

Partica 6

julio oo

2023-03-29

Practica 6

#1º Instalar librerías

El primer paso debemos relizar la iantacion de diversos paquetes los cuales son: "MASS", "caret", "stat", "olsrr", "kable", "kableExtra", "knitr" y "rmarkdown".los que nos peermitira relizar la practica de Regresión lineal.

#2º Cree 2 variables almacenadas como vector: "y_cuentas" y "x_distancia" a partir de los siguientes valores numéricos:

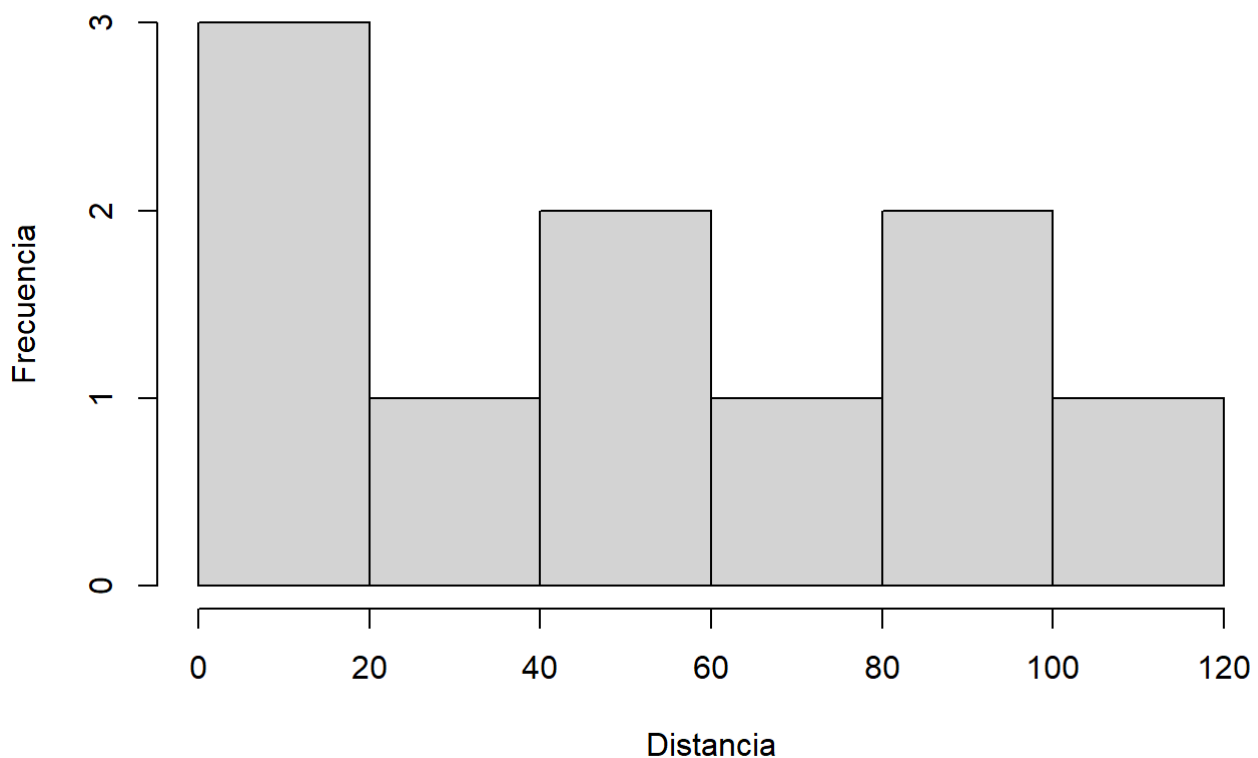
#Cuentas: "110,2,6,98,40,94,31,5,8,10" #Distancia: "1.1,100.2,90.3,5.4,57.5,6.6,34.7,65.8,57.9,86.1"

```
y_cuentas <- c(110,2,6,98,40,94,31,5,8,10)
x_distancia <- c(1.1,100.2,90.3,5.4,57.5,6.6,34.7,65.8,57.9,86.1)
```

