Introduccción al aprendizaje estadístico

Ilustración regresión lineal: airquality

#### Juan David Ospina Arango

II Congreso Colombiano de Estadística

#### Semestre 2019-02

Sys.setlocale("LC\_TIME","Spanish")

## Warning in Sys.setlocale("LC\_TIME", "Spanish"): OS reports request to set  
## locale to "Spanish" cannot be honored

## [1] ""

Ilustración de la regresión lineal con el conjunto de datos airquality

Carga de los datos

Los datos están incluidos en la instalación de R. El comando *data()* pone el conjunto de datos en la memoria:

data("airquality")

El comando *summary()* muestra algunas medidas descriptivas de las variables:

summary(airquality)

## Ozone Solar.R Wind Temp   
## Min. : 1.00 Min. : 7.0 Min. : 1.700 Min. :56.00   
## 1st Qu.: 18.00 1st Qu.:115.8 1st Qu.: 7.400 1st Qu.:72.00   
## Median : 31.50 Median :205.0 Median : 9.700 Median :79.00   
## Mean : 42.13 Mean :185.9 Mean : 9.958 Mean :77.88   
## 3rd Qu.: 63.25 3rd Qu.:258.8 3rd Qu.:11.500 3rd Qu.:85.00   
## Max. :168.00 Max. :334.0 Max. :20.700 Max. :97.00   
## NA's :37 NA's :7   
## Month Day   
## Min. :5.000 Min. : 1.0   
## 1st Qu.:6.000 1st Qu.: 8.0   
## Median :7.000 Median :16.0   
## Mean :6.993 Mean :15.8   
## 3rd Qu.:8.000 3rd Qu.:23.0   
## Max. :9.000 Max. :31.0   
##

El conjunto de datos tiene un tamaño de:

dim(airquality)

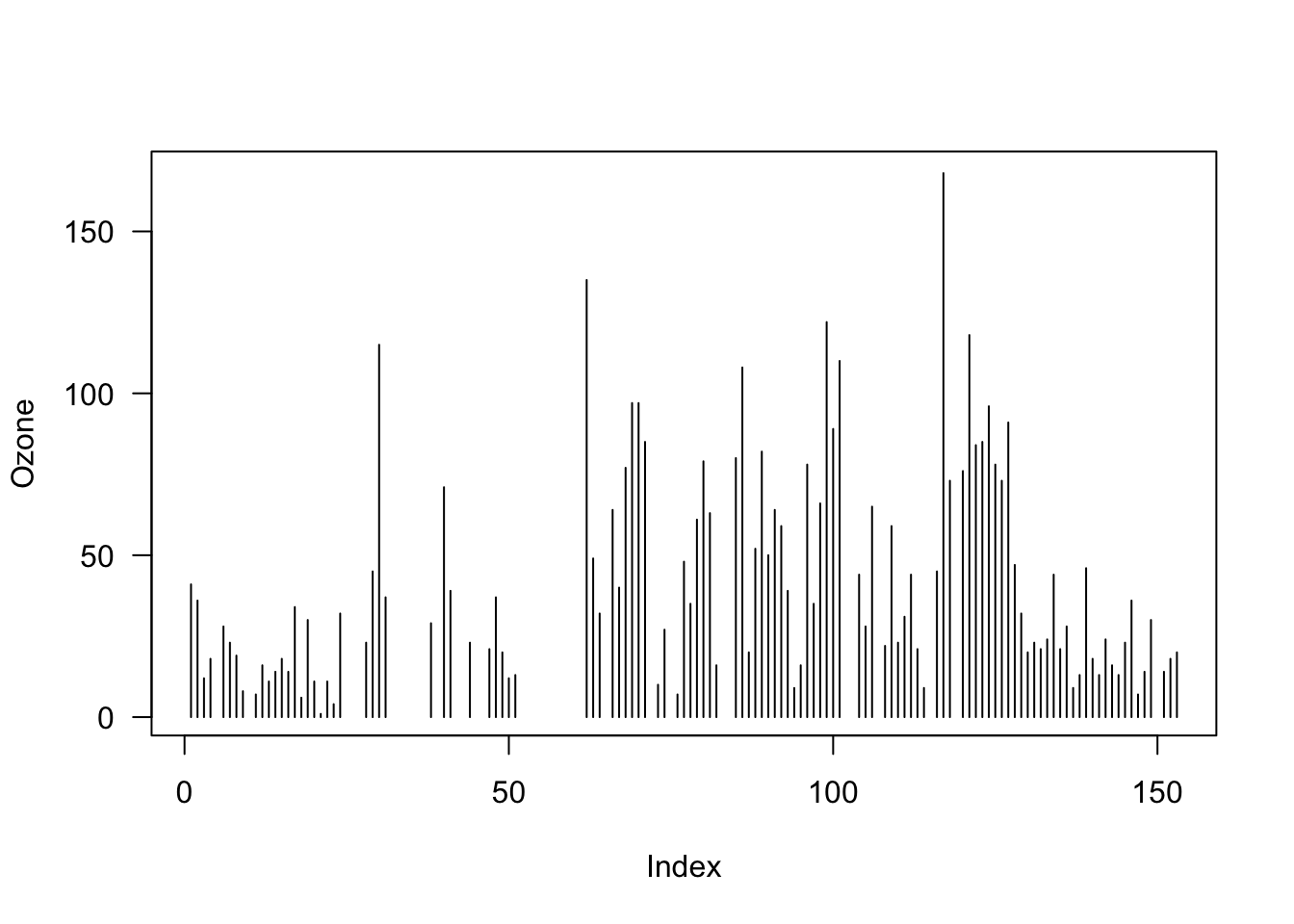
## [1] 153 6

Se tienen 153 observaciones, pero la variable respuesta tiene 37 valores faltantes. El comando *lm()*, que ejecuta la regresión lineal, excluirá todos los registros con valores faltantes. El analista deberá decidir si imputa los datos faltantes o si continua con un conjunto de datos de tamaño reducido.

Análisis exploratorio

Primero graficaremos la serie de datos:

