Trabajo Regresión con funciones núcleo

Miguel García Moreno Hugo Montilla Anta Lola Velasco Ballesta

2026-01-23

Table of contents

Variables

Preparamos la tabla cambiándole el nombre a las columnas por comodidad.

```
data <- read.table("auto-mpg.data")
colnames(data) <- c("mpg","cylinders","displacement","horsepower",
                    "weight","acceleration","model year",
                    "origin","car name")
```

Definimos las variables como nos piden. ¡Ojo!: x1 tiene valores `NA`, se crea una máscara para seleccionar esos valores y posteriormente poder redefinir x1 sin dichos elementos.

```
y <- data$mpg
x1 <- as.numeric(data$horsepower)
```

```
Warning: NAs introducidos por coerción
```

```
x2 <- data$weight

masc=is.na(x1)
x1<- x1[!masc]
```

Una vez preparadas las variables podemos pasar a realizar los ejercicios.

Tarea 1

> **Ejercicio 1**
>
> Determina el parámetro ventana para la estimación de la función de densidad de y, utilizando núcleo gaussiano, con los métodos: validación cruzada y plug-in.

Para determinar el valor de las ventanas usaremos las funciones siguientes, las cuales usan el núcleo gaussiano por defecto.

```
hvc_y=bw.ucv(y) #  Validación cruzada
hvc_y
```

```
[1] 1.797839
```

```
hpi_y=bw.SJ(y)  #  Plug-in
hpi_y
```

```
[1] 1.94071
```

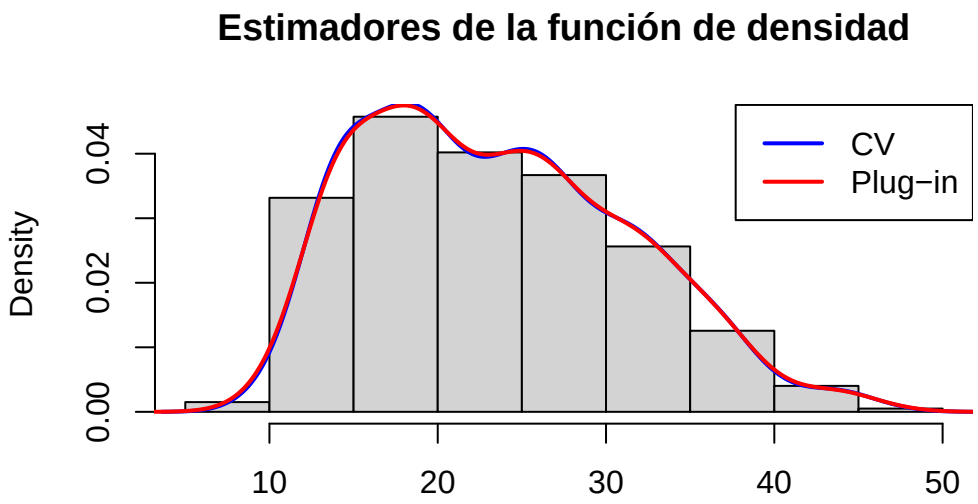
> **Ejercicio 2**
>
> Representa los estimadores núcleo asociados en un mismo gráfico. Comenta brevemente el resultado.

Definimos los distintos estimadores determinando las ventanas que deben ser usadas.

```
d1_y=density(y, bw = hvc_y) # Validación cruzada
d2_y=density(y, bw = hpi_y) # Plug-in
```

Representamos los estimadores sobre el histograma de frecuencia de la variable y.

```
hist(y,freq=FALSE,main='Estimadores de la función de densidad', xlab='')
lines(d1_y, col='blue', lwd= 2 )
lines(d2_y, col = 'red',lwd= 2)
legend("topright", legend = c("CV", "Plug-in"), col = c("blue","red"),lwd = 2)
```



Vemos que los dos tipos de estimaciones dan resultados muy parecidos. Comparandolo con el histograma de frecuencias real, comprobamos que parece ser un buen ajuste.

