Práctica AE

Antonio Galián Gálvez

2024-10-28

```
# Cargamos las librerías
library(gplots)
```

```
##
## Attaching package: 'gplots'
```

```
## The following object is masked from 'package:stats':
##
##     lowess
```

```
library(corrplot)
```

```
## corrplot 0.95 loaded
```

```
library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.1-8
```

```
library(MASS)
```

0. Cargamos los datos y eliminamos la columna train

```
# Cargamos los datos con separador de tabulador
datos <- read.delim("prostate.data.txt", header = TRUE, sep = "\t")

# Eliminamos la columna train
datos <- datos[, -ncol(datos)]
```

1. Exploración de datos

```
# Vemos las variables que hay
ncol(datos)
```

```
## [1] 10
```

```
# Eliminamos la columna id
datos <- datos[, -1]

# Comprobamos si hay NA
sum(is.na(datos))
```

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

```r
# Comprobamos si las variables estan estandarizadas
summary(datos)
```

```
##      lcavol           lweight           age              lbph
##  Min.   :-1.3471   Min.   :2.375   Min.   :41.00   Min.   :-1.3863
##  1st Qu.: 0.5128   1st Qu.:3.376   1st Qu.:60.00   1st Qu.:-1.3863
##  Median : 1.4469   Median :3.623   Median :65.00   Median : 0.3001
##  Mean   : 1.3500   Mean   :3.629   Mean   :63.87   Mean   : 0.1004
##  3rd Qu.: 2.1270   3rd Qu.:3.876   3rd Qu.:68.00   3rd Qu.: 1.5581
##  Max.   : 3.8210   Max.   :4.780   Max.   :79.00   Max.   : 2.3263
##       svi              lcp              gleason          pgg45
##  Min.   :0.0000   Min.   :-1.3863   Min.   :6.000   Min.   :  0.00
##  1st Qu.:0.0000   1st Qu.:-1.3863   1st Qu.:6.000   1st Qu.:  0.00
##  Median :0.0000   Median :-0.7985   Median :7.000   Median : 15.00
##  Mean   :0.2165   Mean   :-0.1794   Mean   :6.753   Mean   : 24.38
##  3rd Qu.:0.0000   3rd Qu.: 1.1787   3rd Qu.:7.000   3rd Qu.: 40.00
##  Max.   :1.0000   Max.   : 2.9042   Max.   :9.000   Max.   :100.00
##       lpsa
##  Min.   :-0.4308
##  1st Qu.: 1.7317
##  Median : 2.5915
##  Mean   : 2.4784
##  3rd Qu.: 3.0564
##  Max.   : 5.5829
```

```r
dim(datos)
```

```
## [1] 97  9
```

```r
names(datos)
```

```
## [1] "lcavol"  "lweight" "age"     "lbph"    "svi"     "lcp"     "gleason"
## [8] "pgg45"   "lpsa"
```

```r
str(datos)
```

```
## 'data.frame':    97 obs. of  9 variables:
##  $ lcavol : num  -0.58 -0.994 -0.511 -1.204 0.751 ...
##  $ lweight: num  2.77 3.32 2.69 3.28 3.43 ...
##  $ age    : int  50 58 74 58 62 50 64 58 47 63 ...
##  $ lbph   : num  -1.39 -1.39 -1.39 -1.39 -1.39 ...
##  $ svi    : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ lcp    : num  -1.39 -1.39 -1.39 -1.39 -1.39 ...
##  $ gleason: int  6 6 7 6 6 6 6 6 6 6 ...
##  $ pgg45  : int  0 0 20 0 0 0 0 0 0 0 ...
##  $ lpsa   : num  -0.431 -0.163 -0.163 -0.163 0.372 ...
```

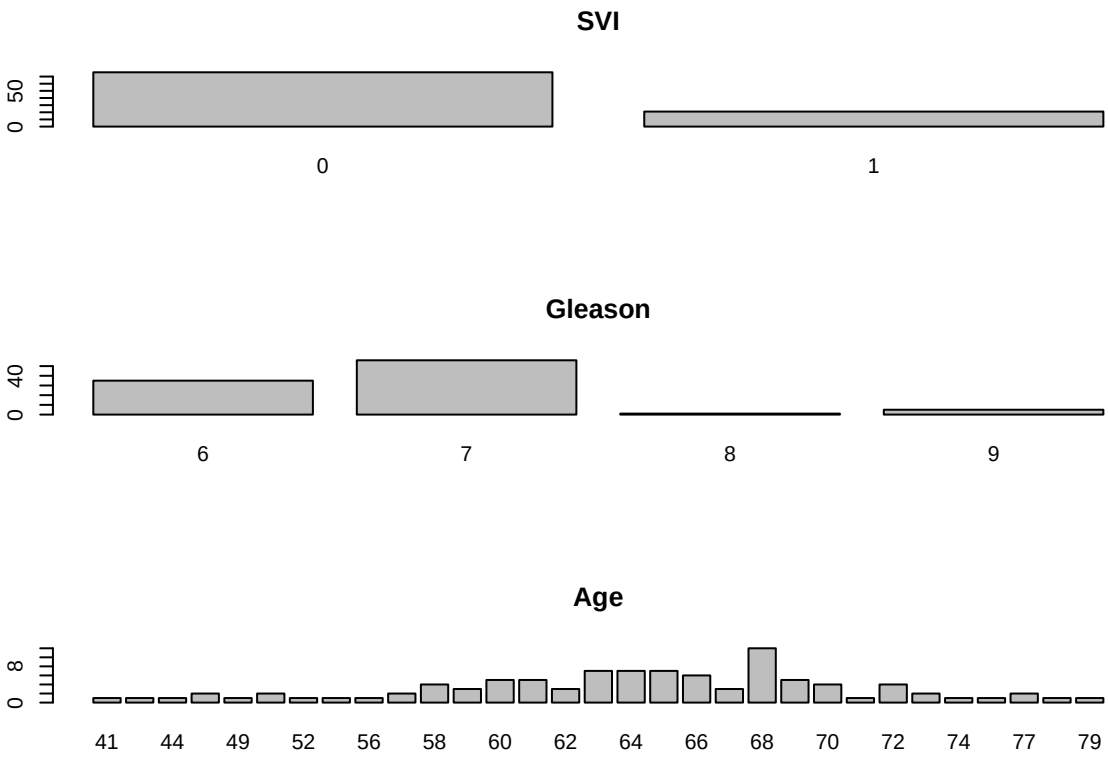
- Hay 10 variables, 9 si quitamos el id del paciente
- Las variables son numéricas
- La variable correspondiente al identificador del paciente es la primera columna
- No hay valores nulos
- Las variables no están ni normalizadas ni estandarizadas
- Hay variables que están en escala logarítmica ya que algunas variables tienen valores negativos a pesar de estar definidas estrictamente positivas, como la concentración en ng/m