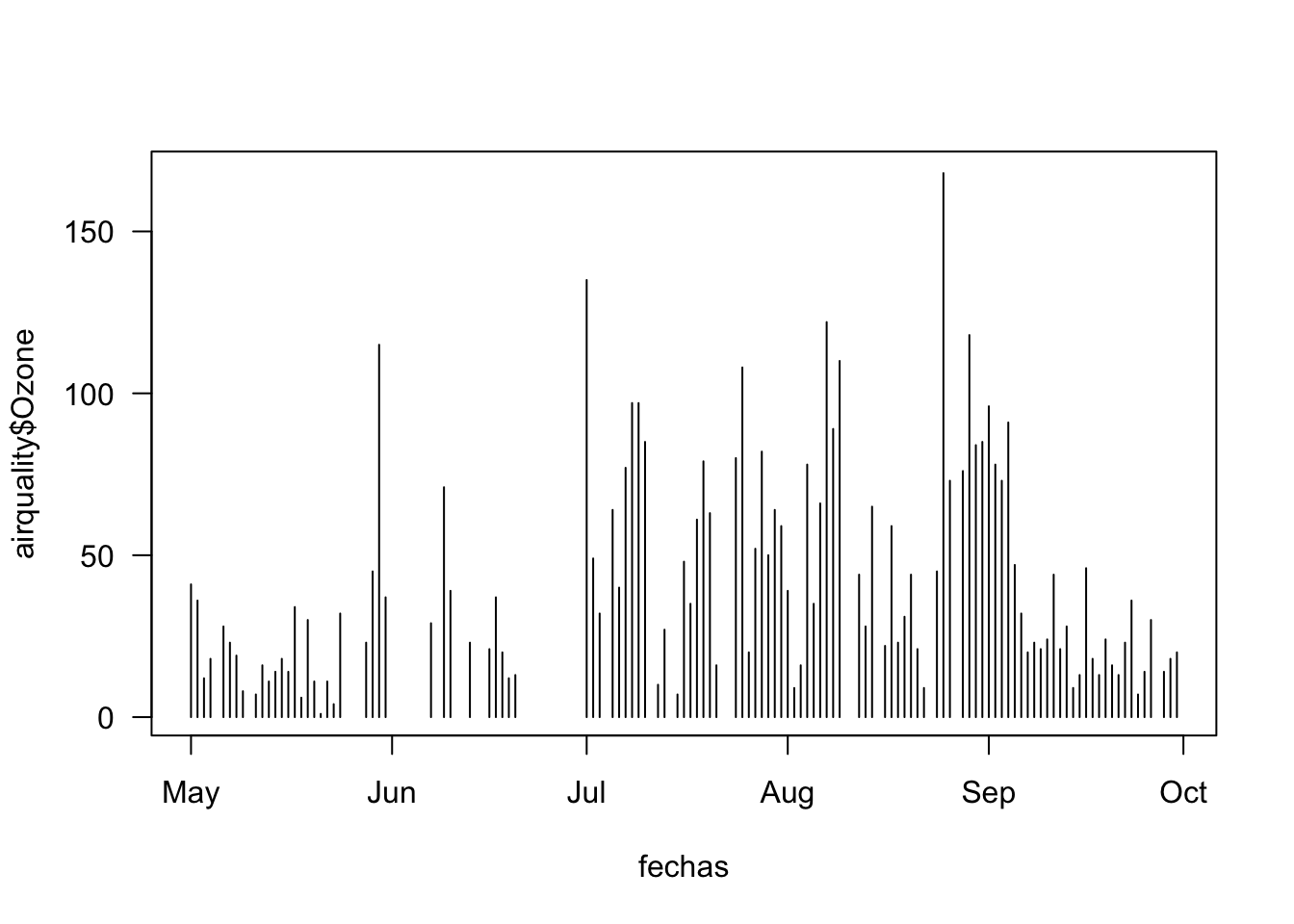
with(airquality,  
 plot(Ozone,type="h",las=1))



Para ver mejor los datos crearemos el vector de fechas:

fechas<-apply(cbind(1973,airquality$Month,airquality$Day),1,paste,collapse="-")  
fechas<-as.Date(fechas,"%Y-%m-%d")

plot(fechas,airquality$Ozone,type="h",las=1)



De la gráfica puede verse que junio es el mes con mas valores faltantes. Veamos cómo se distribuyen los valores faltantes por mes:

aggregate(is.na(airquality$Ozone)~airquality$Month,FUN=sum)

Extraigamos ahora de la fecha los nombres del mes y del día de la semana;

fechas\_mes<-months(fechas)  
fechas\_dia<-weekdays(fechas)

Desafortunadamente, los vectores anteriores no contienen información sobre el orden natural de sus valores. Esta información debe agregarse manualmente así:

echas\_mes<-factor(fechas\_mes,  
 levels=c("May","June","July",  
 "August","September"),ordered = TRUE)  
fechas\_dia<-factor(fechas\_dia,levels=c("Monday","Tuesday","Wednesday",  
 "Thursday","Friday","Saturday", "Sunday"),,ordered = TRUE)  
  
# fechas\_mes<-factor(fechas\_mes,  
# levels=c("mayo","junio","julio",  
# "agosto","septiembre"))  
# fechas\_dia<-factor(fechas\_dia,levels=c("lunes","martes","miércoles",  
# "jueves","viernes","sábado",  
# "domingo"))

Ahora puede verse mejor la distribución de los datos faltantes. Para los meses se tiene:

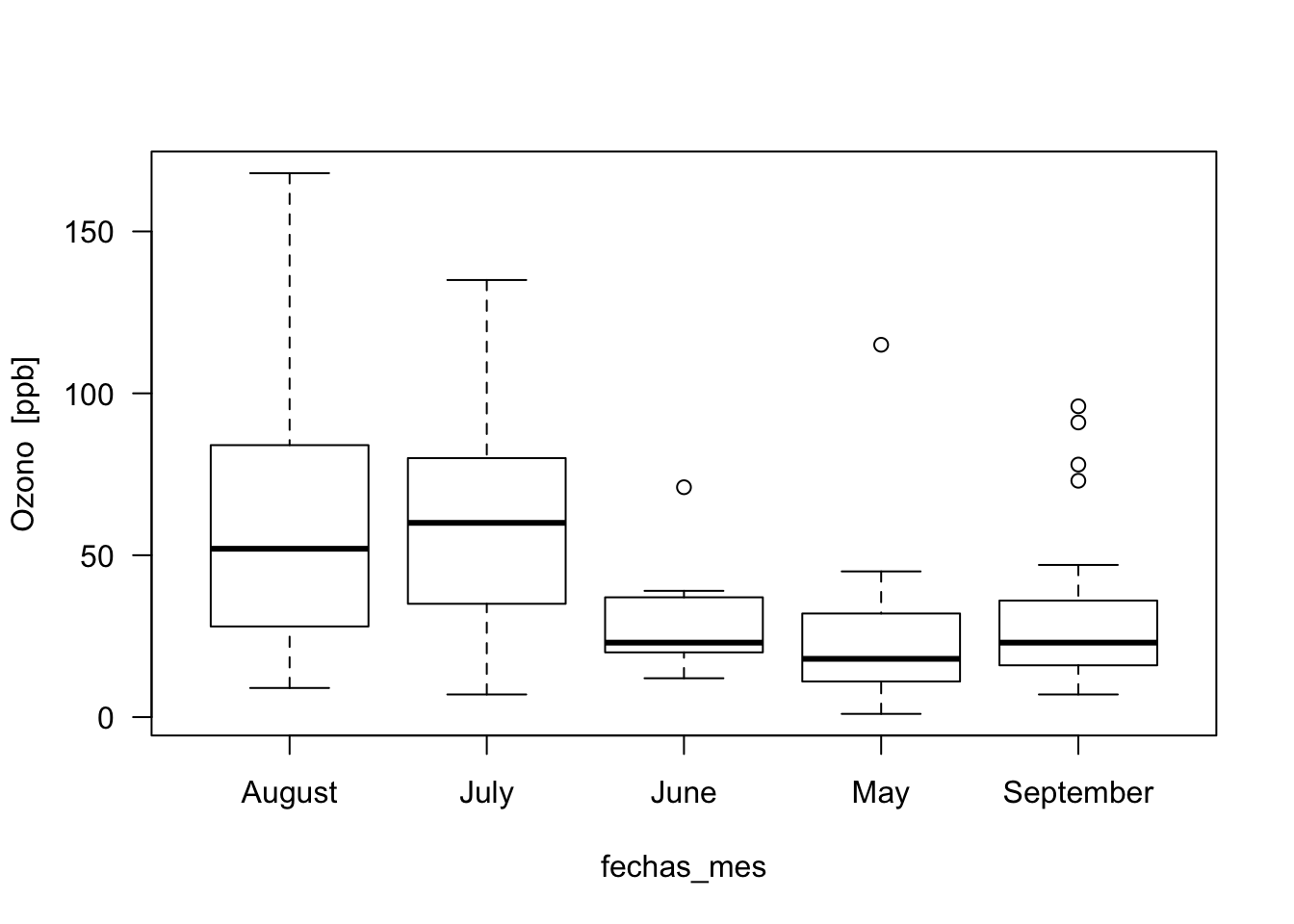
aggregate(is.na(airquality$Ozone)~fechas\_mes,FUN=sum)

Y para los días de la semana se tiene:

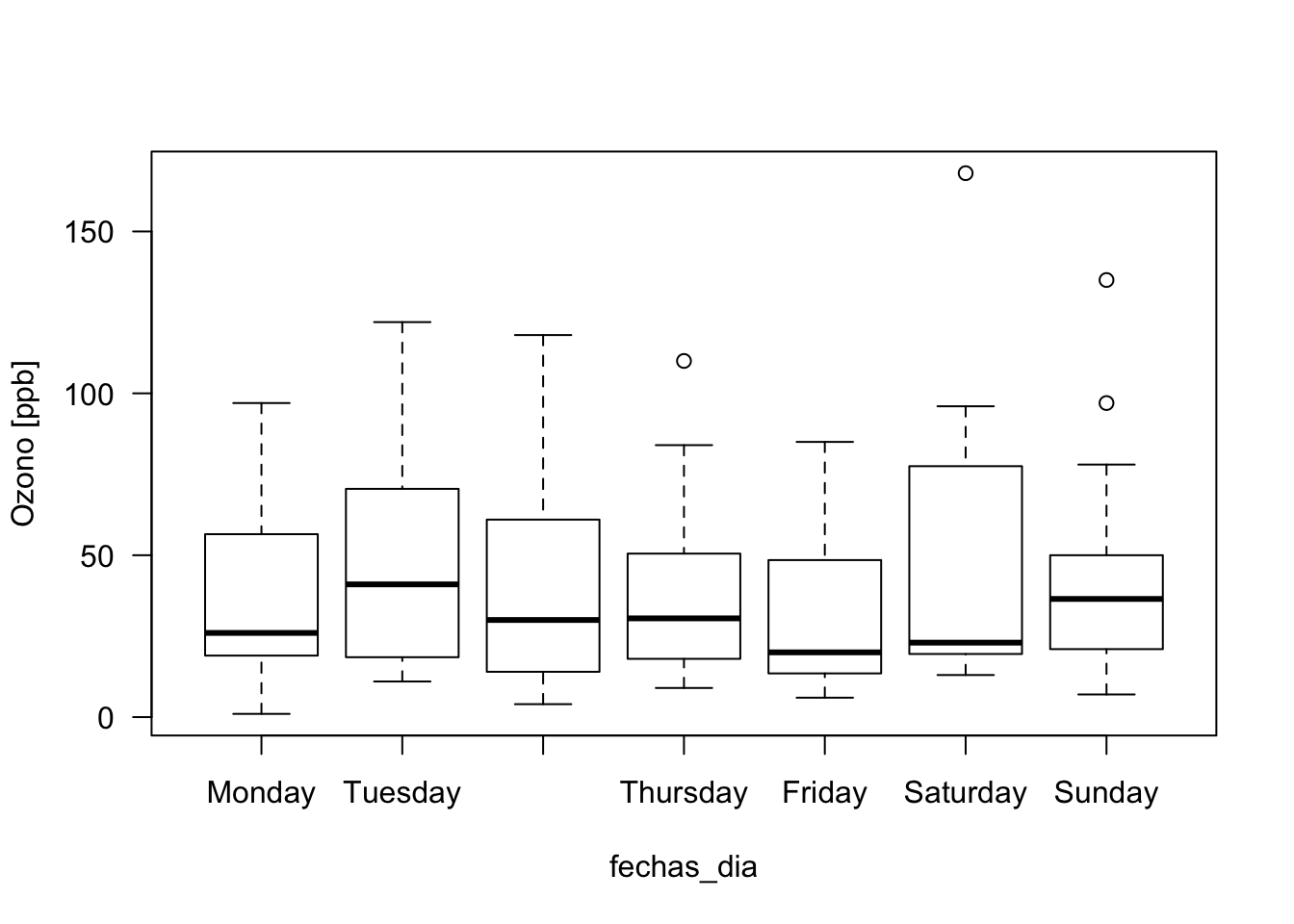
aggregate(is.na(airquality$Ozone)~fechas\_dia,FUN=sum)

Otra forma de explorar la relación de la contaminación con la fecha es con el diagrama de caja y bigotes:

boxplot(airquality$Ozone~fechas\_mes, ylab="Ozono [ppb]",las=1)



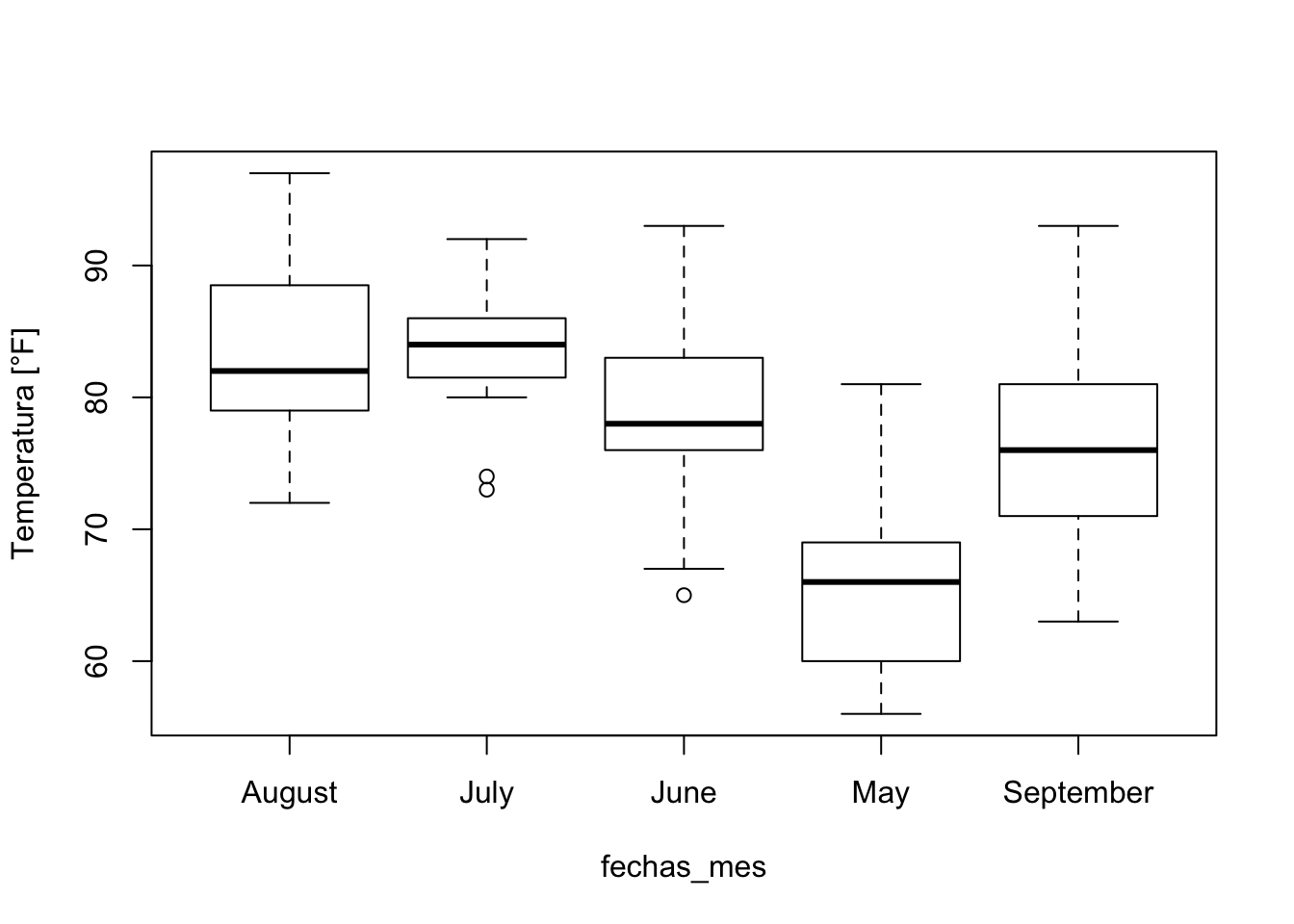
boxplot(airquality$Ozone~fechas\_dia, ylab="Ozono [ppb]",las=1)



En las gráficas anteriores se puede apreciar lo siguiente:

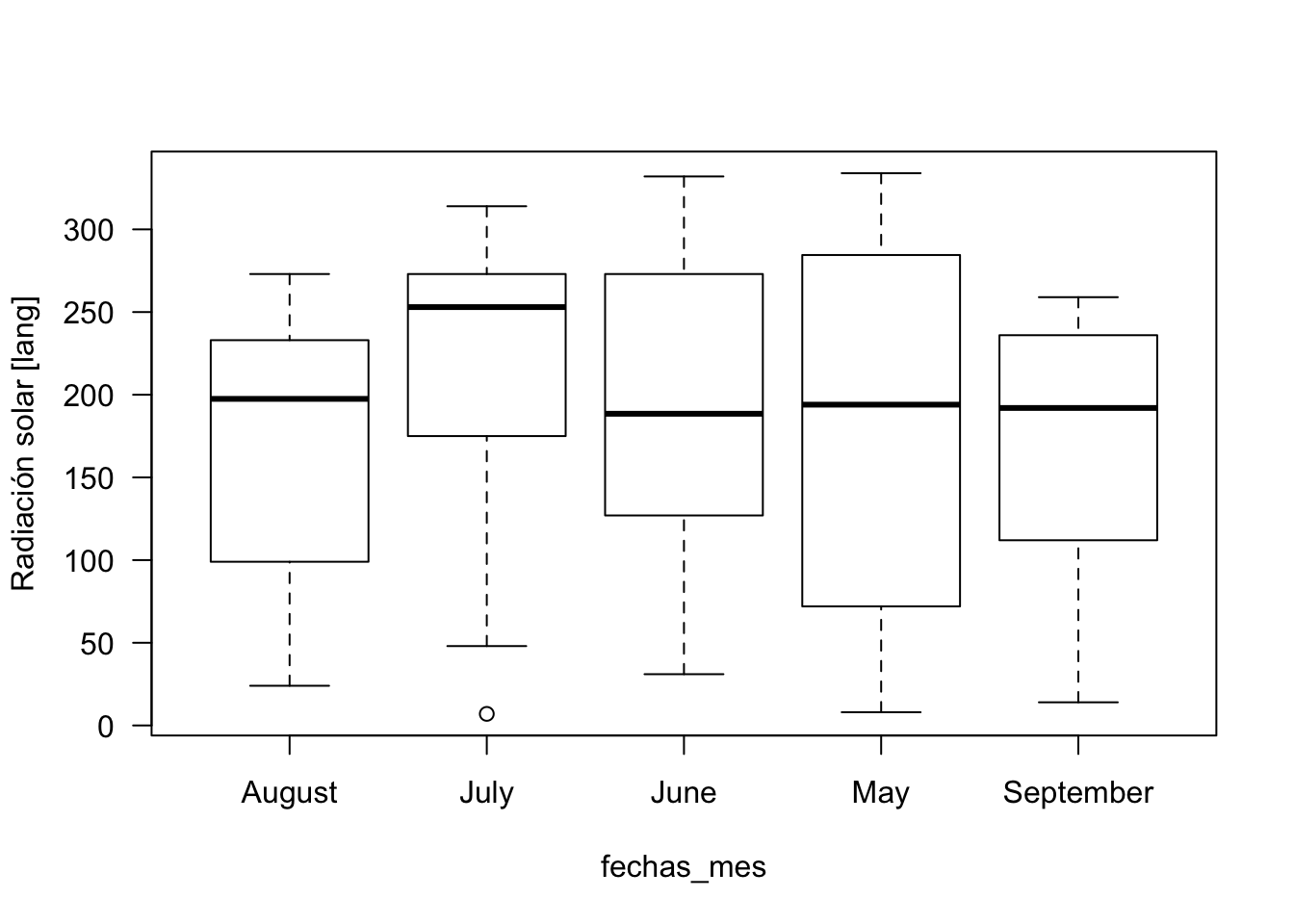
* Junio es el mes con menores niveles de ozono y también el mes de más valores faltantes. Si se excluyeran todas las observaciones de junio se podría introducir un sesgo importante al modelo.
* Hay mayor variabilidad en la variable ozono respecto a la variable mes que respecto a la variable día de la semana. En particular se observa que los meses de julio y agosto presentan los mayores valores. Estos meses son los de mayores temperaturas. Veamoslo:

boxplot(airquality$Temp~fechas\_mes, ylab="Temperatura [°F]")



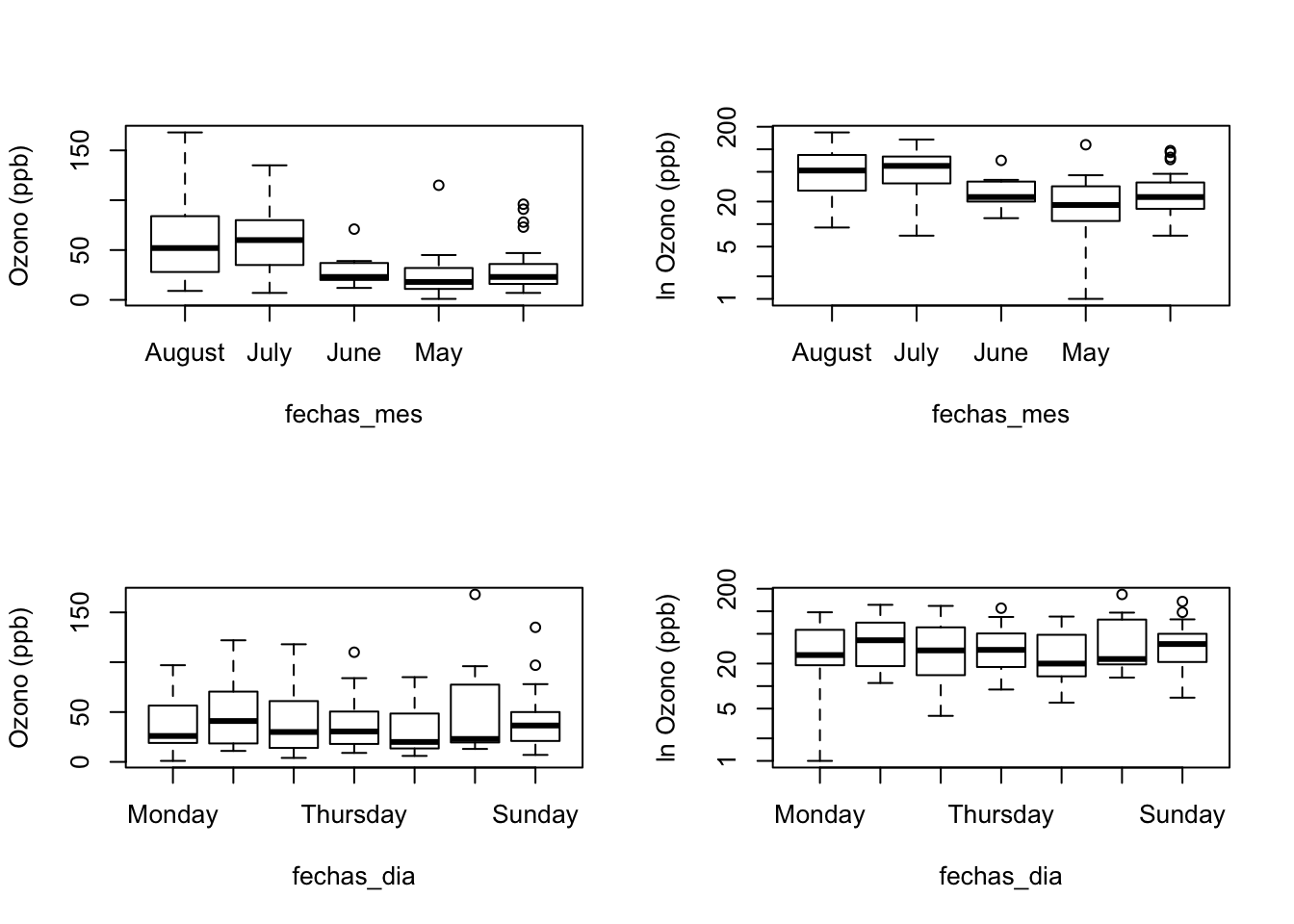
Veamos lo que ocurre para la radiación solar:

boxplot(airquality$Solar.R~fechas\_mes, ylab="Radiación solar [lang]", las=1)