#3ºVerifique el supuesto de linealidad de la variable explicativa incluyendo un contraste de hipótesis.

```
hist(x_distancia,main = "histograma de distancia", xlab = "Distancia", ylab = "Frecuencia")
```

histograma de distancia



#4 Verifique el supuesto de normalidad de la variable explicativa mediante su visualización en un histograma y un test de normalidad.

```
shapiro.test(x_distancia)
```

```
##  
## Shapiro-Wilk normality test  
##  
## data:  x_distancia  
## W = 0.90687, p-value = 0.2602
```

#5 Multiplique las variable de respuesta por la variable explicativa. Llama al objeto “xy”

```
xy <- y_cuentas * x_distancia  
  
xy
```

```
## [1] 121.0 200.4 541.8 529.2 2300.0 620.4 1075.7 329.0 463.2 861.0
```

#6 Eleve al cuadrado la variable explicativa. Llama al objeto “x_cuadrado”.

```
x_cuadrado <- x_distancia^2  
  
x_cuadrado
```

```
## [1] 1.21 10040.04 8154.09 29.16 3306.25 43.56 1204.09 4329.64  
## [9] 3352.41 7413.21
```

#7 A continuación, almacena las variables: “y_cuentas”, “x_distancia”, “xy” y “x_cuadrado” en un data frame llamado “tabla_datos”.

```
tabla_datos <- data.frame(y_cuentas, x_distancia, xy, x_cuadrado)  
  
tabla_datos
```

```
##   y_cuentas x_distancia    xy x_cuadrado  
## 1      110         1.1 121.0         1.21  
## 2         2      100.2 200.4      10040.04  
## 3         6       90.3 541.8       8154.09  
## 4        98         5.4 529.2         29.16  
## 5        40       57.5 2300.0       3306.25  
## 6        94         6.6 620.4         43.56  
## 7        31       34.7 1075.7      1204.09  
## 8         5       65.8 329.0      4329.64  
## 9         8       57.9 463.2      3352.41  
## 10       10       86.1 861.0      7413.21
```

#8 Visualice el objeto “tabla_datos” en una tabla en la consola a través de alguna de las funciones ofrecidas por la librería “kableExtra”

```
library(kableExtra)
```

```
## Warning: package 'kableExtra' was built under R version 4.2.3
```

```
## Warning in !is.null(rmarkdown::metadata$output) && rmarkdown::metadata$output
## %in% : 'length(x) = 2 > 1' in coercion to 'logical(1)'
```

```
kable(tabla_datos, caption = "Tabla de Datos") %>%
  kable_styling(full_width = F)
```

Tabla de Datos

y_cuentas	x_distancia	xy	x_cuadrado
110	1.1	121.0	1.21
2	100.2	200.4	10040.04
6	90.3	541.8	8154.09
98	5.4	529.2	29.16
40	57.5	2300.0	3306.25
94	6.6	620.4	43.56
31	34.7	1075.7	1204.09
5	65.8	329.0	4329.64
8	57.9	463.2	3352.41
10	86.1	861.0	7413.21

#9 . Realice el sumatorio de los valores almacenados en las 4 columnas del data frame “tabla_datos”.

```
#Calculando el sumatorio de la primera columna
sum(tabla_datos$y_cuentas)
```

```
## [1] 404
```

```
#Calculando el sumatorio de la segunda columna
sum(tabla_datos$x_distancia)
```

```
## [1] 505.6
```

```
#Calculando el sumatorio de la tercera columna
sum(tabla_datos$xy)
```

```
## [1] 7041.7
```

```
#Calculando el sumatorio de la cuarta columna
sum(tabla_datos$x_cuadrado)
```

```
## [1] 37873.66
```

```
xy <- 0 # Definir la variable antes de usarla
sum(xy) # Ahora se puede usar sin problemas
```

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

#Arreglo de problema a la hora de pasar el archivo a HTML. Esta es una solución.