> **Ejercicio 3**
>
> Repite los dos apartados anteriores para x1 y x2.

x1

```
#x1
hvc_x1=bw.ucv(x1) #  Validación cruzada
```

```
Warning in bw.ucv(x1): minimum occurred at one end of the range
```

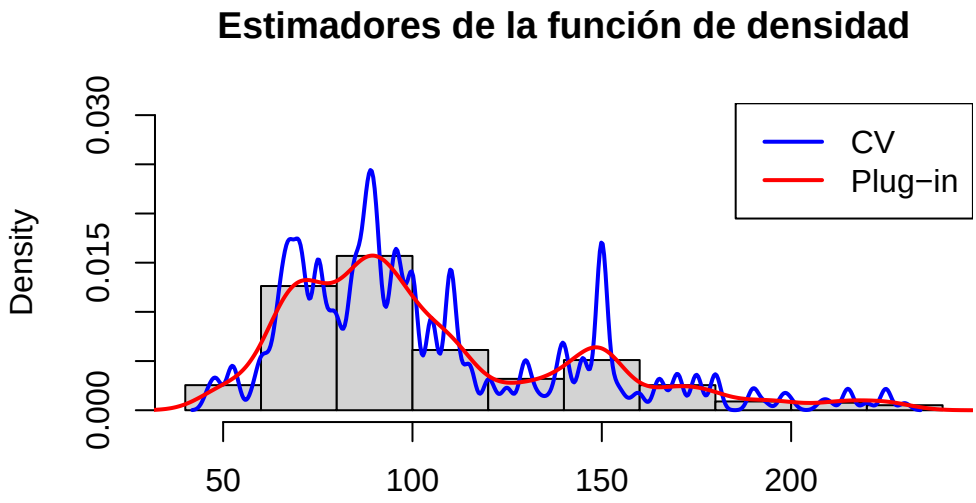
```
hvc_x1
```

```
[1] 1.387066
```

```r
hpi_x1=bw.SJ(x1)  #  Plug-in
hpi_x1
```

```
[1] 5.974012
```

```r
d1_x1=density(x1, bw = hvc_x1) # Validación cruzada
d2_x1=density(x1, bw = hpi_x1) # Plug-in

hist(x1,freq=FALSE,main='Estimadores de la función de densidad', xlab='', ylim=c(0,0.03))
lines(d1_x1, col='blue', lwd= 2 )
lines(d2_x1, col = 'red',lwd= 2)
legend("topright", legend = c("CV", "Plug-in"), col = c("blue","red"),lwd = 2)
```



En este otro observamos diferencias entre las aproximaciones. La estimacion del LSCV es mas rugosa, tiene más pliegues, y en cambio la del plug in es muy suave. Esto se debe a que la ventana del plug in es notablemente mayor que la del LCSV. Es distinto al ejemplo anterior, porque en el anterior ejercicio las ventanas eran muy parecidas.

x2

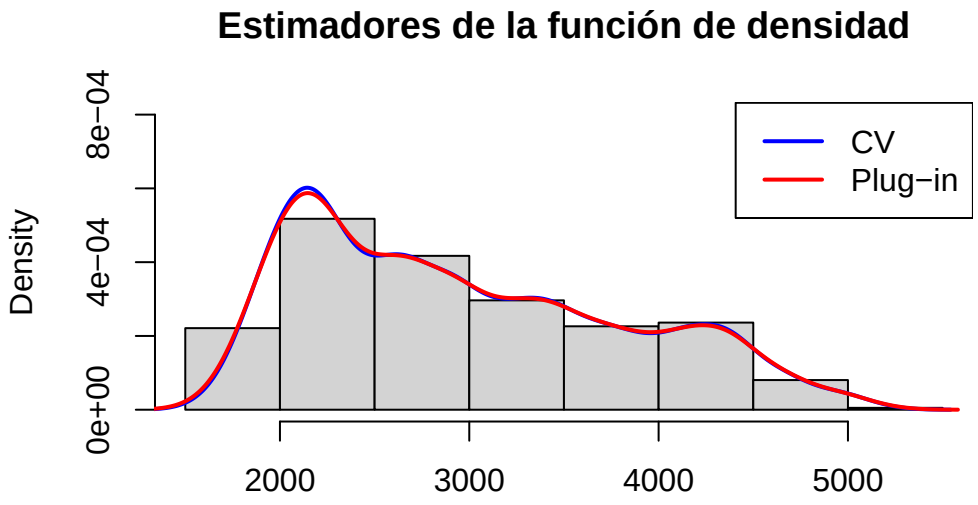
```r
#x2
hvc_x2=bw.ucv(x2) #  Validación cruzada
hvc_x2
```

```
[1] 134.0282
```

```
hpi_x2=bw.SJ(x2)  #  Plug-in
hpi_x2
```

```
[1] 146.9706
```

```
d1_x2=density(x2, bw = hvc_x2) # Validación cruzada
d2_x2=density(x2, bw = hpi_x2) # Plug-in

hist(x2,freq=FALSE,main='Estimadores de la función de densidad', xlab='', ylim=c(0,0.0008))
lines(d1_x2, col='blue', lwd= 2 )
lines(d2_x2, col = 'red',lwd= 2)
legend("topright", legend = c("CV", "Plug-in"), col = c("blue","red"),lwd = 2)
```



En este gráfico observamos un comportamiento muy parecido al del primer apartado.