Veamos qué pasa cuando pasamos a la escala logarítmica:

par(mfrow=c(2,2))  
boxplot(airquality$Ozone~fechas\_mes,ylab="Ozono (ppb)")  
boxplot(airquality$Ozone~fechas\_mes,log="y",ylab="ln Ozono (ppb)")  
boxplot(airquality$Ozone~fechas\_dia,ylab="Ozono (ppb)")  
boxplot(airquality$Ozone~fechas\_dia,log="y",ylab="ln Ozono (ppb)")

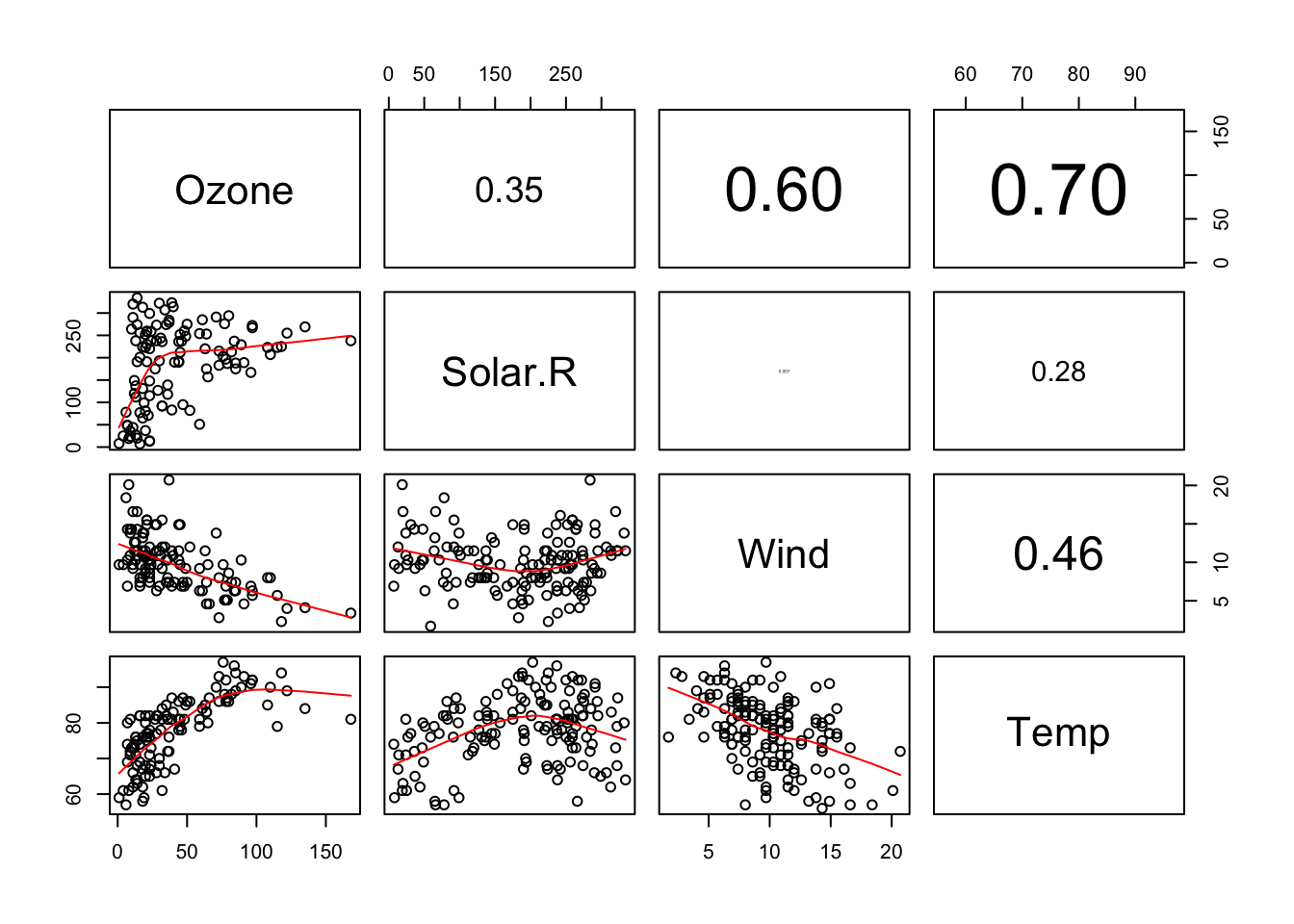


Hasta este punto la transformación logarítmica no resulta ser muy útil. Más adelante se verá su utilidad.

Gráfico de dispersión por pares

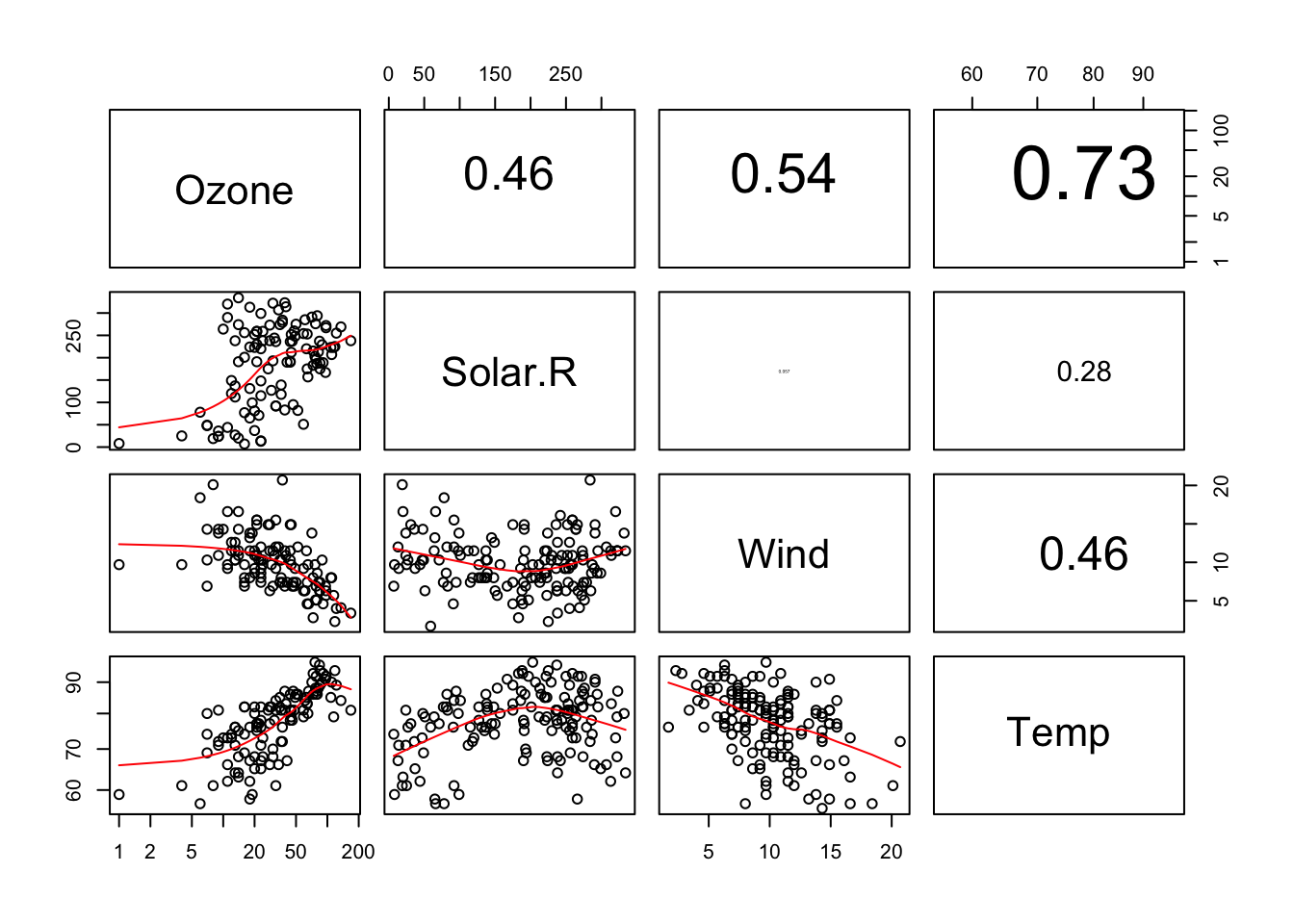
Ahora se obtendrá el gráfico de dispersión por pares. Antes de obtenerla definiremos la función *panel.cor()*, que escribirá la correlación entre los distintos pares de variables:

panel.cor <- function(x, y, digits = 2, prefix = "", cex.cor, ...)  
{  
 usr <- par("usr"); on.exit(par(usr))  
 par(usr = c(0, 1, 0, 1))  
 r <- abs(cor(x, y,use="na.or.complete"))  
 txt <- format(c(r, 0.123456789), digits = digits)[1]  
 txt <- paste0(prefix, txt)  
 if(missing(cex.cor)) cex.cor <- 0.8/strwidth(txt)  
 text(0.5, 0.5, txt, cex = cex.cor \* r)  
}  
pairs(airquality[,1:4], lower.panel = panel.smooth, upper.panel = panel.cor)



Obtengamos el mismo gráfico pero poniendo la variable Ozone en escala logarítmica (note los cambios en la función *panel.cor()* para visualizar correctamente el cambio de escala):

panel.cor <- function(x, y, digits = 2, prefix = "", cex.cor, ...)  
{  
 usr <- par("usr"); on.exit(par(usr))  
 if(par("xlog")){x<-log(x)}  
 if(par("ylog")){y<-log(y)}  
 par(usr = c(0, 1, 0, 1))  
 r <- abs(cor(x, y,use="na.or.complete"))  
 txt <- format(c(r, 0.123456789), digits = digits)[1]  
 txt <- paste0(prefix, txt)  
 if(missing(cex.cor)) cex.cor <- 0.8/strwidth(txt)  
 pos\_x<-ifelse(par("xlog"),4,0.5)  
 pos\_y<-ifelse(par("ylog"),4,0.5)  
 text(pos\_x,pos\_y, txt, cex = cex.cor \* r)  
}  
pairs(airquality[,1:4], lower.panel = panel.smooth, upper.panel = panel.cor,log = c(1,4))



Estimación de un modelo de regresión lineal

Estimación con *lm()*

Estimaremos un modelo logarítmico utilizando la función *lm()*:

modelo<-lm(I(log(Ozone))~Wind+Solar.R+I(Solar.R^2)+Temp,data=airquality)

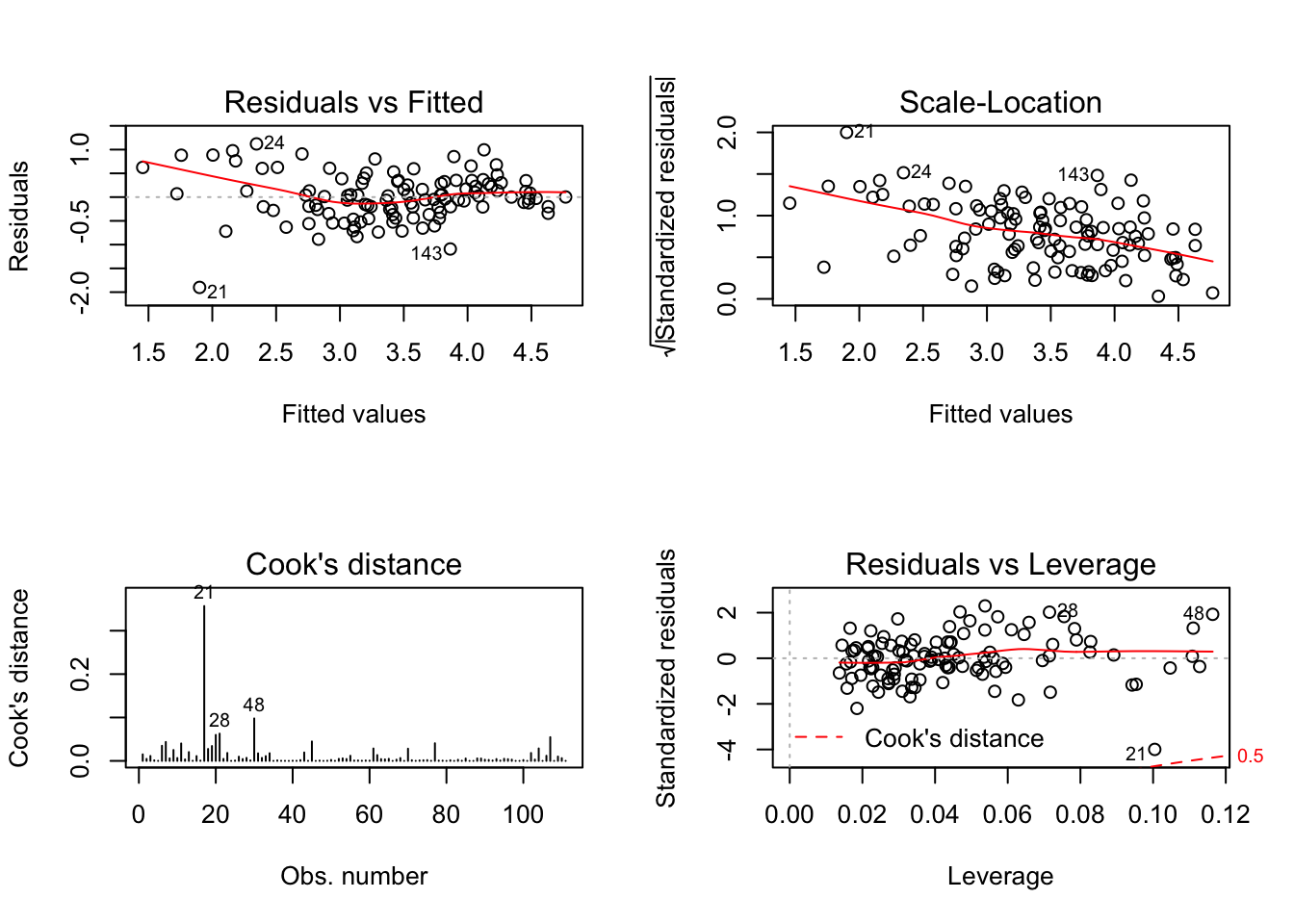
Veamos el desempeño del modelo:

summary(modelo)

##   
## Call:  
## lm(formula = I(log(Ozone)) ~ Wind + Solar.R + I(Solar.R^2) +   
## Temp, data = airquality)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -1.9005 -0.2702 0.0025 0.3128 1.1195   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) -2.369e-01 5.458e-01 -0.434 0.665200   
## Wind -5.876e-02 1.555e-02 -3.779 0.000261 \*\*\*  
## Solar.R 7.259e-03 2.399e-03 3.026 0.003113 \*\*   
## I(Solar.R^2) -1.413e-05 6.957e-06 -2.031 0.044735 \*   
## Temp 4.492e-02 6.355e-03 7.068 1.74e-10 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 0.5013 on 106 degrees of freedom  
## (42 observations deleted due to missingness)  
## Multiple R-squared: 0.677, Adjusted R-squared: 0.6648   
## F-statistic: 55.54 on 4 and 106 DF, p-value: < 2.2e-16

Ahora observemos los diferentes gráficos diangósticos del modelo:

par(mfrow=c(2,2))  
plot(modelo,which=1)  
plot(modelo,which=3)  
plot(modelo,which=4)  
plot(modelo,which=5)



Estimación “manual”

En esta sección se presenta la estimación del modelo anterior utilizando la [factorización matricial QR](https://en.wikipedia.org/wiki/QR_decomposition) ([enlace útil](http://genomicsclass.github.io/book/pages/qr_and_regression.html)).