2. Análisis de variable categóricas

```r
# Convertimos las variables en factores
datos$svi <- as.factor(datos$svi)
datos$gleason <- as.factor(datos$gleason)
datos$age <- as.factor(datos$age)

# Hacemos attach a los datos
attach(datos)

# Comprobamos que las variables son categóricas
str(datos)
```

```
## 'data.frame':    97 obs. of  9 variables:
##  $ lcavol : num  -0.58 -0.994 -0.511 -1.204 0.751 ...
##  $ lweight: num  2.77 3.32 2.69 3.28 3.43 ...
##  $ age    : Factor w/ 31 levels "41","43","44",..: 6 11 27 11 15 6 17 11 4 16 ...
##  $ lbph   : num  -1.39 -1.39 -1.39 -1.39 -1.39 ...
##  $ svi    : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
##  $ lcp    : num  -1.39 -1.39 -1.39 -1.39 -1.39 ...
##  $ gleason: Factor w/ 4 levels "6","7","8","9": 1 1 2 1 1 1 1 1 1 1 ...
##  $ pgg45  : int  0 0 20 0 0 0 0 0 0 0 ...
##  $ lpsa   : num  -0.431 -0.163 -0.163 -0.163 0.372 ...
```

```r
# Comprobamos la dispersión de sus valores
par(mfrow = c(3, 1))
plot(svi, main = "SVI")
plot(gleason, main = "Gleason")
plot(age, main = "Age")
```



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

3. Análisis de frecuencias

- **¿Qué porcentaje de pacientes con la puntuación de Gleason igual a 7, presenta índice igual svi igual a 0?**

```r
# Seleccionamos los pacientes con la puntuación de Gleason igual a 7 y los que tienen svi igual a 0 den
datos.gleason7 <- datos[datos$gleason == "7", ]
datos.gleason7.svi0 <- datos.gleason7[datos.gleason7$svi == "0", ]

# Vemos los pacientes que hay en datos.gleason_7_0 y en datos
patients.gleason7.svi0 <- nrow(datos.gleason7.svi0)
patients <- nrow(datos)

# Dividimos la cantidad de pacientes filtrados entre el total
porcentaje <- patients.gleason7.svi0 / patients * 100
porcentaje
```

```
## [1] 38.14433
```

Vemos que el porcentaje es del 38.14433%.

- **¿Qué porcentaje de pacientes con índice svi igual a 0 tiene la puntuación de Gleason igual a 7?**

```r
# Seleccionamos los individuos con svi igual a 0 y con gleason igual a 7 dentro de estos
datos.svi0 <- datos[datos$svi == "0", ]
datos.svi0.gleason7 <- datos.svi0[datos.svi0$gleason == "7", ]

# Hacemos el porcentaje
porcentaje <- nrow(datos.svi0.gleason7) / nrow(datos.svi0) * 100
porcentaje
```

```
## [1] 48.68421
```

Vemos que el porcentaje es del 48.68421%.

- **Estas dos variables, ¿son independientes?**

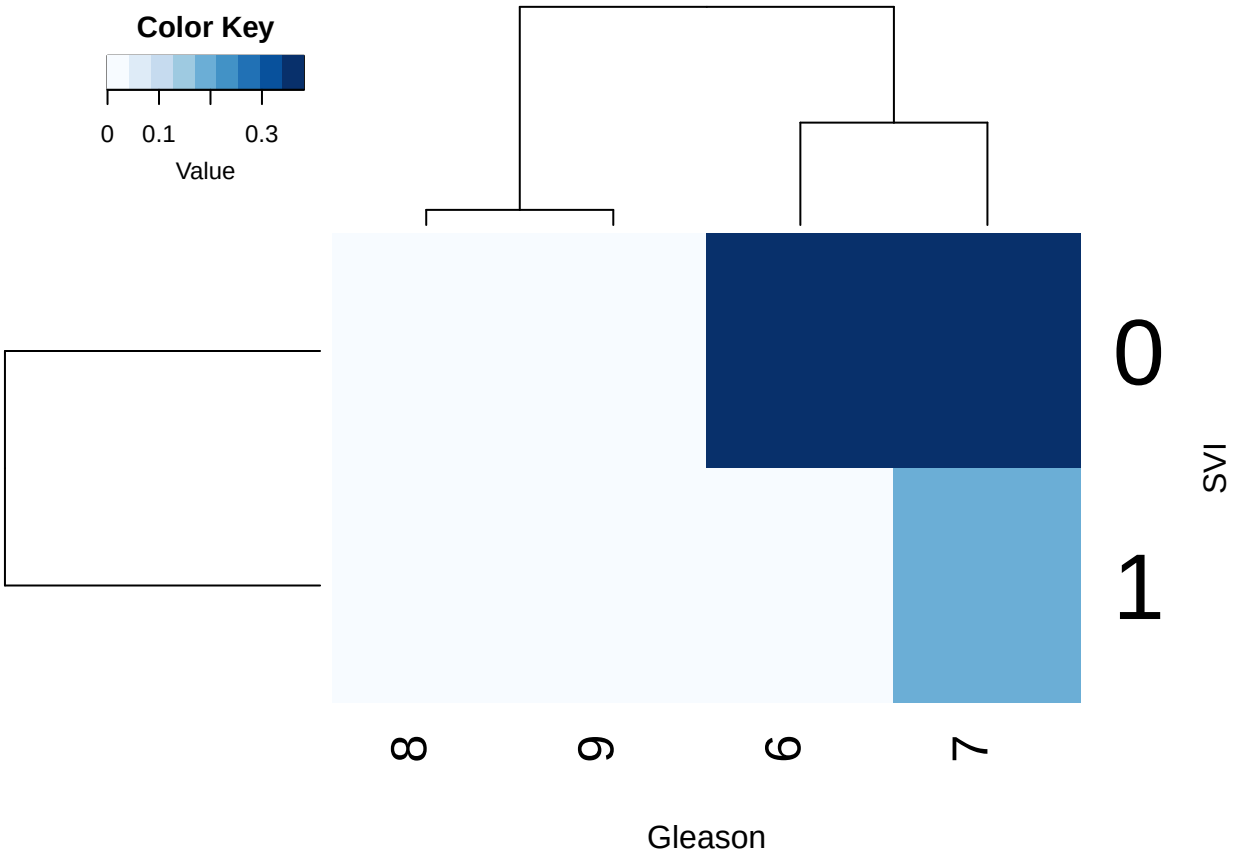
```r
# Creamos una tabla con las dos variables
tabla <- table(svi, gleason)

# Creamos tablas de probabilidad por fila y por columna
addmargins(prop.table(tabla, 1), 2) * 100
```

```
##    gleason
## svi          6          7          8          9        Sum
##   0  46.052632  48.684211   1.315789   3.947368 100.000000
##   1   0.000000  90.476190   0.000000   9.523810 100.000000
```

```r
addmargins(prop.table(tabla, 2), 1) * 100
```

```
##      gleason
## svi           6         7         8         9
##   0   100.00000  66.07143 100.00000  60.00000
##   1     0.00000  33.92857   0.00000  40.00000
##   Sum 100.00000 100.00000 100.00000 100.00000
```

```r
# Realizamos un gráfico de la tabla para visualizar mejor la Independencia
heatmap.2(
    prop.table(tabla),
    xlab = "Gleason", ylab = "SVI",
    density.info = "none",
    col = blues9,
    trace = "none"
)
```



Se puede ver en la gráfica que la mayoría de los casos se acumulan en zonas concretas:

- Cuando SVI es 0, se acumulan en Gleason = 6 y 7.
- Cuando SVI es 1, se acumulan en Gleason = 7.

