste de clasificar No como Yes es 5 veces superior a clasificar Yes como No.

Construir un árbol de clasificación considerando los costes mencionados y aplicando el prodceimiento de recorte 1-ES. Evaluar el modelo (acierto, sensitividad, especificidad).

Carguemos las librerias que utilizarémos para trabajar con arboles

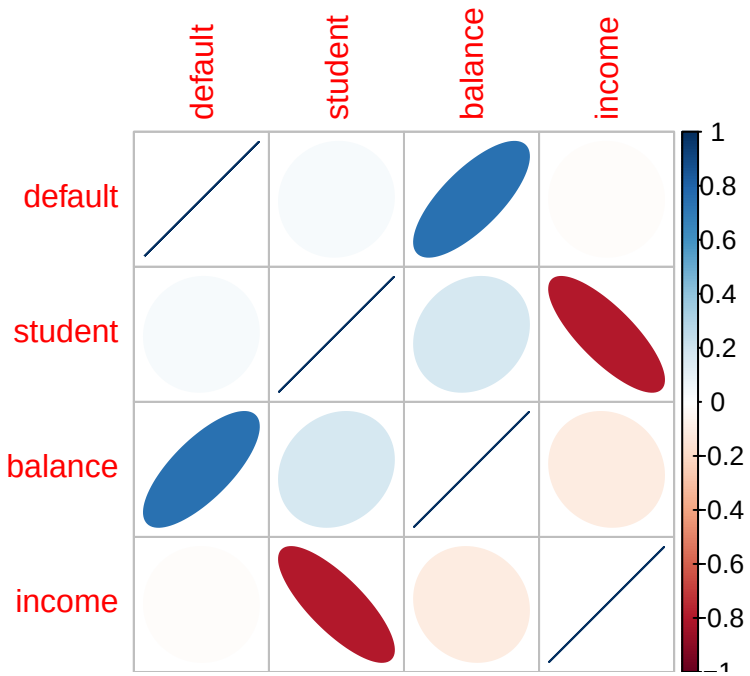
```
library(rpart)
library(rpart.plot)
```

Lectura de datos y pequeño análisis exploratorio.

```
Default=read.table(file=
  "C:/Users/AlvaroSanchez91/Desktop/Master Big Data Sevilla/ML1 Machine Learning I/6. Árboles de Clasifi
                    ,header=TRUE)
```

Antes de dividir el conjunto de datos en entrenamiento y test, parece razonable hacer un rapido analisis exploratorio del mismo.

```
Default2=within(Default, {default <- as.numeric(default)-1
  student <- as.numeric(student)-1})
R=cor(Default2)#no es lo mas apropiado
library(corrplot)
corrplot(R, method = 'ellipse')
```