Tarea 2

> **Ejercicio 4**
>
> Estima la función de densidad conjunta de (y, x1) utilizando un estimador núcleo, con núcleo gaussiano y ventana calculada con método plug-in. Escribe la matriz de ventanas y representa el estimador obtenido.

¡Ojo!: Antes quitamos en x1 los elementos `NA`. Por tanto, debemos quitarle a y los correspondientes.

```
y_masc = y[!masc]

length(x1)
```

```
[1] 392
```

```
length(y_masc)
```

```
[1] 392
```

Para la función de densidad conjunta usaremos la librería `ks`.

```
library(ks)
```

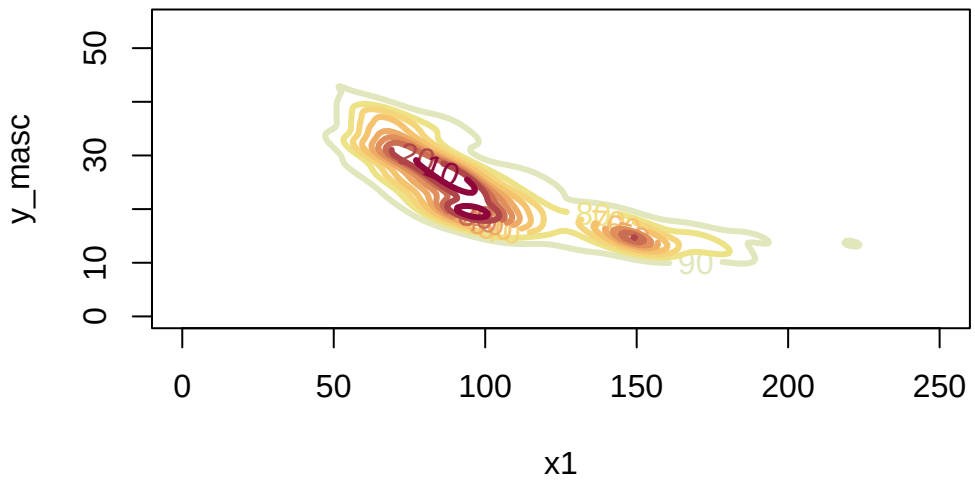
Calculamos la matriz de ventanas.

```
mat = cbind(x1,y_masc)
Hp = Hpi(mat)
Hp
```

```
          [,1]       [,2]
[1,]  66.10216 -10.980567
[2,] -10.98057   3.695583
```

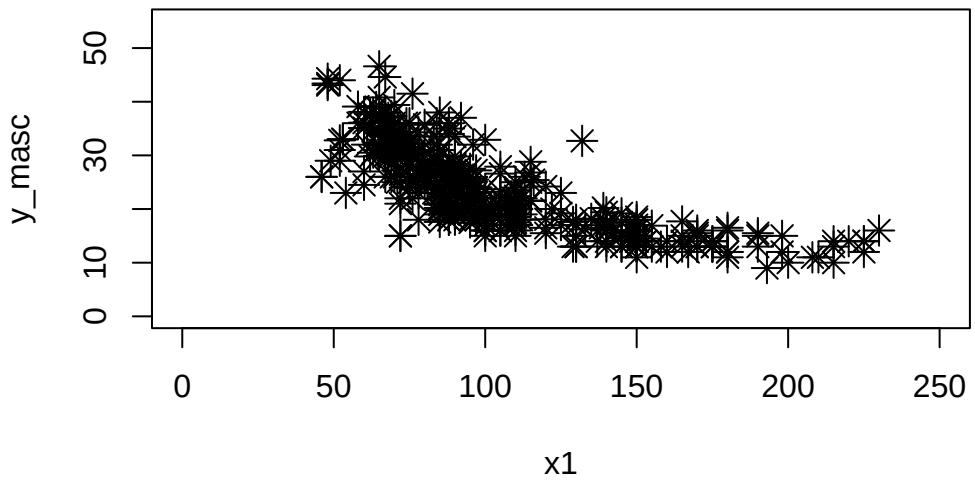
Expresamos la función de densidad conjunta usando `kde`, si no se especifica la matriz de ventanas ni el núcleo usa el método plug-in y el núcleo gaussiano por defecto. Representamos los contornos.

```r
fhat.pi = kde(mat,H=Hp)

v= 1:9*10
plot(fhat.pi, cont=v, lwd=3,ylim=c(0,55),xlim=c(0,250)) # añadimos contornos
```



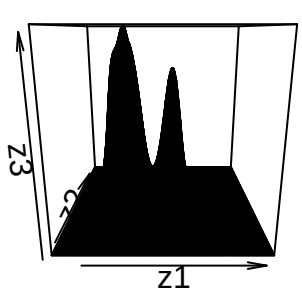
Comparamos con la distribución.