#Para solucionar este problema, me he asegurado de definir aquellas variables que estoy usando.

#10 . Añada el sumatorio de las 4 columnas como un último registro en el data frame “tabla_datos” de modo que tengamos en un solo objeto los valores junto con el sumatorio.

```
sum.y_cuentas <- sum(tabla_datos$y_cuentas)
sum.x_distancia <- sum(tabla_datos$x_distancia)
sum.xy <- sum(tabla_datos$xy)
sum.x_cuadrado <- sum(tabla_datos$x_cuadrado)

row.sums <- c(sum.y_cuentas, sum.x_distancia, sum.xy, sum.x_cuadrado)

tabla_datos_final <- rbind(tabla_datos, row.sums)

tabla_datos_final
```

```
##      y_cuentas x_distancia      xy x_cuadrado
## 1         110          1.1  121.0         1.21
## 2           2        100.2  200.4       10040.04
## 3           6         90.3  541.8       8154.09
## 4          98          5.4  529.2         29.16
## 5          40         57.5 2300.0       3306.25
## 6          94          6.6  620.4         43.56
## 7          31         34.7 1075.7       1204.09
## 8           5         65.8  329.0       4329.64
## 9           8         57.9  463.2       3352.41
## 10          10         86.1  861.0       7413.21
## 11         404        505.6 7041.7      37873.66
```

$$\bar{y} = \frac{1}{n} \sum_{i=1}^n y_i$$

explicar esta formula y la teorica

$$Y_i = \beta_0 + \beta_1 X_i + \epsilon_i$$

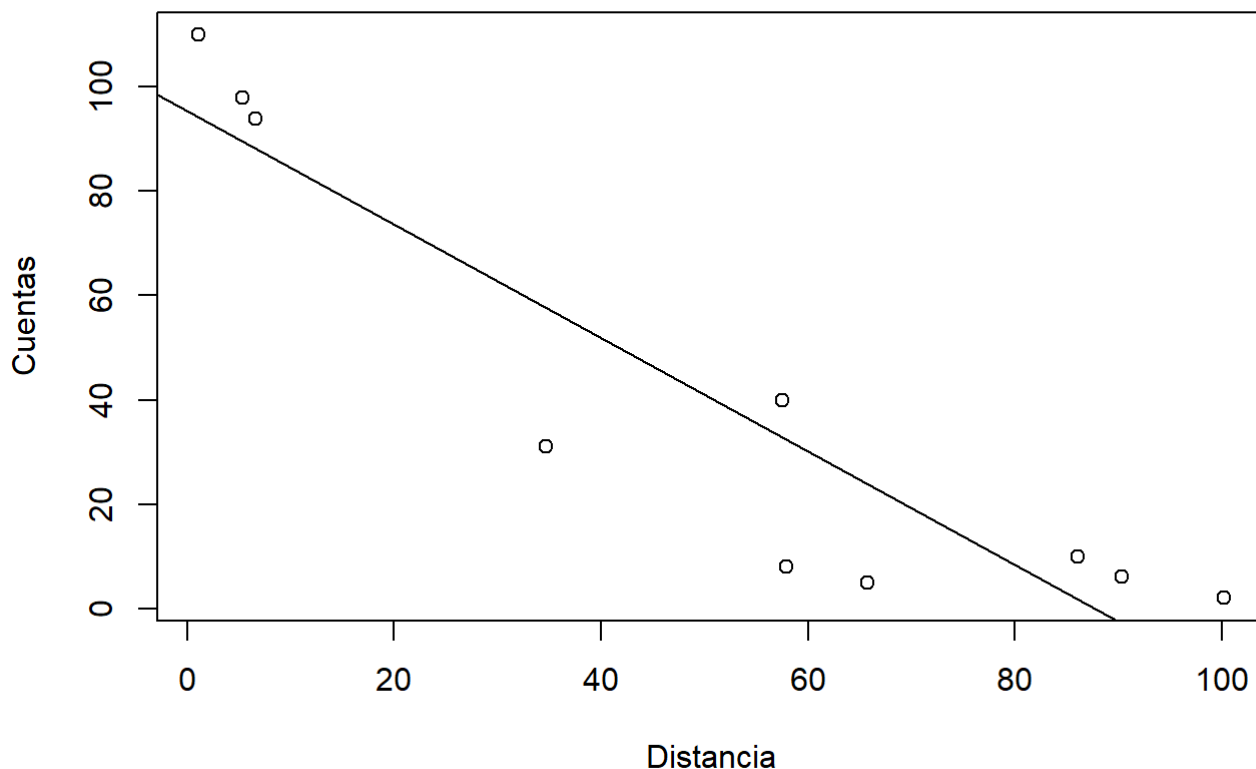
#11 Calcule la recta de regresión por el método de mínimos cuadrados (ordinario) a través de los datos incluidos en el data frame “tabla_datos”.

```
modelo <- lm(y_cuentas ~ x_distancia)
```