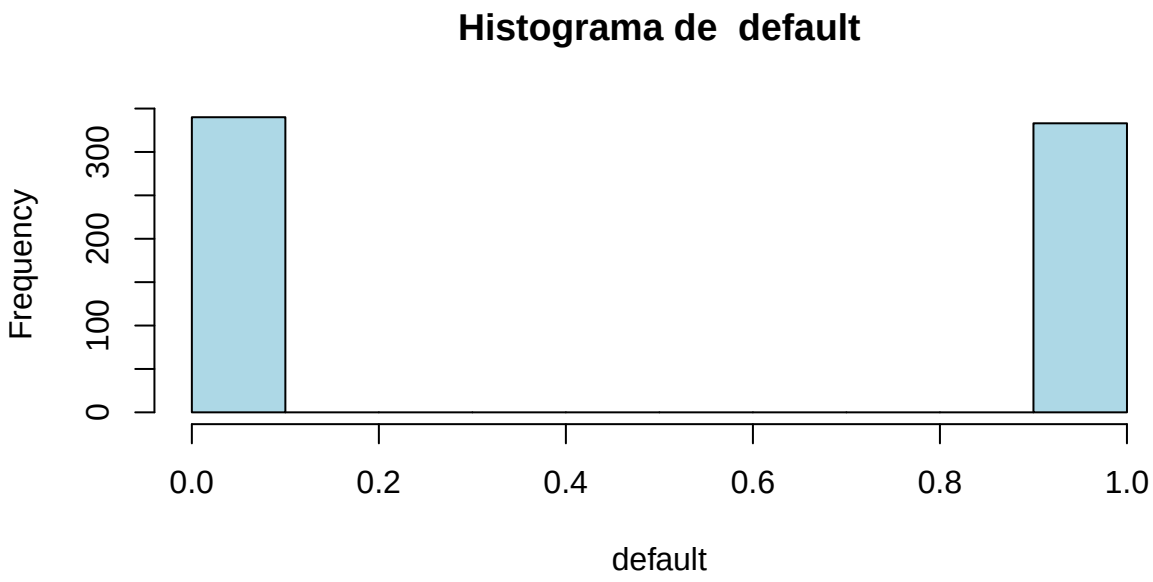

```
head(Default)
```

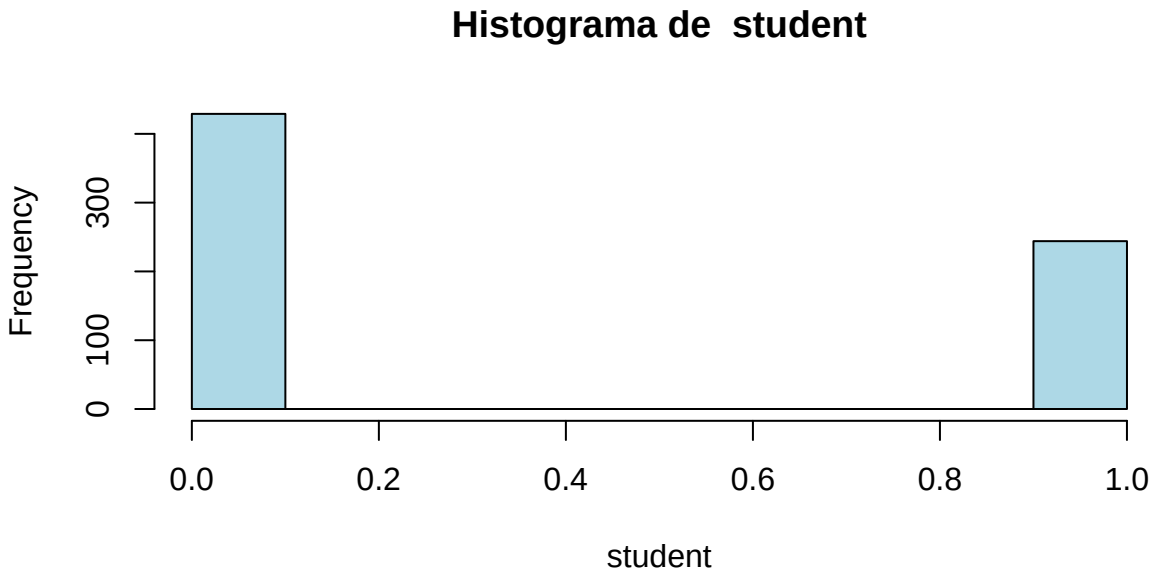
```
##   default student   balance   income
## 1      No     Yes  700.3352 15905.21
## 2      No      No 1095.0727 26464.63
## 3      No     Yes  256.3257 15627.66
## 4      No      No 1716.5954 51056.87
## 5     Yes      No 2063.5719 37372.76
## 6      No     Yes  824.6166 10062.58
```