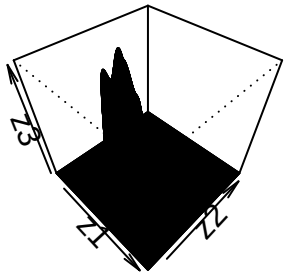
```r
plot(x1,y_masc, pch=8, cex=1.5,ylim=c(0,55),xlim=c(0,250))
```



Comprobamos que se asemejan mucho. Parece una buena estimación.

```
# Representación en 3D

z1 = fhat.pi$eval.points[[1]]

z2 = fhat.pi$eval.points[[2]]

z3 = fhat.pi$estimate

layout(matrix(1:2, ncol=2))
persp(z1,z2,z3,col='gold')
persp(z1,z2,z3,col='gold',theta=45,phi=45) # Mismo gráfico pero rotado
```






```
layout(1) #para que vuelva a verse una gráfica
```

> **Ejercicio 5**
>
> Da una estimación de la función de densidad conjunta en (y, x1) = (20, 150).

```
predict(fhat.pi,x=c(150,20))
```

```
[1] 8.706612e-05
```

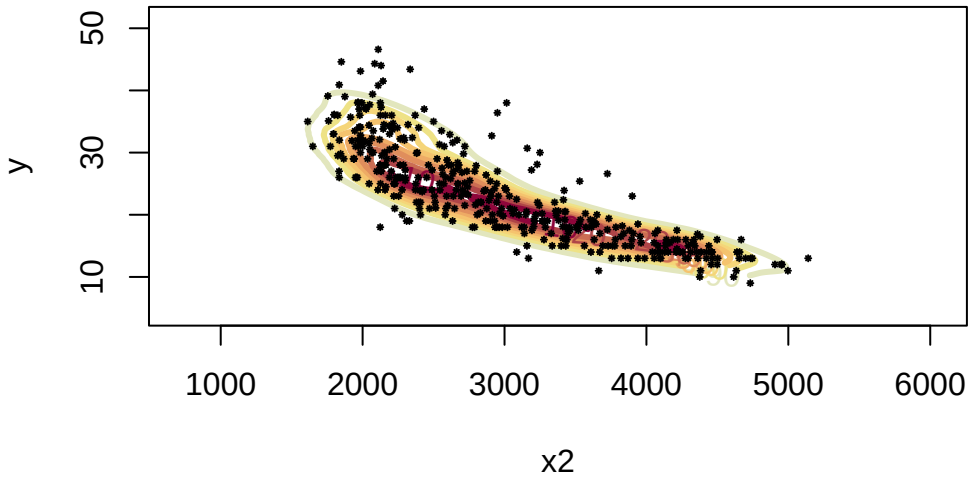
> **Ejercicio 6**
>
> Repite los dos apartados anteriores para (y, x2) y (y, x2) = (25, 3000).

Nota: en este caso usamos la variable y original.

```
mat = cbind(x2, y)

# Ventana
Hp = Hpi(mat)
Hp
```

```
           [,1]        [,2]
[1,] 59829.8738 -387.091432
[2,]  -387.0914    4.256489
```

```
fhat.pi = kde(mat,H=Hp)

v= 1:9*10
plot(fhat.pi, cont=v, lwd=3)
points(x2, y,pch=8, cex=0.3)
```




```
# Representación en 3D

z1 = fhat.pi$eval.points[[1]]

z2 = fhat.pi$eval.points[[2]]

z3 = fhat.pi$estimate

layout(matrix(1:2, ncol=2))
persp(z1,z2,z3,col='gold')
persp(z1,z2,z3,col='gold',theta=45,phi=45) # Mismo grafico pero rotado
```


```
layout(1)
```

Hacemos la predicción de la densidad conjunta dado los valores del enunciado.

```
predict(fhat.pi,x=c(3000,25))
```

```
[1] 1.92177e-05
```

Tarea 3

> **Ejercicio 7**
>
> Estima la curva de regresión de y sobre x1 utilizando un estimador local lineal (p = 1), con núcleo gaussiano, y ventana calculada con los métodos: validación cruzada y plug-in.