Por lo tanto, como los datos no se distribuyen por igual en todos los casos, las dos variables son dependientes.

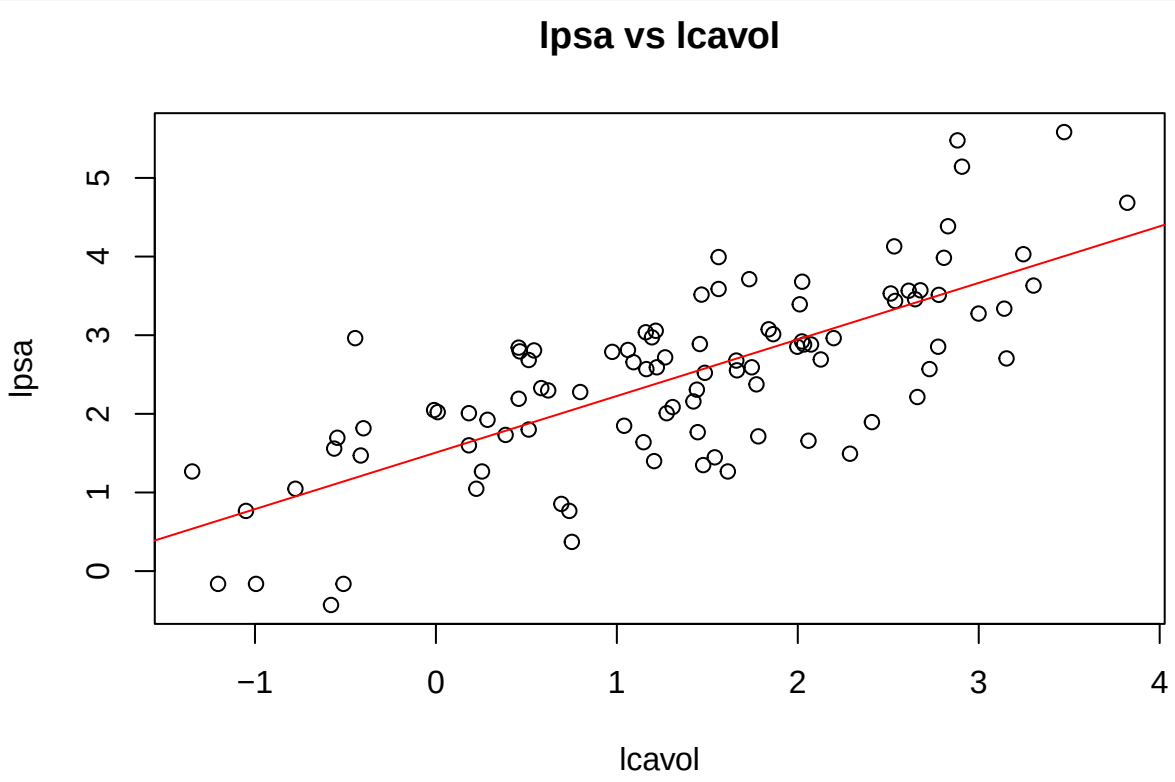
4. Regresión lineal simple

```r
# Realizamos el modelo lineal
recta <- lm(lpsa ~ lcavol)

# Vemos el modelo
recta.summary <- summary(recta)
recta.summary
```

```
##
## Call:
## lm(formula = lpsa ~ lcavol)
```

```
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.67624 -0.41648  0.09859  0.50709  1.89672
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.50730    0.12194   12.36   <2e-16 ***
## lcavol       0.71932    0.06819   10.55   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.7875 on 95 degrees of freedom
## Multiple R-squared:  0.5394, Adjusted R-squared:  0.5346
## F-statistic: 111.3 on 1 and 95 DF,  p-value: < 2.2e-16
```

```r
# Representamos el modelo sobre los datos
plot(lcavol, lpsa, main = "lpsa vs lcavol")
abline(recta, col = "red")
```




```r
# Realizamos el intervalo de confianza al 95%
intervals <- confint(recta, level = 0.95)
intervals
```

```
##                 2.5 %    97.5 %
## (Intercept) 1.2652222 1.7493727
## lcavol      0.5839404 0.8547004
```

```r
# Calculamos el porcentaje de variación relativo de los intervalos de confianza respecto al valor predi
abs(intervals[1, 1] - intervals[1, 2]) / recta.summary$coefficients[1, 1] * 100
```

```
## [1] 32.12044
```

```r
abs(intervals[2, 1] - intervals[2, 2]) / recta.summary$coefficients[2, 1] * 100
```

```
## [1] 37.64109
```

```r
# Calculamos la suma de cuadrados de los residuos (RSE)
r1 <- residuals(recta)
RSE <- sqrt(sum(r1^2) / (dim(datos)[1] - 2))
RSE / mean(lpsa) * 100
```

```
## [1] 31.77468
```

Vamos a analizar el modelo lineal:

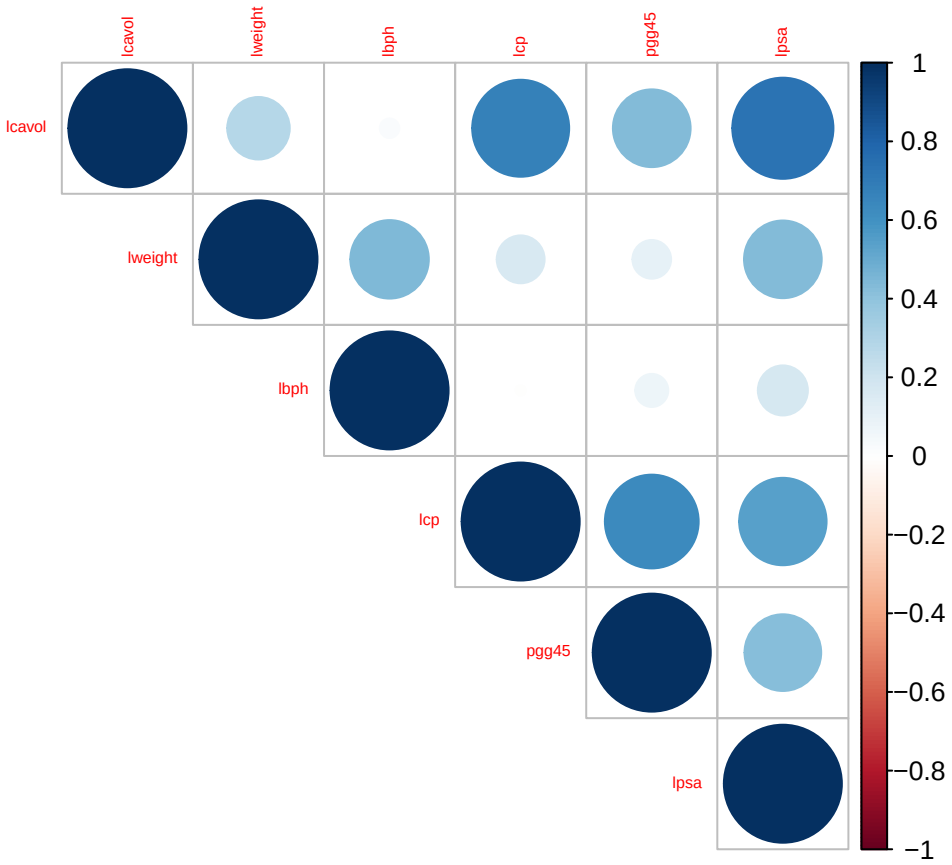
- El t-value es bastante alto (12.36 y 10.55), por lo que los coeficientes están bastante alejados de ser nulos.
- El p-value es bastante bajo (<2e-16), lo que refuerza que los coeficientes no son nulos.
- El coeficiente R^2 no es muy alto, por lo que quizás el modelo lineal no sea el mejor modelo al que los datos se ajusten. Como el valor es 0.5394, el modelo explica el 53.94% de la variabilidad de lpsa respecto a lcavol.
- El RSE es de 0.7875. Si lo dividimos entre la media de lpsa, vemos que tendríamos un error del 31.77%, lo que indica que el modelo no es muy bueno.
- La lontidud de los intervalos de confiana de las variables representan un 32.1% y 37.6% respecto a los valores estimados de los coeficientes. Esto es una variablilidad importante.