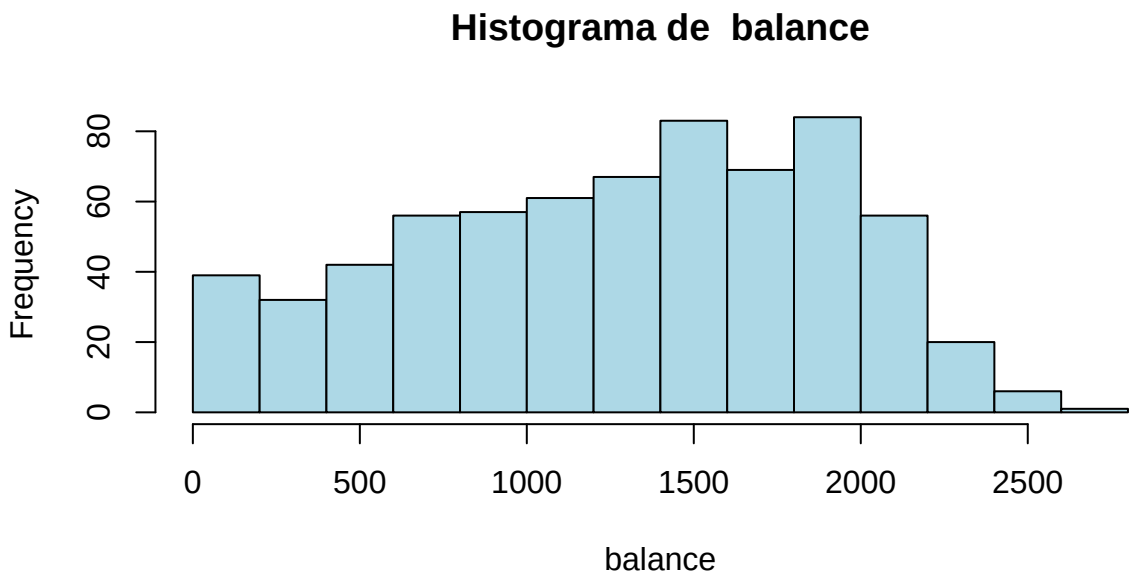
```
dim(Default)
```

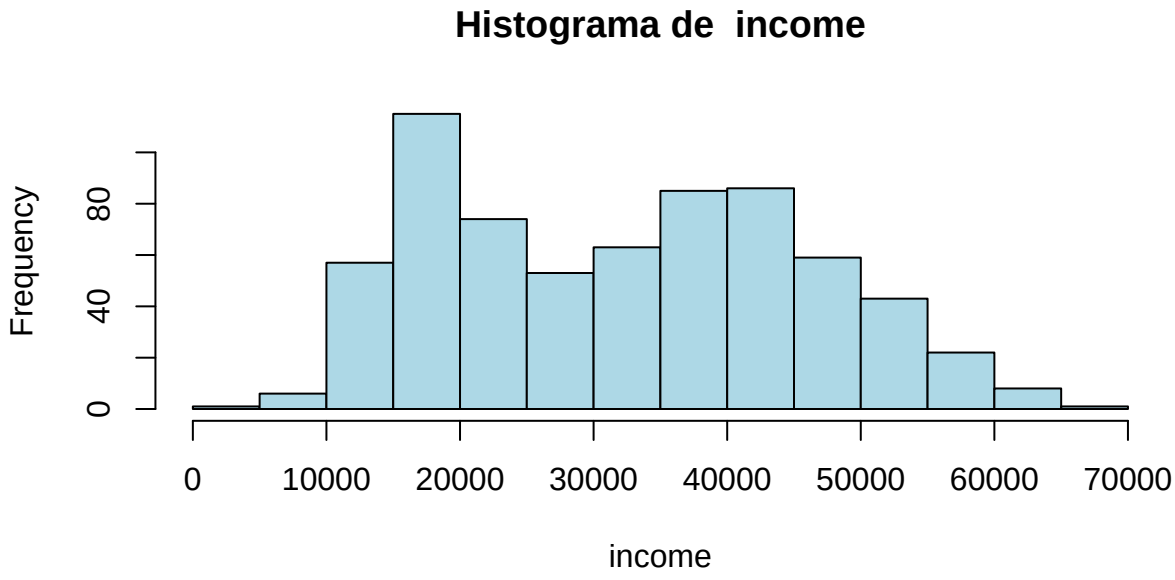
```
## [1] 673   4
```

```
for (i in 1:4){
  hist(Default2[,i],
       col = 'lightblue', xlab = colnames(Default)[i],
       main=paste('Histograma de ', colnames(Default)[i]))
}
```

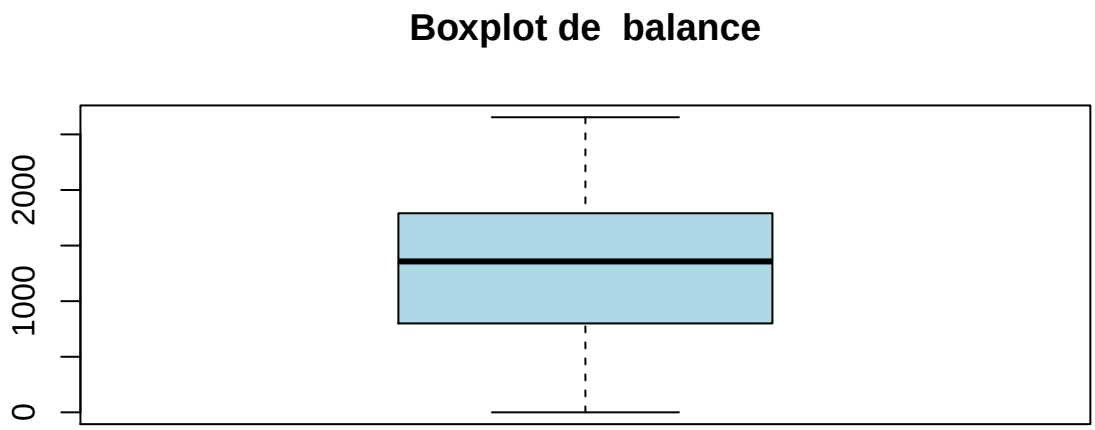

Histograma de default
Frequency
0
100
200
300
0.0
0.2
0.4
0.6
0.8
1.0
default

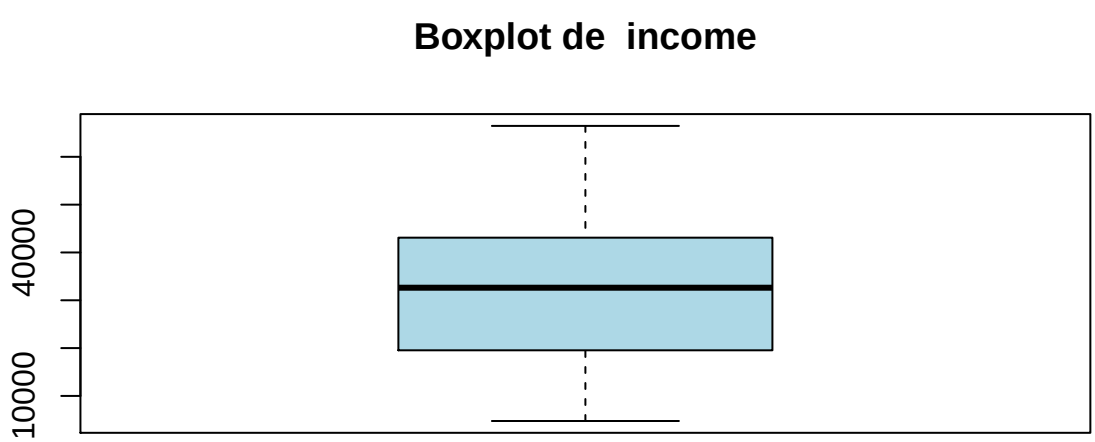

Histograma de student
Frequency
0
100
300
0.0
0.2
0.4
0.6
0.8
1.0
student






```
for (i in 3:4){
  boxplot(Default[,i], col = 'lightblue',
          xlab = colnames(Default)[i],
          main=paste('Boxplot de ', colnames(Default)[i]))
}
```






```
sum(is.na(Default))
```

```
## [1] 0
```