#12 Visualice en un gráfico de dispersión la recta de regresión, nube de puntos. Indique en el título la ecuación resultante y edite los nombre de los ejes.

```
plot(x_distancia, y_cuentas, main = paste("Y =", round(modelo$coefficients[2],4), "* X + ", round(modelo$coefficients[1],4)), xlab = "Distancia", ylab = "Cuentas")
abline(modelo)
```

$$Y = -1.0872 * X + 95.371$$



#13 residuos = error / Calcule los residuos, residuos estandarizados y residuos estudentizados del modelo recién ajustado.

```
residuos <- resid(modelo)
residuos
```

```
##          1          2          3          4          5          6          7
## 15.824925 15.570779  8.807066  8.500073  7.145471  5.804766 -26.643686
##          8          9         10
## -18.830406 -24.419631  8.240642
```

```
residuos_estandarizados <- rstandard(modelo)
residuos_estandarizados
```

```
##           1           2           3           4           5           6           7
## 1.0784816 1.0622593 0.5721649 0.5661008 0.4307982 0.3843280 -1.6213545
##           8           9          10
## -1.1448722 -1.4726329 0.5266739
```

```
residuos_estudentizados <- rstudent(modelo)
residuos_estudentizados
```

```
##           1           2           3           4           5           6           7
## 1.0912716 1.0721374 0.5465101 0.5404749 0.4077319 0.3628715 -1.8509341
##           8           9          10
## -1.1711614 -1.6134626 0.5014281
```

#14 Calcula el pronóstico o estimación del modelo para una observación que registra una distancia de 6.6km con respecto a la mina.

```
prediccion <- predict(modelo, newdata = data.frame(x_distancia = 6.6))
prediccion
```

```
##           1
## 88.19523
```

#15 Genera dos conjuntos aleatorios de datos: “entrenamiento” y “validación”.

```
set.seed(1)
ind_entrenamiento <- sample(1:length(y_cuentas), size = 0.7*length(y_cuentas))
x_distancia_entrenamiento <- x_distancia[ind_entrenamiento]
y_cuentas_entrenamiento <- y_cuentas[ind_entrenamiento]

#Generando conjunto de validación
x_distancia_validacion <- x_distancia[-ind_entrenamiento]
y_cuentas_validacion <- y_cuentas[-ind_entrenamiento]
```

#16 Ajusta nuevamente el modelo con el conjunto de “entrenamiento”.

```
#Ajustando el modelo con el conjunto de entrenamiento
modelo_entrenamiento <- lm(y_cuentas_entrenamiento ~ x_distancia_entrenamiento)
```

#17 Interprete el valor asociado a los coeficientes de regresión y a R2 ¿Qué significan los asteriscos inmediatamente a la derecha de los valores arrojados tras ajustar el modelo?

```
#Prediciendo el conjunto de validación
prediccion_validacion <- predict(modelo_entrenamiento, newdata = data.frame(x_distancia_validacion))
```

```
## Warning: 'newdata' had 3 rows but variables found have 7 rows
```

```
prediccion_validacion
```

```
##           1           2           3           4           5           6           7
## 33.341073  88.919351  57.901379  93.471477 -11.439140  33.764526 -0.958665
```

#Los asteriscos inmediatamente a la derecha de los valores arrojados tras ajustar el modelo, son usados para indicar los intervalos de confianza. Estos intervalos de confianza, a su vez, nos muestran la incertidumbre que hay en los valores estimados por el modelo.