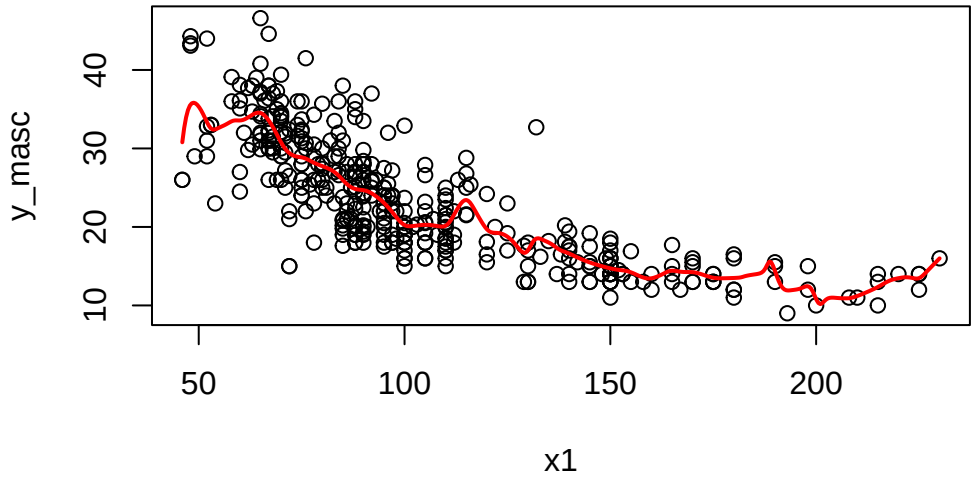
Usamos locpoly del paquete `KernSmooth`, que toma por defecto el núcleo gaussiano y la ventana obtenida por el método 'plug-in'.

```
library(KernSmooth)
```

```
h_pi = dpill(x1,y_masc)
h_pi #ventana plug-in
```

```
[1] 2.504173
```

```
m_pi = locpoly(x1, y_masc,bandwidth=h_pi)

plot(x1, y_masc)
lines(m_pi, col='red', lwd=2)
```



Usamos npregbw del paquete **np** para calcular la ventana mediante validación cruzada.

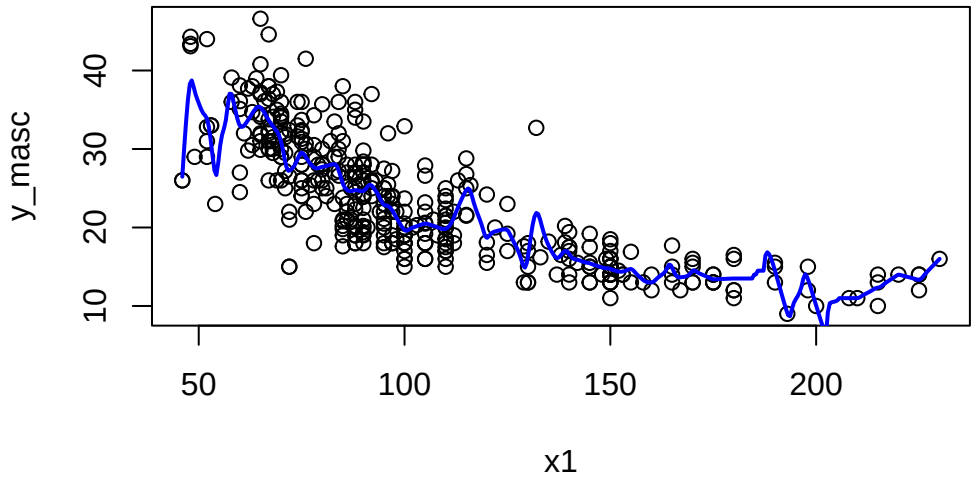
```
library(np)
```

```
bw_vc = npregbw(y_masc~x1, regtype="ll") #hay que indicar el grado
```

```
Multistart 1 of 1 |
Multistart 1 of 1 |
Multistart 1 of 1 |
Multistart 1 of 1 /
Multistart 1 of 1 |
```

```
Multistart 1 of 1 |
```

```
h_vc = bw_vc$bw #ventana validación cruzada

m_vc = locpoly(x1,y_masc, bandwidth = h_vc)

plot(x1, y_masc)
lines(m_vc, col="blue", lwd=2,type="l")
```



Viendo las gráficas podemos ver que el primer método (plug in) tiene una ventana más grande, dibujando una función de densidad estimada más suave que la del segundo método (validación cruzada) que vuelve a tener una ventana más pequeña.