Definamos la matriz de costes.

```
L=matrix(c(0,1,5,0),2,2)
rownames(L)=colnames(L)=levels(Default$default)
L
```

```
##     No Yes
## No   0   5
## Yes  1   0
```

Separacion en conjuntos de entrenamiento y test.

```
set.seed(123)
n<- nrow(Default)
nent<- ceiling(0.7*n)
indient=sample(n,nent)

train=Default[indient,]
test=Default[-indient,]
```

Primer arbol de clasificación.

Creamos un arbol de clasificación sobre los datos de entrenamiento indicando la matriz de costes definida.

```
default.rpart_complete <- rpart(default~., data=train,
                     method="class",cp=0,parms=list(loss=L))
```

Podemos ver las reglas que hemos generado tras ajustar el modelo.

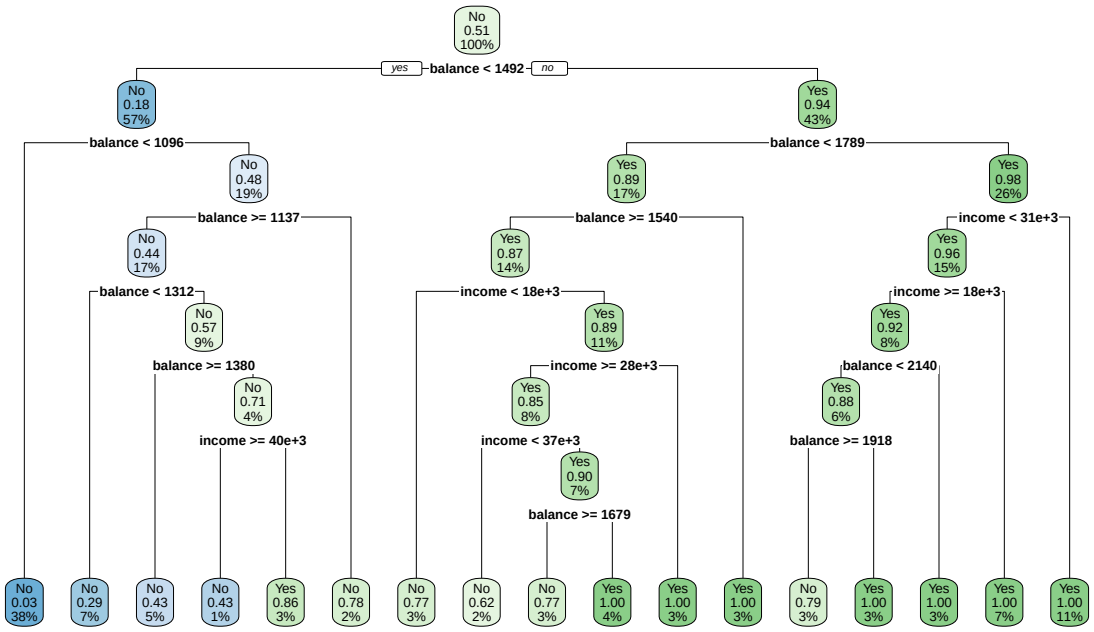
```
default.rpart_complete
```

```
## n= 472
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##   1) root 472 240 No (0.49152542 0.50847458)
##     2) balance< 1492.335 268  48 No (0.82089552 0.17910448)
##       4) balance< 1095.791 180   6 No (0.96666667 0.03333333) *
##       5) balance>=1095.791 88  42 No (0.52272727 0.47727273)
##        10) balance>=1137.209 79  35 No (0.55696203 0.44303797)
##          20) balance< 1312.482 35  10 No (0.71428571 0.28571429) *
##          21) balance>=1312.482 44  25 No (0.43181818 0.56818182)
##            42) balance>=1379.787 23  10 No (0.56521739 0.43478261) *
##            43) balance< 1379.787 21  15 No (0.28571429 0.71428571)
##              86) income>=40158.62 7   3 No (0.57142857 0.42857143) *
##              87) income< 40158.62 14  10 Yes (0.14285714 0.85714286) *
##        11) balance< 1137.209 9   7 No (0.22222222 0.77777778) *
##     3) balance>=1492.335 204  60 Yes (0.05882353 0.94117647)
##       6) balance< 1788.541 79  45 Yes (0.11392405 0.88607595)
##        12) balance>=1539.914 67  45 Yes (0.13432836 0.86567164)
##          24) income< 17540.17 13  10 No (0.23076923 0.76923077) *
##          25) income>=17540.17 54  30 Yes (0.11111111 0.88888889)
##            50) income>=27507.36 39  30 Yes (0.15384615 0.84615385)
##             100) income< 36551.28 8   5 No (0.37500000 0.62500000) *
##             101) income>=36551.28 31  15 Yes (0.09677419 0.90322581)
##               202) balance>=1679.468 13  10 No (0.23076923 0.76923077) *
##               203) balance< 1679.468 18   0 Yes (0.00000000 1.00000000) *
##            51) income< 27507.36 15   0 Yes (0.00000000 1.00000000) *
##        13) balance< 1539.914 12   0 Yes (0.00000000 1.00000000) *
##       7) balance>=1788.541 125  15 Yes (0.02400000 0.97600000)
##        14) income< 30824.43 72  15 Yes (0.04166667 0.95833333)
##          28) income>=17966.03 38  15 Yes (0.07894737 0.92105263)
##            56) balance< 2139.677 26  15 Yes (0.11538462 0.88461538)
```

```
##             112) balance>=1917.609 14  11 No (0.21428571 0.78571429) *
##             113) balance< 1917.609 12   0 Yes (0.00000000 1.00000000) *
##           57) balance>=2139.677 12   0 Yes (0.00000000 1.00000000) *
##         29) income< 17966.03 34   0 Yes (0.00000000 1.00000000) *
##       15) income>=30824.43 53   0 Yes (0.00000000 1.00000000) *
```