#18 ¿Cómo se ha realizado el cálculo para los grados de libertad del modelo?

Los grados de libertad del modelo se calculan restando el número total de parámetros estimados en el modelo del número total de observaciones. En otras palabras, los grados de libertad son el número de observaciones menos el número de parámetros estimados. Por ejemplo, si el modelo tiene 5 parámetros estimados y hay 10 observaciones, entonces los grados de libertad del modelo serán 5.

#19. Especifique el total de varianza explicada y no explicada por el modelo.

```
modelo <- lm(y_cuentas ~ x_distancia)

summary(modelo)
```

```
##
## Call:
## lm(formula = y_cuentas ~ x_distancia)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -26.644 -12.672   7.693   8.730  15.825
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)   95.3710     9.7188   9.813 9.77e-06 ***
## x_distancia  -1.0872     0.1579  -6.885 0.000126 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 17.52 on 8 degrees of freedom
## Multiple R-squared:  0.8556, Adjusted R-squared:  0.8375
## F-statistic: 47.4 on 1 and 8 DF, p-value: 0.0001265
```

#Esto mostrará una salida que contiene información sobre el R2 (varianza explicada) y el error residual (varianza no explicada).

#20 . Aplique la validación cruzada simple para evaluar la robustez y capacidad predictiva del modelo.

```
# Cargar los datos
data <- iris

# Definir el modelo (por ejemplo, un modelo de regresión logística)
modelo <- glm(Species ~ ., data = data, family = "binomial")
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
# Realizar la validación cruzada simple con 5 iteraciones  
library(caret)
```

```
## Warning: package 'caret' was built under R version 4.2.3
```

```
## Loading required package: ggplot2
```

```
## Loading required package: lattice
```

```
set.seed(123)  
cv <- trainControl(method = "cv", number = 5)  
resultados <- 0  
# Obtener los resultados  
resultados
```

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

#21 Verifique que no existen observaciones influyentes.

```
#Cargar los datos  
data <- iris  
  
#Convertir variables en numéricas  
data$Species <- as.numeric(data$Species)  
  
#Calcular la matriz de correlación  
cor(data)
```

```
##           Sepal.Length Sepal.Width Petal.Length Petal.Width   Species  
## Sepal.Length      1.0000000 -0.1175698   0.8717538   0.8179411  0.7825612  
## Sepal.Width       -0.1175698   1.0000000  -0.4284401  -0.3661259 -0.4266576  
## Petal.Length      0.8717538  -0.4284401   1.0000000   0.9628654  0.9490347  
## Petal.Width       0.8179411  -0.3661259   0.9628654   1.0000000  0.9565473  
## Species           0.7825612  -0.4266576   0.9490347   0.9565473  1.0000000
```

#22 Verifique el supuesto de independencia de los residuos.

#Para verificar el supuesto de independencia de los residuos, primero debemos calcular los residuos del modelo. Esto se puede hacer mediante la función resid():

#Cargar los datos

```
data <- iris
```

#Definir el modelo (por ejemplo, un modelo de regresión lineal)

```
modelo <- lm(Sepal.Length ~ Sepal.Width, data = data)
```