En lo que sigue asumimos que el modelo de regresión lineal se escribe de forma matricial como \(Y=X\beta+\varepsilon\), donde \(Y\) es un vector de \(n\times 1\), \(X\) es una matriz de datos de dimensión \(n \times p\), \(\beta\) es un vector de coeficientes de dimensión \(p\times 1\) y \(\varepsilon\) es un vector de errores de dimensión \(n\times 1\) de media \(0 \times 1\_{n}\), dispersión \(\Sigma=\sigma^2 I\_{n\times n}\).

El estimador del vectro de coeficientes se puede obtener como \(\hat \beta = (X^TX)^{-1}X^TY\), pero también como el vector que satisface \(X^TX \beta=X^TY\).

Construyamos el vector \(Y\) y la matriz de datos \(X\):

Y<-log(airquality$Ozone)  
X0<-rep(1,length(Y))  
X1<-airquality$Wind  
X2<-airquality$Solar.R  
X3<-airquality$Solar.R^2  
X4<-airquality$Temp  
X<-as.matrix(cbind(X0,X1,X2,X3,X4))  
Y<-as.matrix(Y,nrow=length(Y),ncol=1)

Para estimar el modelo es necesario eliminar los valores faltantes. Podemos utilizar el siguiente código para encontrarlos:

filas\_con\_NA<-which(is.na(apply(cbind(X,Y),1,sum)))

Ahora se retiran los valores faltantes:

Y<-Y[-filas\_con\_NA,1]  
X<-X[-filas\_con\_NA,]

Para estimar \(\beta\) primero se calculará \((X^TX)^{-1}X^T\):

X\_g<-solve(t(X)%\*%X)%\*%t(X)

Ahora se obtienen los coeficientes estimados:

b<-X\_g%\*%Y

La matriz de proyección \(H\) se define como \(H=X(X^TX)^{-1}X^T\) y se calcula así:

H<-X%\*%solve(t(X)%\*%X)%\*%t(X)

Utilizando la matriz \(H\) se puede calcular el vector de residuales sin utilizar los coeficientes, así:

I\_nn<-diag(1,nrow=length(Y))  
r<-(I\_nn-H)%\*%Y

Lo anterior permite tener un estimador de \(\sigma^2\), así:

s2<-t(r)%\*%r/(length(Y)-4-1)  
s<-sqrt(s2)  
print(s)

## [,1]  
## [1,] 0.5013012

Con la descomposición QR

Si se aplica la factorización QR sobre \(X\), entonces ésta puede escribirse como \(X=QR\_1\), con \(Q\) [matriz ortogonal](https://en.wikipedia.org/wiki/Orthogonal_matrix) y \(R\_1=[R^T \ 0\_{(n-p)\times p}^T]^T\), con \(R\) matriz triangular superior. Esta descomposicón se obtiene con la función *qr()*:

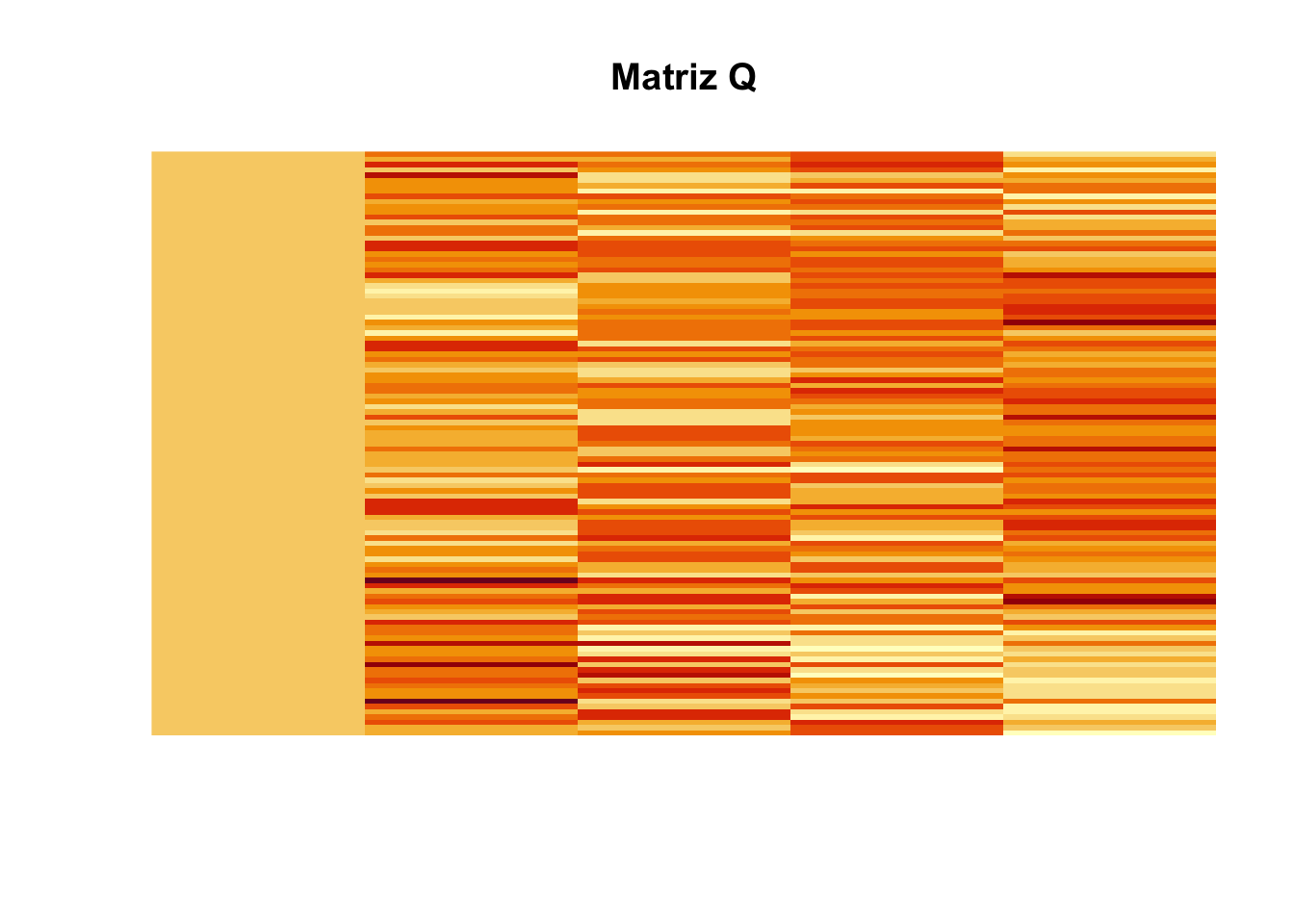
QR\_des<-qr(X)

Ahora se obtienen las matrices \(Q\) y \(R\):

Q<-qr.Q(QR\_des)  
R<-qr.R(QR\_des)

La matriz \(Q\) se puede visualizar como imagen así:

image(t(Q),main="Matriz Q",axes=FALSE)



La matriz \(R\) de dimensión \(p\times p\) se visualiza simplemente con *print()*:

print(R)

## X0 X1 X2 X3 X4  
## [1,] -10.53565 -104.72060 -1947.0078 -446559.95 -819.59793  
## [2,] 0.00000 37.31361 -121.5891 -18455.87 -49.69469  
## [3,] 0.00000 0.00000 948.2498 311345.49 23.26297  
## [4,] 0.00000 0.00000 0.0000 -76324.55 27.52764  
## [5,] 0.00000 0.00000 0.0000 0.00 78.87798

Para estimar \(\beta\) recordemos que este satisface \(X^TX \beta=X^TY\). Pero \(X^TX=R^TR\), así que primero puede resolverse por sustitución el sistema \(R^Tz=X^TY\) para \(z\) y luego se resuelve por sustitución el sistema \(R\beta=z\) para \(\beta\).