Esto no resulta nada facil de leer, de modo que podemos recurrir a una grafica que nos ayude.

```
rpart.plot(default.rpart_complete,main="Arbol con datos Default",
     uniform=TRUE)
```



Regla 1-ES.

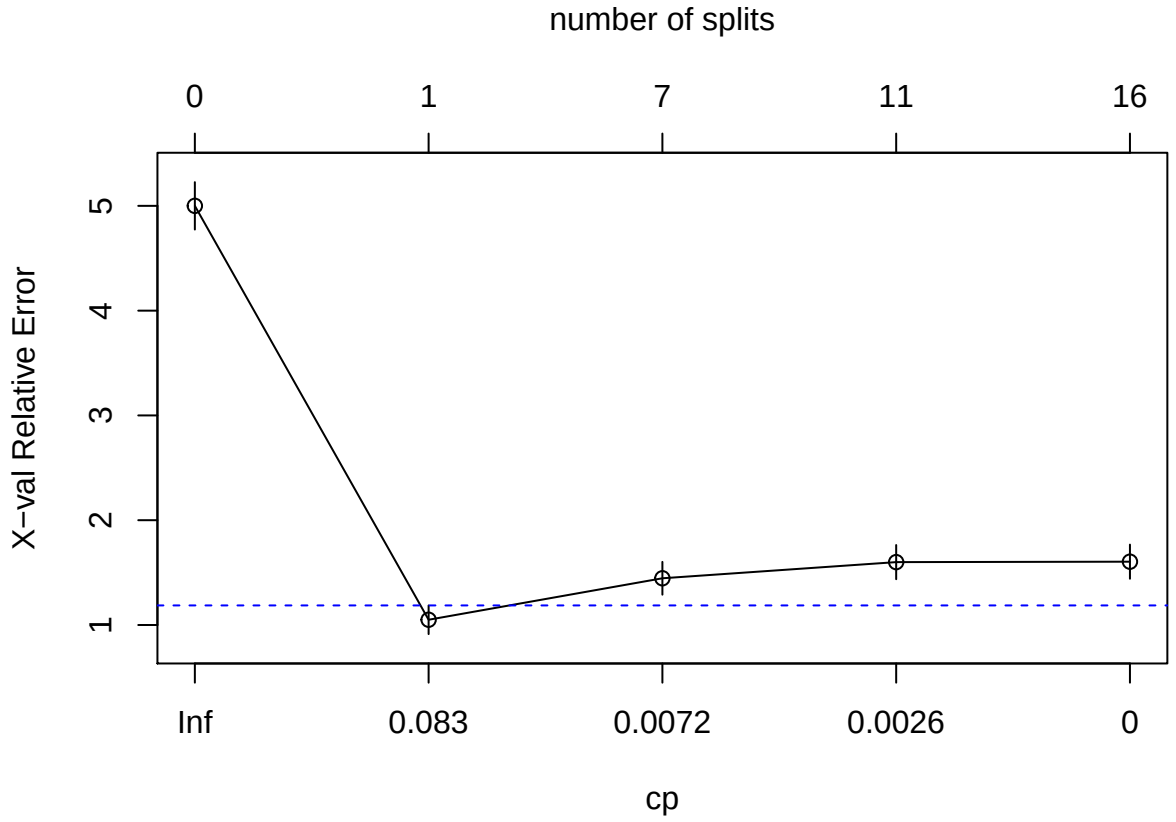
Veamos unos errores que nos da la propia función para distintos valores del parametro de complegidad cp. Este influye a la hora de decidir si dividir un nodo, cuanto mayor sea mas penalizará dividir un nodo lejano de la raiz. Así que podremos controlar el tamaño del arbol y, por tanto, el sobreajuste.

```
printcp(default.rpart_complete,digits=3)
```

```
##
## Classification tree:
## rpart(formula = default ~ ., data = train, method = "class",
##     parms = list(loss = L), cp = 0)
##
## Variables actually used in tree construction:
## [1] balance income
##
```

```
## Root node error: 240/472 = 0.508
##
## n= 472
##
##         CP nsplit rel error xerror  xstd
## 1 0.55000      0     1.000   5.00 0.226
## 2 0.01250      1     0.450   1.05 0.137
## 3 0.00417      7     0.367   1.45 0.157
## 4 0.00167     11     0.350   1.60 0.163
## 5 0.00000     16     0.342   1.60 0.163
```

```
plotcp(default.rpart_complete,lty=2,upper="splits",col="blue")
```