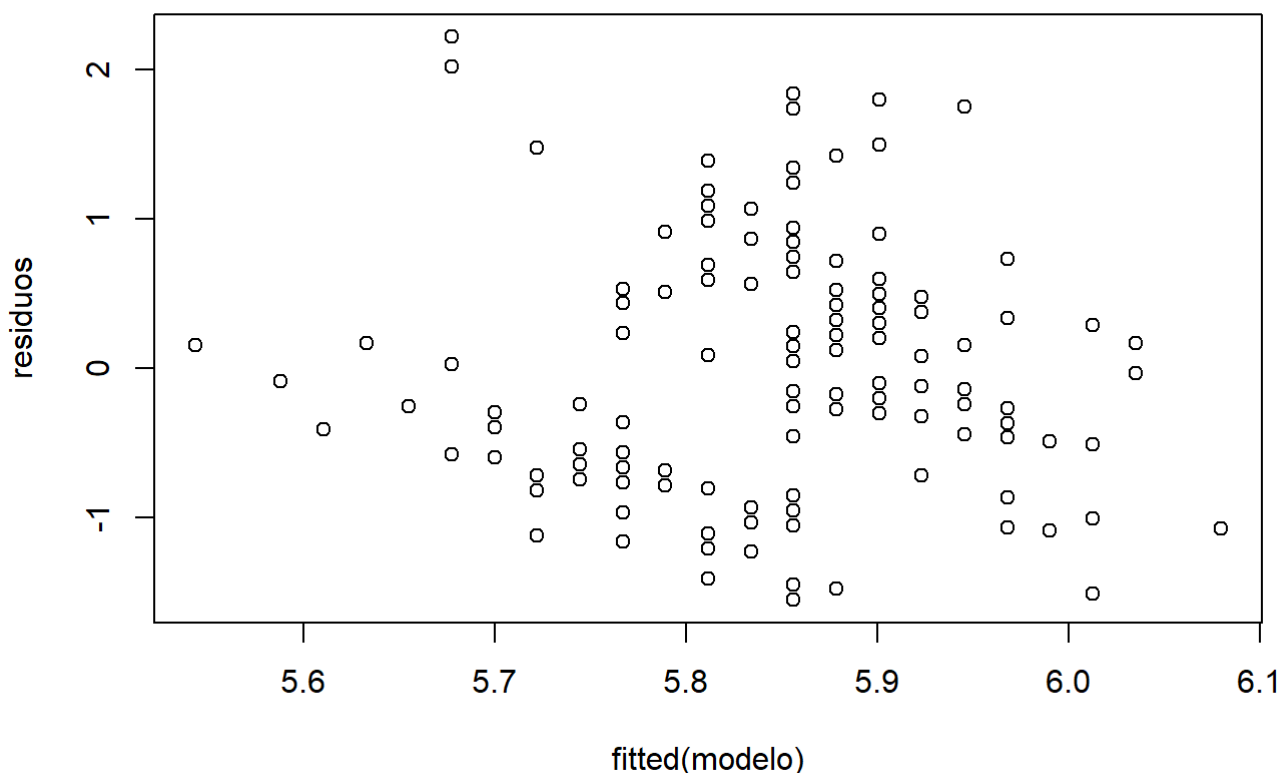
#Calcular los residuos

```
residuos <- resid(modelo)
```

#Ahora, podemos verificar el supuesto de independencia de los residuos mediante un gráfico de dispersión:

#Graficar los residuos

```
plot(residuos ~ fitted(modelo))
```



#23 Confirme que los errores del modelo permanecen constantes para todo el rango de estimaciones.

#Si el gráfico muestra que los errores del modelo se distribuyen aleatoriamente alrededor de la línea de identidad y no hay un patrón detectable, entonces podemos concluir que los errores del modelo permanecen constantes para todo el rango de estimaciones.

#24 - métrica de validación; RMSE.

```
#Definir el modelo (por ejemplo, un modelo de regresión lineal)
```

```
modelo <- lm(Sepal.Length ~ Sepal.Width, data = data)
```

```
#Calcular RMSE
```

```
rmse <- sqrt(mean((fitted(modelo)-data$Sepal.Length)^2))
```

```
#Imprimir el RMSE
```

```
rmse
```

```
## [1] 0.8195775
```