Está claro que las variables lpsa y lcavol están relacionadas. Sin embargo, aunque el p-value del ajuste lineal sea bajo, otros factores como el R^2, los residuos y los intervalos de confianza nos indican que los datos están muy dispersos respecto al modelo. En el caso de quedarnos con el modelo, podemos pasar de un modelo lineal a un modelo de potencias, donde cavol es una potencia de psa con exponente β_1.

$$\ln(lpsa) = \beta_0 + \beta_1 \ln(lcavol) \Rightarrow psa = e^{\beta_0 + \beta_1 \ln(lcavol)} = \tilde{\beta_0} cavol^{\beta_1}$$

5. Regresión lineal multiple

```r
# Seleccionamos las columnas numéricas
num_cols <- which(sapply(datos, is.numeric))

# Hacemos un plot de la matriz de correlación
corrplot(cor(datos[, num_cols]), type = "upper", tl.cex = 0.5)
```




```
# Cremos un dataframe con solo los datos numéricos
datos.num <- datos[, c(-3, -7, -8)]

# Realizamos un modelo lineal entre lpsa y las variables numéricas
rectaMul <- lm(lpsa ~ ., data = datos.num)

# Vemos el modelo lineal
summary(rectaMul)
```

```
##
## Call:
## lm(formula = lpsa ~ ., data = datos.num)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.84878 -0.38372 -0.00413  0.45189  1.55468
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.36496    0.69866  -0.522  0.60269
## lcavol       0.54790    0.08637   6.343 8.56e-09 ***
## lweight      0.53036    0.19769   2.683  0.00867 **
## lbph         0.07999    0.05643   1.418  0.15971
## svi1         0.75975    0.24122   3.150  0.00221 **
## lcp         -0.03638    0.08088  -0.450  0.65391
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

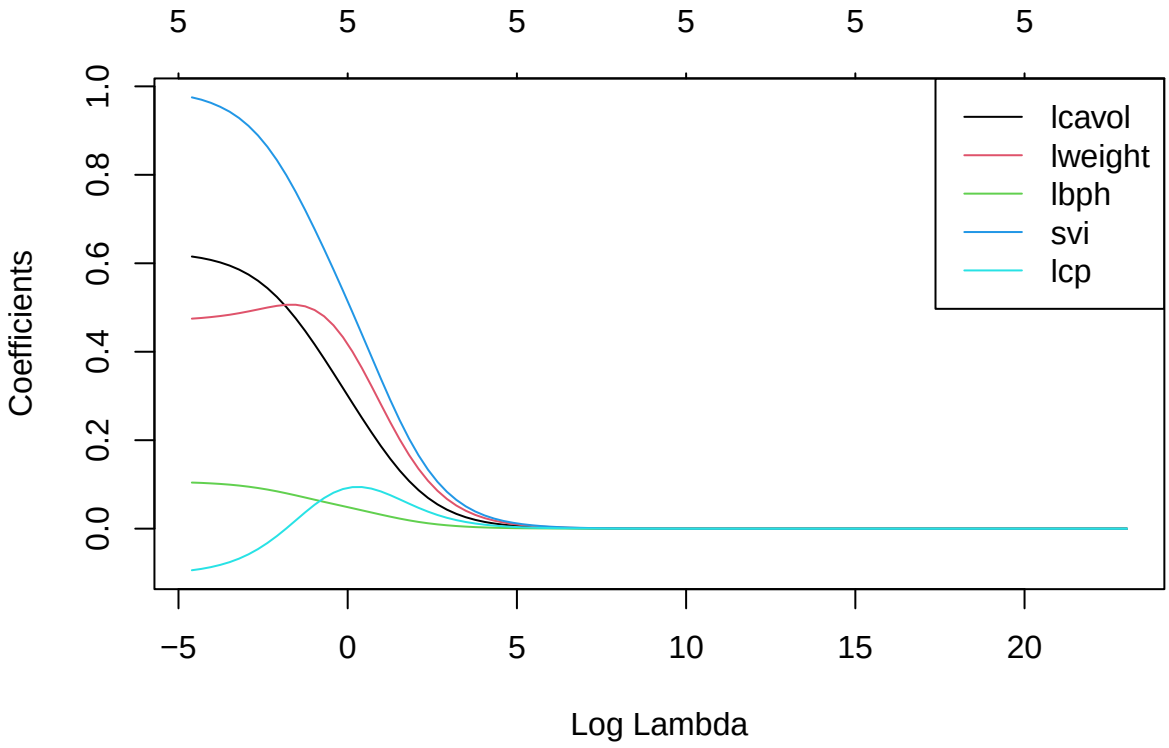
```
##
## Residual standard error: 0.7071 on 91 degrees of freedom
## Multiple R-squared:  0.6443, Adjusted R-squared:  0.6248
## F-statistic: 32.97 on 5 and 91 DF,  p-value: < 2.2e-16
```

Se puede ver en la matriz de correlación como algunas variables parecen tener cierta dependencia. En el caso de lpsa, parece estar relacionada con lcavol, lweight, lcp y en menor medida con pgg45. Si analizamos el p-value de los coeficientes, podemos ver que solo las variables svi1, lweight y lcavol tienen valores bajos, acompañados de t-values relativamente altos. Sin embargo, los valores de R^2 y RSE son mejores que en el modelo lineal simple con lcavol, siendo de 0.6443 y 0.7071 respectivamente. Por lo tanto, podemos determinar que lpsa está relacionada con algunas variables y que el modelo lineal múltiple da mejores resultados que el simple, pero siguen siendo resultados con grandes errores. Además, se podrían eliminar algunas variables con las cuales no parece haber dependencia.

6. Modelo de Ridge y Lasso

Realizamos el modelo Rigde

```r
# Seleccionamos los datos para realizar los modelos
x <- model.matrix(lpsa ~ ., datos.num)[, -1]
y <- datos.num$lpsa