Mediante la regla 1-ES elegimos el parametro de complejidad cp, pues no es recomendable tomar directamente aquel que tiene un error de validación cruzada menor. Se tiene en cuenta la varianza de dichos errores de validación cruzada para elegir el parametro.

```
#Tabla con las estimaciones VC
cptabla<- default.rpart_complete$cptable
#Regla 1-ES
CP1ES<- min(cptabla[,4])+cptabla[which.min(cptabla[,4]),5]
CP1ES
```

```
## [1] 1.186769
```

```
cptabla[cptabla[,4]<CP1ES,]
```

```
##         CP    nsplit rel error    xerror      xstd
## 0.0125000 1.0000000 0.4500000 1.0500000 0.1367694
```

```
cprecorte<- cptabla[cptabla[,4]<CP1ES,][1]
cprecorte
```

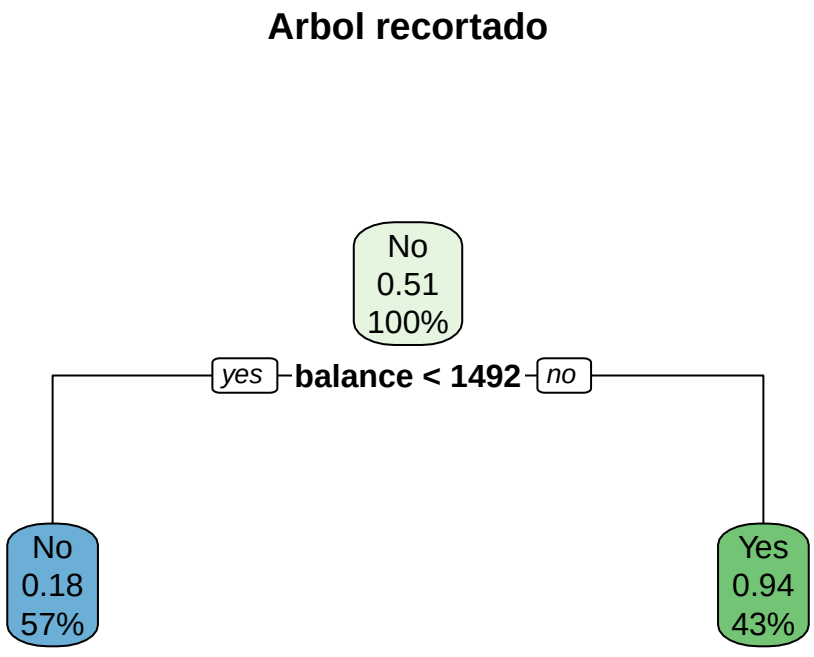
```
##     CP
## 0.0125
```

Recortemos el arbol original tomando este cp.

```
default.rpart1es<-prune.rpart(default.rpart_complete,cp=cprecorte)
default.rpart1es
```

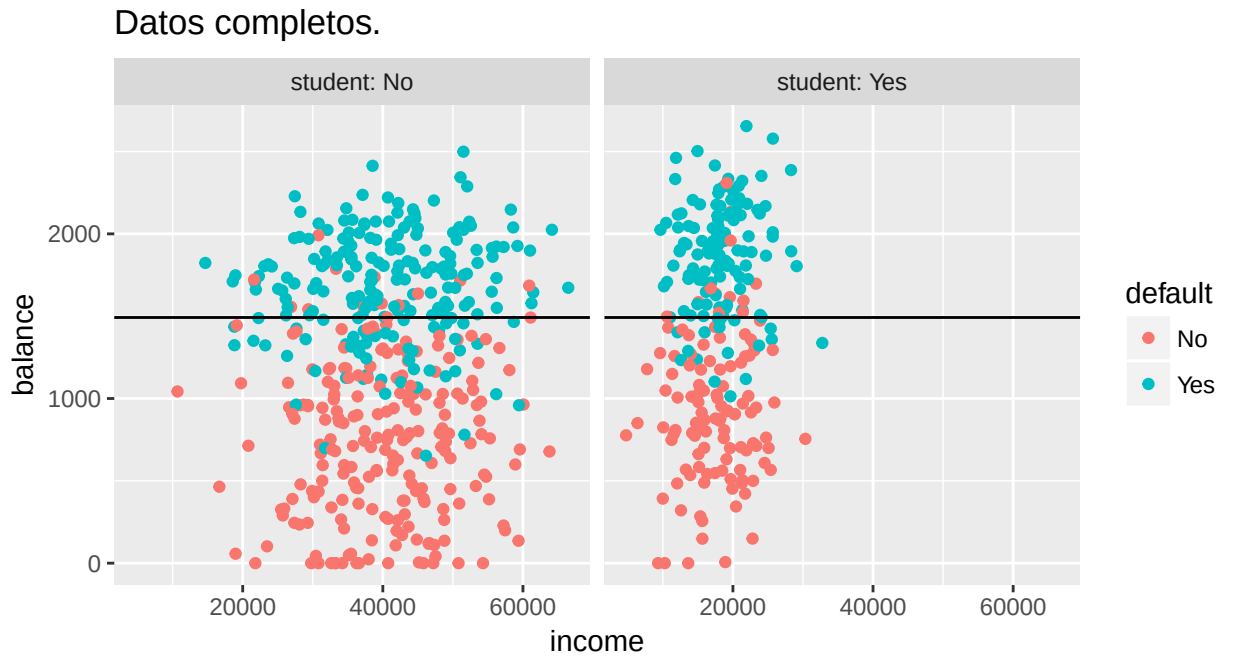
```
## n= 472
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
## 1) root 472 240 No (0.49152542 0.50847458)
##   2) balance< 1492.335 268  48 No (0.82089552 0.17910448) *
##   3) balance>=1492.335 204  60 Yes (0.05882353 0.94117647) *
```

```
rpart.plot(default.rpart1es,main="Arbol recortado",uniform=TRUE)
```