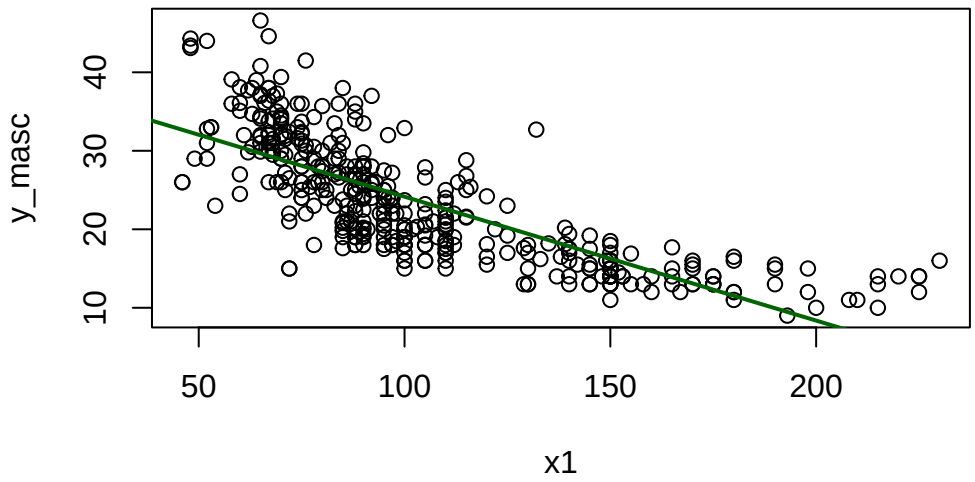
> **Ejercicio 8**
>
> Calcula la línea de regresión y escribe su expresión.

```
lr = lm(y_masc~x1)
lr
```

```
Call:
```

```
lm(formula = y_masc ~ x1)

Coefficients:
(Intercept)           x1
    39.9359      -0.1578
```

```
plot(x1,y_masc)
abline(lr, col="darkgreen", lwd=2)
```

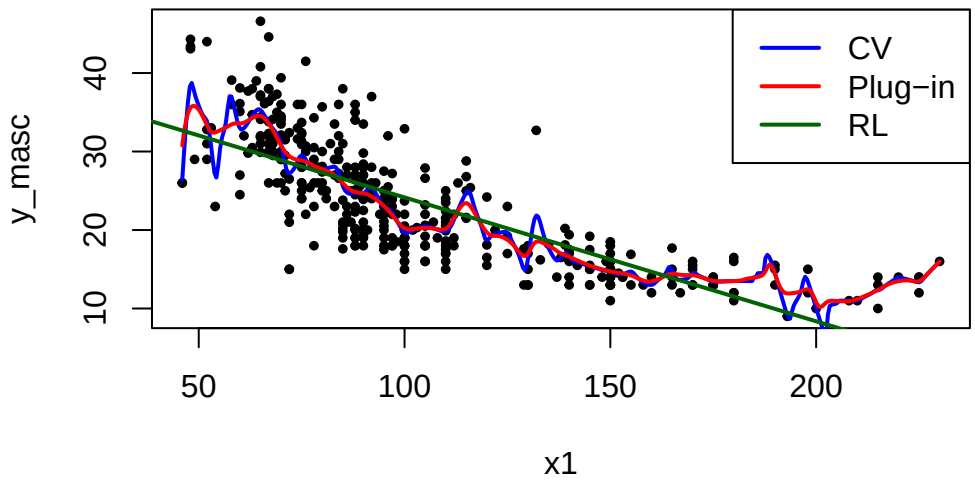


Su expresión sería:

$y\_masc = 39.9359 - 0.1578x_1$

> **Ejercicio 9**
>
> Representa los tres estimadores en un mismo gráfico. Comenta brevemente el resultado.

```
plot(x1,y_masc,pch=20,cex=0.8)
lines(m_vc,col="blue",lwd=2)
lines(m_pi,col="red",lwd=2)
abline(lr,col="darkgreen",lwd=2)
legend("topright", legend = c("CV", "Plug-in","RL"), col = c("blue","red","darkgreen"),
       lwd = 2)
```



Plug in es una estimación más suave de nuevo que la estimación CV. En este probablemente sería buena idea usar el método plug in porque parece adaptarse mejor a la varianza de los datos.

> **Ejercicio 10**
>
> Estima la curva de regresión en x1 = 110 con los tres estimadores.