# Seleccionamos la semilla para los números aleatorios
set.seed(1)

# Seleccionamos los conjuntos de entrenamiento y test
train <- sample(seq(1, nrow(x)), nrow(x) / 2) # conjunto de entrenamiento
test <- (-train) # conjunto de testeo

# Guardamos los conjuntos de entrenamiento y test en variables
x.train <- x[train, ]
x.test <- x[test, ]
y.test <- y[test]
y.train <- y[train]

# Hacemos una malla con los valore de lambda
malla <- 10^seq(10, -2, length = 100)
malla.ridge.train <- glmnet(x.train, y.train, alpha = 0, lambda = malla) # regresion ridge sin CV con c

# Representamos los valores de los coeficientes a lo largo del valor lambda
plot(malla.ridge.train, xvar = "lambda")
legend("topright", lty = 1, col = 2:ncol(datos.num)-1, legend = names(datos.num[-ncol(datos.num)]))
```




```
# Realizamos una regresión Ridge con CV
cv.out.ridge.train <- cv.glmnet(x.train, y.train, alpha = 0)

# Seleccionamos el mejor lambda
bestlam.ridge.train <- cv.out.ridge.train$lambda.min
bestlam.ridge.train
```

```
## [1] 0.1997304
```

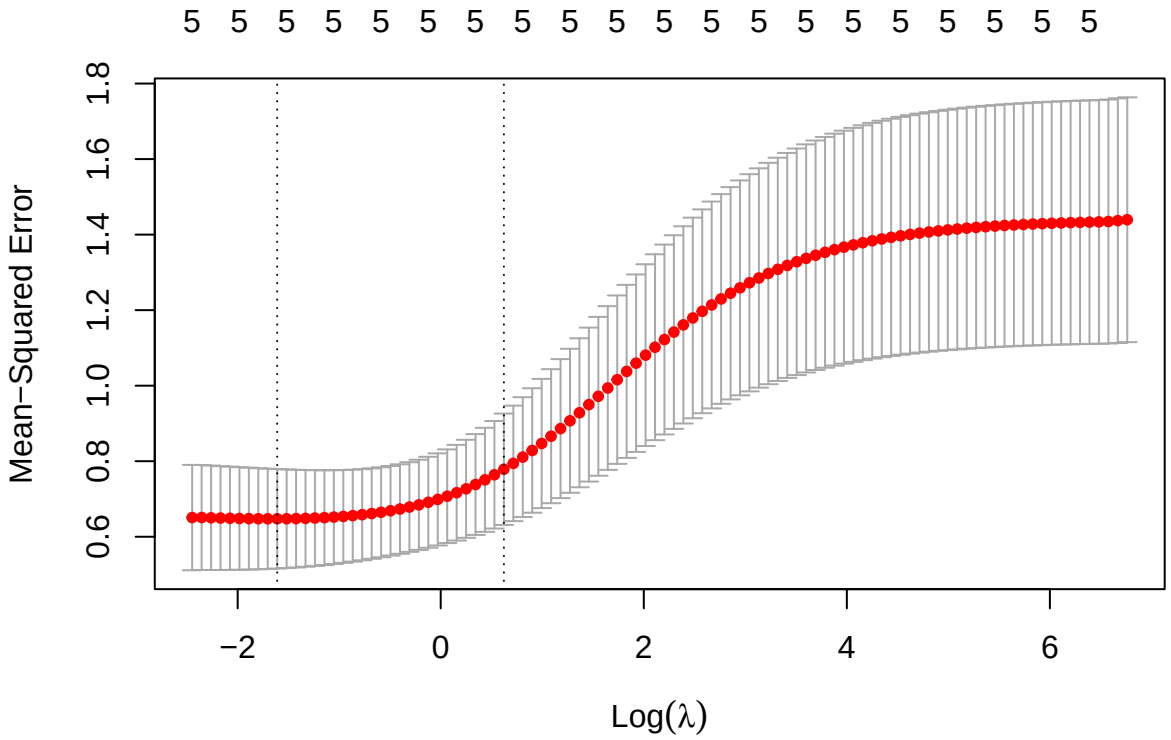
```
# Realizamos la regresión Ridge con CV y el mejor lambda
ridge.train <- glmnet(x.train, y.train, alpha = 0, lambda = bestlam.ridge.train)
coef(ridge.train)[, 1]
```

```
## (Intercept)      lcavol     lweight        lbph        svi1         lcp
## -0.15520758  0.48266384  0.50638792  0.07656639  0.77210598  0.01365333
```

```
# Realizamos una regresión múltiple con los datos de entrenamiento
rectaMul.train <- glmnet(x.train, y.train, alpha = 0, lambda = 0)
coef(rectaMul.train)[, 1]
```

```
## (Intercept)      lcavol     lweight        lbph        svi1         lcp
##  -0.2924278   0.6263806   0.4693546   0.1068825   0.9934064  -0.1042145
```

```
# Realizamos una gráfica para ver el MSE de los ajustes para cada valor de lambda
plot(cv.out.ridge.train)
```




```
# Hacemos una predicción del Ridge para sacar los valores del MSE
ridge.pred <- predict(ridge.train, newx = x.test) # predicion de 'ridge.train' en conjunto de testeo
ridge.MSE <- mean((ridge.pred - y.test)^2) # MSE estimado de Ridge

# Hacemos una predicción el modelo de regresión lineal múltiple para sacar los valores del MSE
rectaMul.pred <- predict(rectaMul.train, newx = x[test, ]) # predicion de 'rectaMul.train' en conjunto
rectMul.MSE <- mean((rectaMul.pred - y.test)^2) # MSE estimado de rectaMul