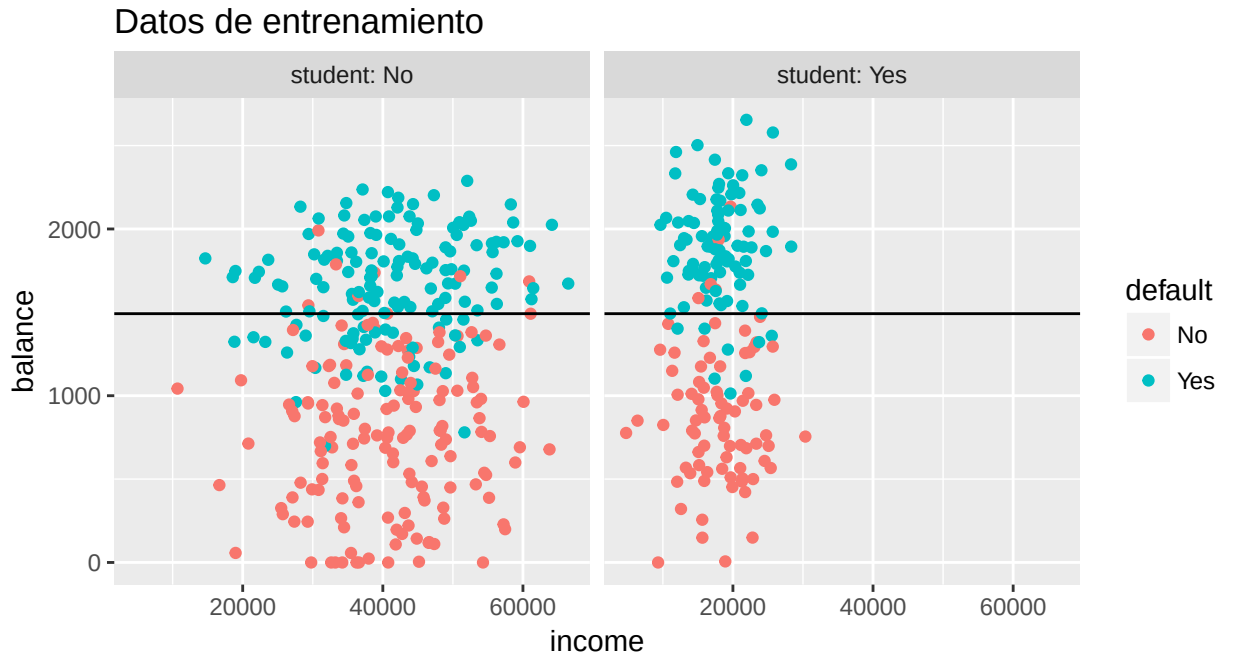


Este metodo nos lleva a un arbol consistente en una única division sobre la variable balance. Veamos una nuve de puntos para ver si este corte basandose en solo una variable es razonable.

```
library(ggplot2)
ggplot(data=Default,aes(x=income,y=balance,color=default))+
  geom_point()+facet_wrap(~student,labeller = "label_both")+
  ggtitle("Datos completos.")+geom_hline(yintercept=1492)
```




```
ggplot(data=train,aes(x=income,y=balance,color=default))+
  geom_point()+facet_wrap(~student,labeller = "label_both")+
  ggtitle('Datos de entrenamiento')+geom_hline(yintercept=1492)
```



Aparte de balance, podríamos decir que student afectaría ligeramente. Pero en general se ve como un separador tan simple parece razonable.

Evaluación del modelo.