La función locpoly no tiene la función predict por lo que tenemos que usar approx.

```
pred_pi <- approx(x = m_pi$x,y = m_pi$y,
                  xout = 110)$y
pred_cv <- approx(x = m_vc$x,y = m_vc$y,
                  xout = 110)$y
pred_reg <- predict(lr,newdata=data.frame(x1=110))

predicciones <- c(pred_pi,pred_cv,pred_reg)
names(predicciones) <- c("Plug in","CrossValidation","RL")
predicciones
```

```
     Plug in CrossValidation             RL
    20.16253        19.83644        22.57294
```

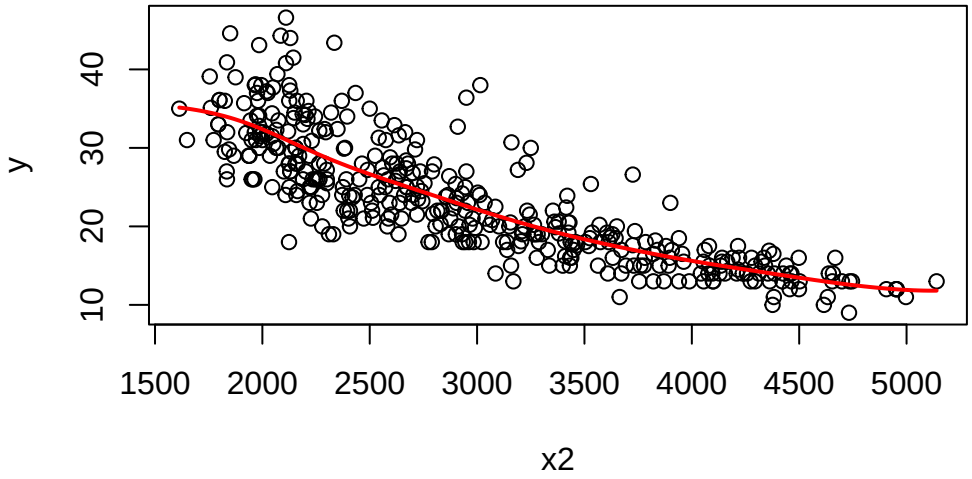
Ejercicio 11

Repite los cuatro apartados anteriores para y sobre x2 y x2 = 3000.

```
h_pi = dpill(x2,y)
h_pi #ventana plug-in
```

```
[1] 262.4867
```

```
m_pi = locpoly(x2, y,bandwidth=h_pi)

plot(x2, y)
lines(m_pi, col='red', lwd=2)
```

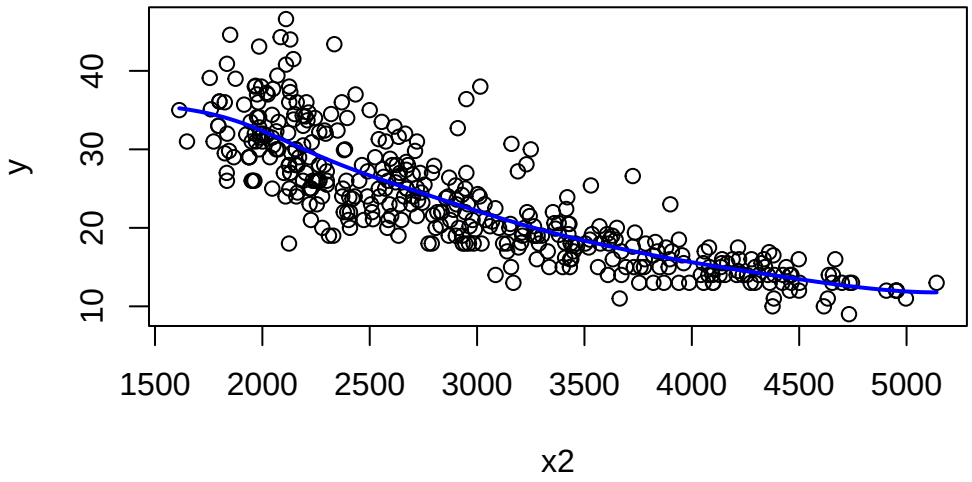



```
bw_vc = npregbw(y~x2, regtype="ll") #hay que indicar el grado
```

```
Multistart 1 of 1 |
Multistart 1 of 1 |
Multistart 1 of 1 |
Multistart 1 of 1 /
```

```
Multistart 1 of 1 |
Multistart 1 of 1 |
```