# Comparamos los valores de los MSE con las medias de los valores de testeo
ridge.MSE / mean(y.test) * 100
```

```
## [1] 19.28006
```

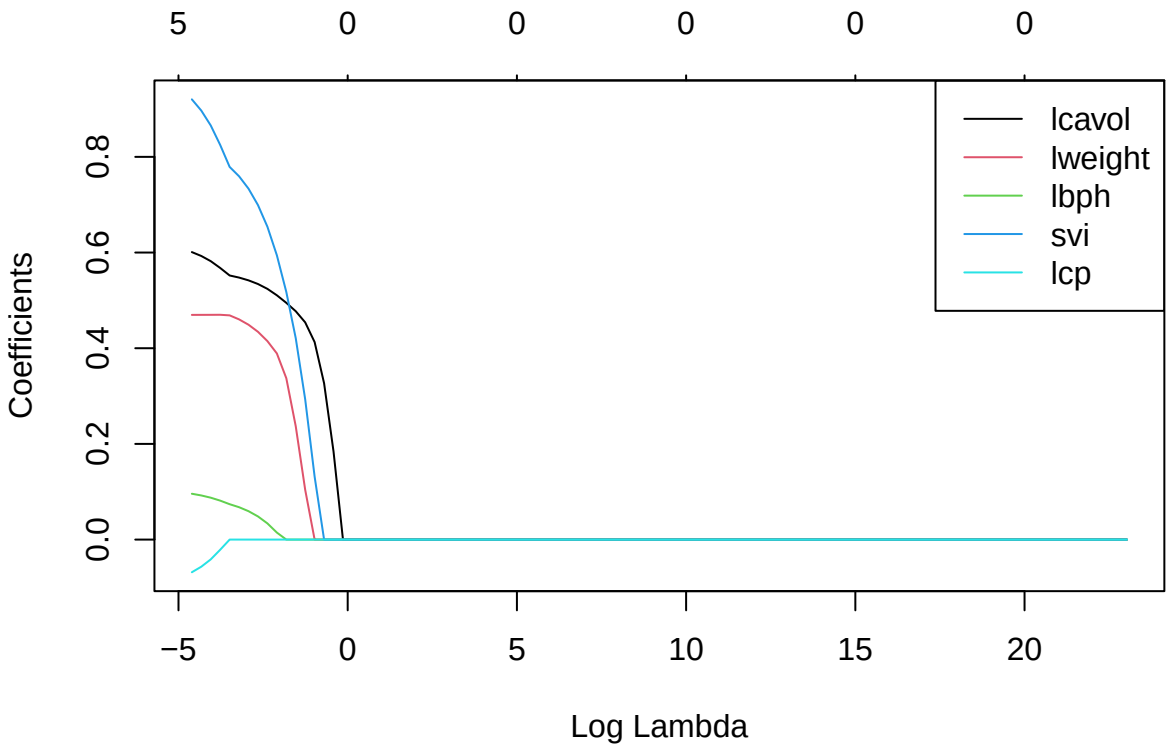
```
rectMul.MSE / mean(y.test) * 100
```

```
## [1] 20.71263
```

Tras realizar la validación cruzada, podemos ver como el mejor λ es $\lambda = 0.1997304$. Al comparar el modelo Ridge con el λ óptimo y el modelo de regresión lineal múltiple, se observa cómo el valor de MSE es menor para el modelo de regresión Ridge. Comparando estos valores con la media de los valores del conjunto de testeo, se observa una mejora de aproximadamente un 1.4% (del 20.7% al 19.3%).

Realizamos el modelo Lasso

```
# Realizamos una regresión LASSO sin CV para el conjunto de entrenamiento
malla.lasso.train <- glmnet(x.train, y.train, alpha = 1, lambda = malla)

# Representamos los valores de los coeficientes a lo largo del valor lambda
plot(malla.lasso.train, xvar = "lambda")
legend("topright", lty = 1, col = 2:ncol(datos.num) - 1, legend = names(datos.num[-ncol(datos.num)]))
```



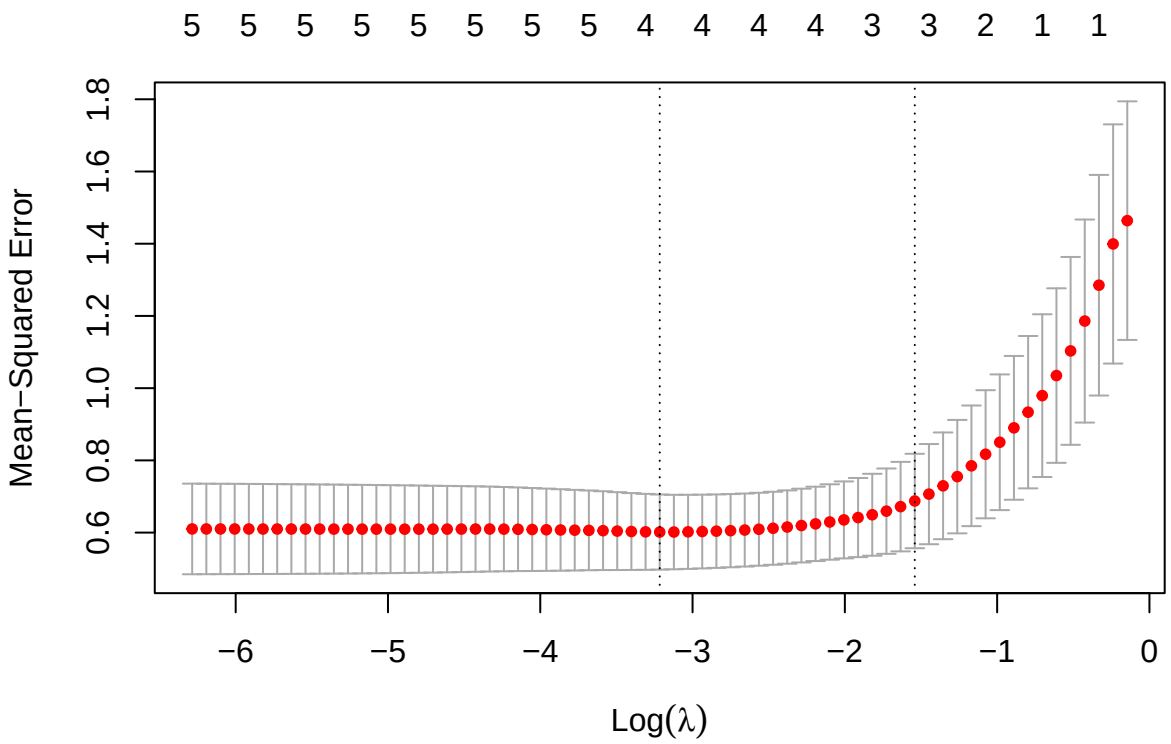

```
# Realizamos una regresión LASSO con CV para el conjunto de entrenamiento
cv.out.lasso.train <- cv.glmnet(x[train, ], y[train], alpha = 1)

# Seleccionamos el mejor lambda (el que minimiza el MSE)
bestlam.lasso.train <- cv.out.lasso.train$lambda.min
bestlam.lasso.train
```

```
## [1] 0.0401305
```

```
# Realizamos una regresión LASSO con el mejor lambda para el conjunto de entrenamiento
lasso.train <- glmnet(x[train, ], y[train], alpha = 1, lambda = bestlam.lasso.train)
coef(lasso.train)[, 1]
```

```
## (Intercept)      lcavol     lweight        lbph        svi1         lcp
## -0.07700425  0.54778095  0.46042266  0.06784849  0.75986294  0.00000000
```

```
# Realizamos una gráfica para ver el MSE de los ajustes para cada valor de lambda
plot(cv.out.lasso.train) # MSE vs log(lambda)
```




```
# Hacemos una predicción del LASSO para sacar los valores del MSE
lasso.pred <- predict(lasso.train, newx = x[test, ]) # predicion de 'lasso.train' en conjunto de testeo
lasso.MSE <- mean((lasso.pred - y.test)^2) # MSE estimado de lasso
lasso.MSE
```

```
## [1] 0.4669899
```

```
lasso.MSE / mean(y.test) * 100
```

```
## [1] 19.54749
```

```
# Hacemos una predicción el modelo de regresión lineal múltiple para sacar los valores del MSE
rectaMul.pred <- predict(rectaMul.train, newx = x[test, ])
rectaMul.MSE <- mean((rectaMul.pred - y.test)^2) # MSE estimado de rectaMul
rectaMul.MSE
```