Veamos que tal se comporta el modelo sobre el conjunto test.

```
predicciones=predict(default.rpart1es,test, type = 'class')
valor_real=test$default
matriz_confusion=table(valor_real,predicciones)
matriz_confusion
```

```
##           predicciones
## valor_real No Yes
##        No  94  14
##        Yes 24  69
```

```
prop.table(matriz_confusion, 1)
```

```
##           predicciones
## valor_real        No       Yes
##        No  0.8703704 0.1296296
##        Yes 0.2580645 0.7419355
```

```
diag(prop.table(matriz_confusion, 1))#Especificidad y sensibilidad respectivamente.
```

```
##        No       Yes
## 0.8703704 0.7419355
```

Vemos que tenemos mayor especificidad que sensibilidad, lo cual es razonable, ya que damos mas importancia a tener un falso positivo que un falso negativo.

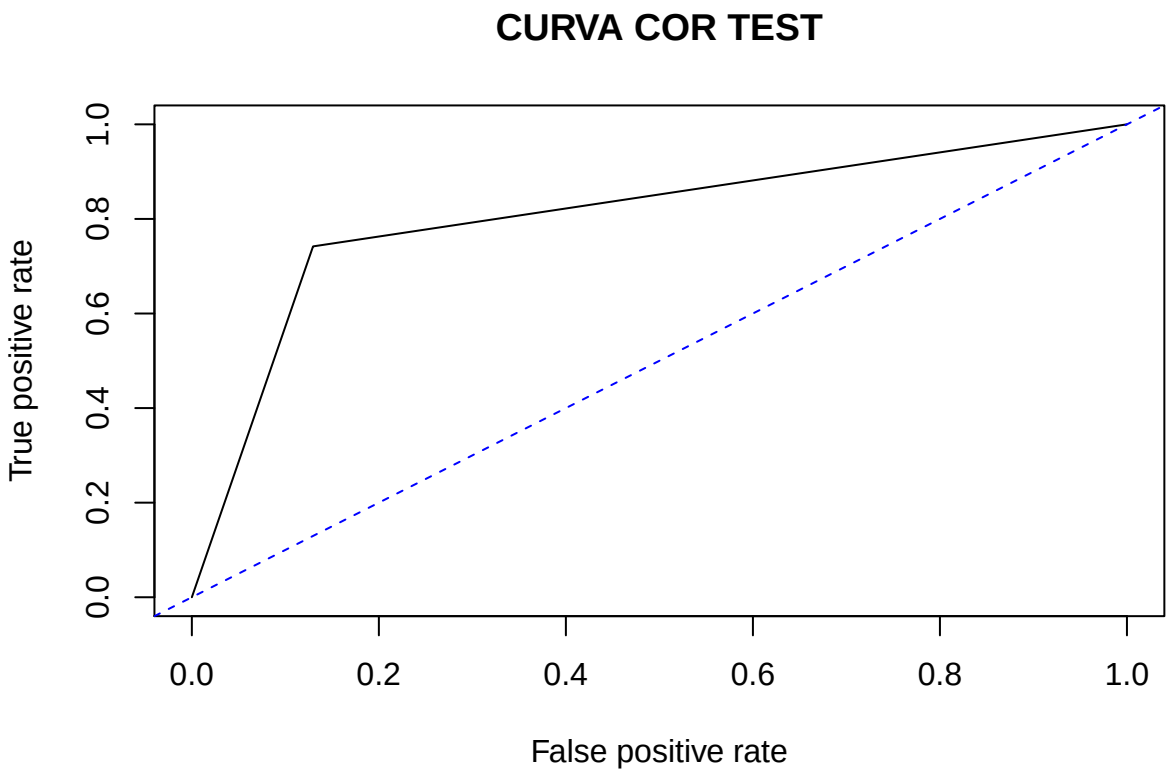
```
sum(diag(prop.table(matriz_confusion)))
```

```
## [1] 0.8109453
```

Tenemos un acierto totla del 81%.

Area bajo la curva operativa caracteristica:

```
library(ROCR)
probabi<- predict(default.rpart1es,test,type="prob")[,2] #Prob. yes
prediobj<-prediction(probabi,test$default)
plot(performance(prediobj, "tpr","fpr"),main="CURVA COR TEST")
abline(a=0,b=1,col="blue",lty=2)
```




```
auc<- as.numeric(performance(prediobj,"auc")@y.values)
cat("AUC test= ",auc ,"\n")
```

```
## AUC test=  0.8061529
```

Observemos que nuestro arblol, al ser tan simple, solo da dos opciones de probabilidades (una para cada clasificación). Por esto, al hacer la curva roc tenemos este único pico. En cualquier caso, la medida AUC parece relativamente buena.

Calcular en el conjunto test el indicador EMC:

Expected Miscalssification cost=P[No]P[Yes/No]coste[Yes/No]+P[Yes]P[No/Yes]coste[No/Yes]

```
P_No=prop.table(table(Default$default))[1]#P[No]
P_Yes=prop.table(table(Default$default))[2]#P[Yes]
P_Yes.No=(1-diag(prop.table(matriz_confusion, 1)))[1]#P[Yes/No]
P_No.Yes=(1-diag(prop.table(matriz_confusion, 1)))[2]#P[No/Yes]
cost_Yes.No=L[2,1]#coste[Yes/No]
cost_No.Yes=L[1,2]#coste[No/Yes]

EMC=as.numeric(P_No*P_Yes.No*cost_Yes.No+P_Yes*P_No.Yes*cost_No.Yes)
EMC
```

```
## [1] 0.7039398
```