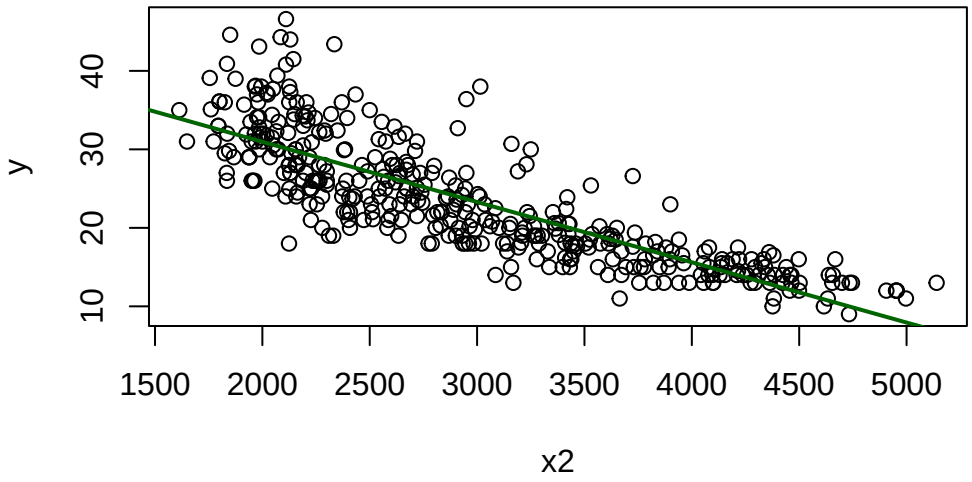
```
h_vc = bw_vc$bw #ventana validación cruzada

m_vc = locpoly(x2,y, bandwidth = h_vc)

plot(x2, y)
lines(m_vc, col="blue", lwd=2)
```




```
lr = lm(y~x2)
lr
```

```
Call:
lm(formula = y ~ x2)

Coefficients:
(Intercept)           x2
  46.317364    -0.007677
```

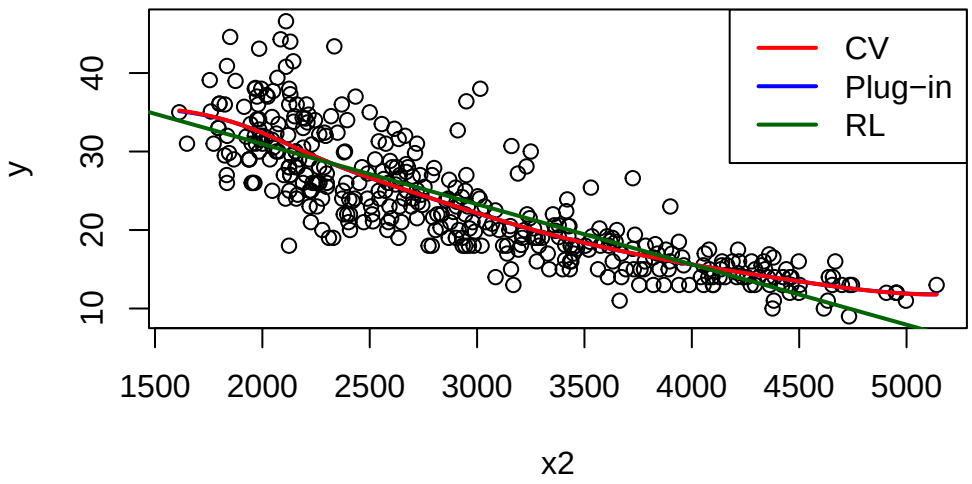
```
plot(x2, y)
abline(lr, col="darkgreen", lwd=2)
```



Su expresión sería:

$y_{masc} = 46.317364 - 0.007677x_2$

```
plot(x2,y)
lines(m_pi,col='blue',lwd=2)
lines(m_vc,col='red',lwd=2)
abline(lr,col='darkgreen',lwd=2)
legend("topright", legend = c("CV", "Plug-in","RL"), col = c("red","blue","darkgreen"),
       lwd = 2)
```



En este caso vemos que CV y plug in han encontrado una ventana similar, concluyendo en estimaciones muy parecidas.

```
pred_pi <- approx(x = m_pi$x,y = m_pi$y,
                          xout = 3000)$y
pred_cv <- approx(x = m_vc$x,y = m_vc$y,
                          xout = 3000)$y
pred_reg <- predict(lr,newdata=data.frame(x2=3000))

predicciones <- c(pred_pi,pred_cv,pred_reg)
names(predicciones) <- c("Plug in","CrossValidation","RL")

predicciones
```

```
     Plug in CrossValidation             RL
    22.14329        22.15018        23.28753
```