```
## [1] 0.494825
```

```
rectaMul.MSE / mean(y.test) * 100
```

```
## [1] 20.71263
```

Podemos observar que el mejor λ para realizar el ajuste es $\lambda = 0.04833733$. Al comparar el modelo LASSO con el λ óptimo y el modelo de regresión lineal múltiple, se observa cómo el valor de MSE es menor para el modelo de regresión LASSO. Comparando estos valores con la media de los valores del conjunto de testeo, se observa una mejora de aproximadamente un 1.2% (del 20.7% al 19.5%).

Se puede ver como Ridge ha dado un valor de MSE un poco menor que LASSO, pero son resultados muy similares.

7. LDA

```r
# Realizamos el modelo LDA
lda <- lda(svi ~ lcavol + lcp + lpsa)
lda
```

```
## Call:
## lda(svi ~ lcavol + lcp + lpsa)
##
## Prior probabilities of groups:
##         0         1
## 0.7835052 0.2164948
##
## Group means:
##     lcavol        lcp     lpsa
## 0 1.017892 -0.6715458 2.136592
## 1 2.551959  1.6018579 3.715360
##
## Coefficients of linear discriminants:
##                LD1
## lcavol -0.08659598
## lcp     0.76640382
## lpsa    0.53153340
```

```r
# Realizamos una predicción sobre el resultado de LDA
lda.pred <- predict(lda)

# Realizamos una tabla de confusión con LDA y SVI
predicted <- lda.pred$class
tabla.confusion.lda <- table(predicted, svi) # tabla de confusion
tabla.confusion.lda <- round(addmargins(prop.table(tabla.confusion.lda)) * 100, 2)
tabla.confusion.lda
```

```
##          svi
## predicted      0      1    Sum
##       0    71.13   4.12  75.26
##       1     7.22  17.53  24.74
##       Sum  78.35  21.65 100.00
```

```r
# Valores acertados
tabla.confusion.lda[1,1] / tabla.confusion.lda[3,1] * 100 # Valores acertados de SVI = 0
```

```
## [1] 90.78494
```

```r
tabla.confusion.lda[2,2] / tabla.confusion.lda[3,2] * 100 # Valores acertados de SVI = 1
```

```
## [1] 80.96998
```

```r
# Valores errados
tabla.confusion.lda[1,2] / tabla.confusion.lda[1,3] * 100 # Valores errados de SVI = 0
```

```
## [1] 5.474356
```

```r
tabla.confusion.lda[2,1] / tabla.confusion.lda[2,3] * 100 # Valores errados de SVI = 1
```

```
## [1] 29.18351
```

Se puede ver en los análisis de la tabla de confusión que el modelo LDA acierta:

- Un 90.8% de las veces cuando SVI = 0.
- Un 81.0% de las veces cuando SVI = 1.

Por otro lado, se ve que el modelo LDA falla:

- Un 5.5% de las veces cuando SVI = 0.
- Un 29.2% de las veces cuando SVI = 1.

Por lo tanto, es un buen modelo a la hora de acertar, pero tiene una gran probabilidad de falso positivo en SVI = 1, prácticamente del 30%.

8. Regresión Logística

```
# Realizamos la regresión logística
lr <- glm(svi ~ lcavol + lcp + lpsa, family = binomial)

# Calculamos las probabilidades del modelo
lr.probs <- predict(lr, type = "response")

# Calculamos las predicciones del modelo
lr.pred <- rep(0, 97)
lr.pred[lr.probs > .5] <- 1

# Hacemos las tablas
tabla.confusion.lr <- prop.table(table(lr.pred, svi)) * 100
tabla.confusion.lr <- round(addmargins(tabla.confusion.lr), 2)
tabla.confusion.lr
```

```
##        svi
## lr.pred      0      1    Sum
##     0    75.26   6.19  81.44
##     1     3.09  15.46  18.56
##     Sum  78.35  21.65 100.00
```

```
# Realizamos los análisis de la tabla de confusión
tabla.confusion.lr[1,1] / tabla.confusion.lr[3,1] * 100 # Valores acertados de SVI = 0
```

```
## [1] 96.05616
```

```
tabla.confusion.lr[2,2] / tabla.confusion.lr[3,2] * 100 # Valores acertados de SVI = 1
```

```
## [1] 71.40878
```

```
tabla.confusion.lr[1,2] / tabla.confusion.lr[1,3] * 100 # Valores errados de SVI = 0
```

```
## [1] 7.600688
```

```
tabla.confusion.lr[2,1] / tabla.confusion.lr[2,3] * 100 # Valores errados de SVI = 1
```

```
## [1] 16.64871
```

```
# Probabilidad de svi = 1 con lcabol = 2.8269, lcp = 1.843, lpsa = 3.285
predict(lr, newdata = data.frame(lcavol = 2.8269, lcp = 1.843, lpsa = 3.285), type = "response")
```

```
##         1
## 0.7710017
```

El modelo de regresión logística acierta:

- Un 96.1% de las veces cuando SVI = 0.
- Un 71.4% de las veces cuando SVI = 1.

Por otro lado, el modelo de regresión logística falla:

- Un 7.6% de las veces cuando SVI = 0.
- Un 16.6% de las veces cuando SVI = 1.

Se puede apreciar como el modelo es bueno para acertar en SVI = 0, y relativamente bueno para acertar en SVI = 1. Además, tiene mucho menos probabilidad de falso positivo que el modelo LDA, teniendo una probabilidad del 7% de falso positivo en SVI = 0 y 16.6% en SVI = 1.

La probabilidad de SVI = 1 con lcavol = 2.8269, lcp = 1.843 y lpsa = 3.285 es del 77.1%.

9. PCA-PCR

```
datos.num$svi <- as.numeric(datos$svi)

variables_pca <- datos.num[, c("lcavol", "lweight", "lbph", "lcp", "svi")]

variables_pca
```

```
##          lcavol  lweight        lbph         lcp svi
## 1  -0.579818495 2.769459 -1.38629436 -1.38629436   1
## 2  -0.994252273 3.319626 -1.38629436 -1.38629436   1
## 3  -0.510825624 2.691243 -1.38629436 -1.38629436   1
## 4  -1.203972804 3.282789 -1.38629436 -1.38629436   1
## 5   0.751416089 3.432373 -1.38629436 -1.38629436   1
## 6  -1.049822124 3.228826 -1.38629436 -1.38629436   1
## 7   0.737164066 3.473518  0.61518564 -1.38629436   1
## 8   0.693147181 3.539509  1.53686722 -1.38629436   1
## 9  -0.776528789 3.539509 -1.38629436 -1.38629436   1
## 10  0.223143551 3.244544 -1.38629436 -1.38629436   1
## 11  0.254642218 3.604138 -1.38629436 -1.38629436   1
## 12 -1.347073648 3.598681  1.26694760 -1.38629436   1
## 13  1.613429934 3.022861 -1.38629436 -0.59783700   1
## 14  1.477048724 2.998229 -1.38629436 -1.38629436   1
## 15  1.205970807 3.442019 -1.38629436 -0.43078292   1
## 16  1.541159072 3.061052 -1.38629436 -1.38629436   1
## 17 -0.415515444 3.516013  1.24415459 -0.59783700   1
## 18  2.288486169 3.649359 -1.38629436  0.37156356   1
## 19 -0.562118918 3.267666 -1.38629436 -1.38629436   1
## 20  0.182321557 3.825375  1.65822808 -1.38629436   1
## 21  1.147402453 3.419365 -1.38629436 -1.38629436   1
## 22  2.059238834 3.501043  1.47476301  1.34807315   1
## 23 -0.544727175 3.375880 -0.79850770 -1.38629436   1
## 24  1.781709133 3.451574  0.43825493  1.17865500   1
## 25  0.385262401 3.667400  1.59938758 -1.38629436   1
## 26  1.446918983 3.124565  0.30010459 -1.38629436   1
## 27  0.512823626 3.719651 -1.38629436 -0.79850770   1
## 28 -0.400477567 3.865979  1.81645208 -1.38629436   1
## 29  1.040276712 3.128951  0.22314355  0.04879016   1
## 30  2.409644165 3.375880 -1.38629436  1.61938824   1
## 31  0.285178942 4.090169  1.96290773 -0.79850770   1
## 32  0.182321557 3.804438  1.70474809 -1.38629436   1
## 33  1.275362800 3.037354  1.26694760 -1.38629436   1
## 34  0.009950331 3.267666 -1.38629436 -1.38629436   1
## 35 -0.010050336 3.216874 -1.38629436 -0.79850770   1
## 36  1.308332820 4.119850  2.17133681 -1.38629436   1
```