Obtengamos \(X^TY\):

XY<-t(X)%\*%Y  
print(XY)

## [,1]  
## X0 3.791679e+02  
## X1 3.580490e+03  
## X2 7.403078e+04  
## X3 1.708641e+07  
## X4 3.017260e+04

Ahora obtengamos \(z\) por sustitución utilizando la función *backsolve()* con el comando upper.tri=FALSE (ya que \(R^T\) es una matriz triangular inferior):

z<-backsolve(t(R),XY,upper.tri = FALSE)  
print(z)

## [,1]  
## [1,] -35.989027  
## [2,] -5.046480  
## [3,] 3.528883  
## [4,] 2.314956  
## [5,] 3.543003

Finalmente obtengamos \(\beta\) por sustitución utilizando nuevamente la función *backsolve()*:

beta<-backsolve(R,z)  
print(beta)

## [,1]  
## [1,] -2.368535e-01  
## [2,] -5.875839e-02  
## [3,] 7.259004e-03  
## [4,] -1.413023e-05  
## [5,] 4.491752e-02

Comparemos los coeficientes obtenidos de las tres maneras:

cbind(coefficients((modelo)),b,beta)

## [,1] [,2] [,3]  
## (Intercept) -2.368535e-01 -2.368535e-01 -2.368535e-01  
## Wind -5.875839e-02 -5.875839e-02 -5.875839e-02  
## Solar.R 7.259004e-03 7.259004e-03 7.259004e-03  
## I(Solar.R^2) -1.413023e-05 -1.413023e-05 -1.413023e-05  
## Temp 4.491752e-02 4.491752e-02 4.491752e-02

Otra manera rápida de utilizar la factorización QR para resolver un modelo lineal es resolver \(R\beta=(Q^TY)\_{n}\) para \(\beta\), donde \((Q^TY)\_{p}\) denota las primeras \(p\) entradas del vector \(Q^TY\). Utilizando esta fórmula se tiene:

beta\_2<-backsolve(R,(t(Q)%\*%Y)[1:5])  
print(beta\_2)

## [1] -2.368535e-01 -5.875839e-02 7.259004e-03 -1.413023e-05 4.491752e-02

Para los errores estándar se tiene que \(\hat {Cov}(\hat \beta)=\hat{\sigma}^2(X^TX)^{-1}\). Pero \((X^TX)^{-1}=(R^TQ^TQR)^{-1}=(R^TR)^{-1}=R^{-1}(R^T)^{-1}\).

Ahora, como \(R\) es una matriz triangular, su inversa se puede obtener por sustitución del sistema \(Rx=u\), para \(u\), donde \(u\) toma los valores \(e\_1=(1 \ 0 \ \ldots \ 0)^T\), \(e\_2=(0 \ 1 \ \ldots \ 0)^T\), \(e\_p=(0 \ 0 \ \ldots \ 1)^T\). Para cada vector \(e\_i\) se obtiene el respectivo valor de \(x\) que es la \(i\)-ésima columna de \(R^{-1}\) ([ver enlace](https://math.stackexchange.com/questions/1143214/method-to-find-the-inverse-of-any-lower-triangular-matrix)). Así se tiene:

e1<-c(1,0,0,0,0)  
e2<-c(0,1,0,0,0)  
e3<-c(0,0,1,0,0)  
e4<-c(0,0,0,1,0)  
e5<-c(0,0,0,0,1)  
r\_1<-backsolve(R,e1)  
r\_2<-backsolve(R,e2)  
r\_3<-backsolve(R,e3)  
r\_4<-backsolve(R,e4)  
r\_5<-backsolve(R,e5)

Luego \(R^{-1}\) se obtiene concatenando los vectores r\_i así:

r\_inv<-as.matrix(cbind(r\_1,r\_2,r\_3,r\_4,r\_5))

Veamos que r\_inv es en efecto \(R^{-1}\):

R%\*%r\_inv

## r\_1 r\_2 r\_3 r\_4 r\_5  
## [1,] 1 0 0 0.000000e+00 -1.776357e-15  
## [2,] 0 1 0 2.775558e-17 0.000000e+00  
## [3,] 0 0 1 0.000000e+00 -5.551115e-17  
## [4,] 0 0 0 1.000000e+00 -5.551115e-17  
## [5,] 0 0 0 0.000000e+00 1.000000e+00

Podemos observar que algunos elementos por fuera de la diagonal no son exactamente cero. Esto se llama *error de redondeo* y puede tener implicaciones en los resultados finales de la estimación. El no invertir la matrix \(R\) sin embargo, representa una eficiencia importante en términos de tiempo.

Como \((R^T)^{-1}=(R^{-1})^{T}\) ya podemos entonces calcular \((X^TX)^{-1}\):

XTX\_inv<-r\_inv%\*%t(r\_inv)  
print(XTX\_inv)

## [,1] [,2] [,3] [,4] [,5]  
## [1,] 1.185395e+00 -2.322958e-02 1.753513e-04 -3.445021e-07 -1.238284e-02  
## [2,] -2.322958e-02 9.621656e-04 1.205307e-05 -3.821426e-08 1.678601e-04  
## [3,] 1.753513e-04 1.205307e-05 2.290258e-05 -6.464940e-08 -2.297629e-05  
## [4,] -3.445021e-07 -3.821426e-08 -6.464940e-08 1.925683e-10 5.796865e-08  
## [5,] -1.238284e-02 1.678601e-04 -2.297629e-05 5.796865e-08 1.607269e-04

Finalmente la matriz de covariazas estimada de los coeficientes es:

cov\_b<-as.numeric(s2)\*XTX\_inv  
print(cov\_b)

## [,1] [,2] [,3] [,4] [,5]  
## [1,] 2.978931e-01 -5.837660e-03 4.406629e-05 -8.657436e-08 -3.111842e-03  
## [2,] -5.837660e-03 2.417949e-04 3.028970e-06 -9.603354e-09 4.218373e-05  
## [3,] 4.406629e-05 3.028970e-06 5.755484e-06 -1.624658e-08 -5.774007e-06  
## [4,] -8.657436e-08 -9.603354e-09 -1.624658e-08 4.839295e-11 1.456769e-08  
## [5,] -3.111842e-03 4.218373e-05 -5.774007e-06 1.456769e-08 4.039112e-05

Comparemos el resultado obtenido con la función \(vcov\) que extrae la matriz de dispersión estimada de los coeficientes:

vcov(modelo)

## (Intercept) Wind Solar.R I(Solar.R^2)  
## (Intercept) 2.978931e-01 -5.837660e-03 4.406629e-05 -8.657436e-08  
## Wind -5.837660e-03 2.417949e-04 3.028970e-06 -9.603354e-09  
## Solar.R 4.406629e-05 3.028970e-06 5.755484e-06 -1.624658e-08  
## I(Solar.R^2) -8.657436e-08 -9.603354e-09 -1.624658e-08 4.839295e-11  
## Temp -3.111842e-03 4.218373e-05 -5.774007e-06 1.456769e-08  
## Temp  
## (Intercept) -3.111842e-03  
## Wind 4.218373e-05  
## Solar.R -5.774007e-06  
## I(Solar.R^2) 1.456769e-08  
## Temp 4.039112e-05

Para completar el ejercicio simplemente calcularemos los estadísticos t y los respectivos valores p:

gl<-dim(X)[1]-4-1 #grados de libertad  
T\_estadisticos<-b/sqrt(diag(cov\_b))  
valores\_p<-2\*pt(-abs(T\_estadisticos),gl)  
print(cbind(T\_estadisticos,valores\_p))

## [,1] [,2]  
## X0 -0.4339599 6.651995e-01  
## X1 -3.7787338 2.607451e-04  
## X2 3.0257715 3.112589e-03  
## X3 -2.0312249 4.473461e-02  
## X4 7.0676134 1.738369e-10