```
## 37  1.423108334 3.657131 -0.57981850  1.65822808   1
## 38  0.457424847 2.374906 -1.38629436 -1.38629436   1
## 39  2.660958594 4.085136  1.37371558  1.83258146   2
## 40  0.797507196 3.013081  0.93609336 -0.16251893   1
## 41  0.620576488 3.141995 -1.38629436 -1.38629436   1
## 42  1.442201993 3.682610 -1.38629436 -1.38629436   1
## 43  0.582215620 3.865979  1.71379793 -0.43078292   1
## 44  1.771556762 3.896909 -1.38629436  0.81093022   1
## 45  1.486139696 3.409496  1.74919985 -0.43078292   1
## 46  1.663926098 3.392829  0.61518564 -1.38629436   1
## 47  2.727852828 3.995445  1.87946505  2.65675691   2
## 48  1.163150810 4.035125  1.71379793 -0.43078292   1
## 49  1.745715531 3.498022 -1.38629436 -1.38629436   1
## 50  1.220829921 3.568123  1.37371558 -0.79850770   1
## 51  1.091923301 3.993603 -1.38629436 -1.38629436   1
## 52  1.660131027 4.234831  2.07317193 -1.38629436   1
## 53  0.512823626 3.633631  1.49290410  0.04879016   1
## 54  2.127040520 4.121473  1.76644166  1.44691898   1
## 55  3.153590358 3.516013 -1.38629436 -1.38629436   1
## 56  1.266947603 4.280132  2.12226154 -1.38629436   1
## 57  0.974559640 2.865054 -1.38629436  0.50077529   1
## 58  0.463734016 3.764682  1.42310833 -1.38629436   1
## 59  0.542324291 4.178226  0.43825493 -1.38629436   1
## 60  1.061256502 3.851211  1.29472717 -1.38629436   1
## 61  0.457424847 4.524502  2.32630162 -1.38629436   1
## 62  1.997417706 3.719651  1.61938824  1.90954250   2
## 63  2.775708850 3.524889 -1.38629436  1.55814462   1
## 64  2.034705648 3.917011  2.00821403  2.11021320   2
## 65  2.073171929 3.623007 -1.38629436 -1.38629436   1
## 66  1.458615023 3.836221  1.32175584 -0.43078292   1
## 67  2.022871190 3.878466  1.78339122  1.32175584   1
## 68  2.198335072 4.050915  2.30757263 -0.43078292   1
## 69 -0.446287103 4.408547 -1.38629436 -1.38629436   1
## 70  1.193922468 4.780383  2.32630162 -0.79850770   1
## 71  1.864080131 3.593194 -1.38629436  1.32175584   2
## 72  1.160020917 3.341093  1.74919985 -1.38629436   1
## 73  1.214912744 3.825375 -1.38629436  0.22314355   2
## 74  1.838961071 3.236716  0.43825493  1.17865500   2
## 75  2.999226163 3.849083 -1.38629436  1.90954250   2
## 76  3.141130476 3.263849 -0.05129329  2.42036813   2
## 77  2.010894999 4.433789  2.12226154  0.50077529   1
## 78  2.537657215 4.354784  2.32630162 -1.38629436   1
## 79  2.648300197 3.582129 -1.38629436  2.58399755   2
## 80  2.779440197 3.823192 -1.38629436  0.37156356   1
## 81  1.467874348 3.070376  0.55961579  0.22314355   1
## 82  2.513656063 3.473518  0.43825493  2.32727771   1
## 83  2.613006652 3.888754 -0.52763274  0.55961579   2
## 84  2.677590994 3.838376  1.11514159  1.74919985   1
## 85  1.562346305 3.709907  1.69561561  0.81093022   1
## 86  3.302849259 3.518980 -1.38629436  2.32727771   2
## 87  2.024193067 3.731699  1.63899671 -1.38629436   1
## 88  1.731655545 3.369018 -1.38629436  0.30010459   2
## 89  2.807593831 4.718052 -1.38629436  2.46385324   2
## 90  1.562346305 3.695110  0.93609336  0.81093022   2
```

```
## 91  3.246490992 4.101817 -1.38629436 -1.38629436   1
## 92  2.532902848 3.677566  1.34807315 -1.38629436   2
## 93  2.830267834 3.876396 -1.38629436  1.32175584   2
## 94  3.821003607 3.896909 -1.38629436  2.16905370   2
## 95  2.907447359 3.396185 -1.38629436  2.46385324   2
## 96  2.882563575 3.773910  1.55814462  1.55814462   2
## 97  3.471966453 3.974998  0.43825493  2.90416508   2
```

```
pr.out <- prcomp(variables_pca, scale = TRUE)

pr.out
```

```
## Standard deviations (1, .., p=5):
## [1] 1.5341177 1.1861896 0.7258177 0.6677143 0.5165108
##
## Rotation (n x k) = (5 x 5):
##                PC1        PC2         PC3         PC4        PC5
## lcavol  0.55604393 -0.0333170 -0.07960760 -0.68945965  0.4560846
## lweight 0.26655685  0.6328337 -0.69068747  0.15410310 -0.1663492
## lbph    0.06148753  0.7309799  0.66785809  0.01944391  0.1243994
## lcp     0.57510875 -0.1496555  0.25430139 -0.06141123 -0.7605344
## svi     0.53407098 -0.2041646  0.07687536  0.70480368  0.4128280
```

```
summary(pr.out)
```

```
## Importance of components:
##                           PC1    PC2    PC3     PC4     PC5
## Standard deviation     1.5341 1.1862 0.7258 0.66771 0.51651
## Proportion of Variance 0.4707 0.2814 0.1054 0.08917 0.05336
## Cumulative Proportion  0.4707 0.7521 0.8575 0.94664 1.00000
```

```
biplot(pr.out, scale = 0)  #biplot
```