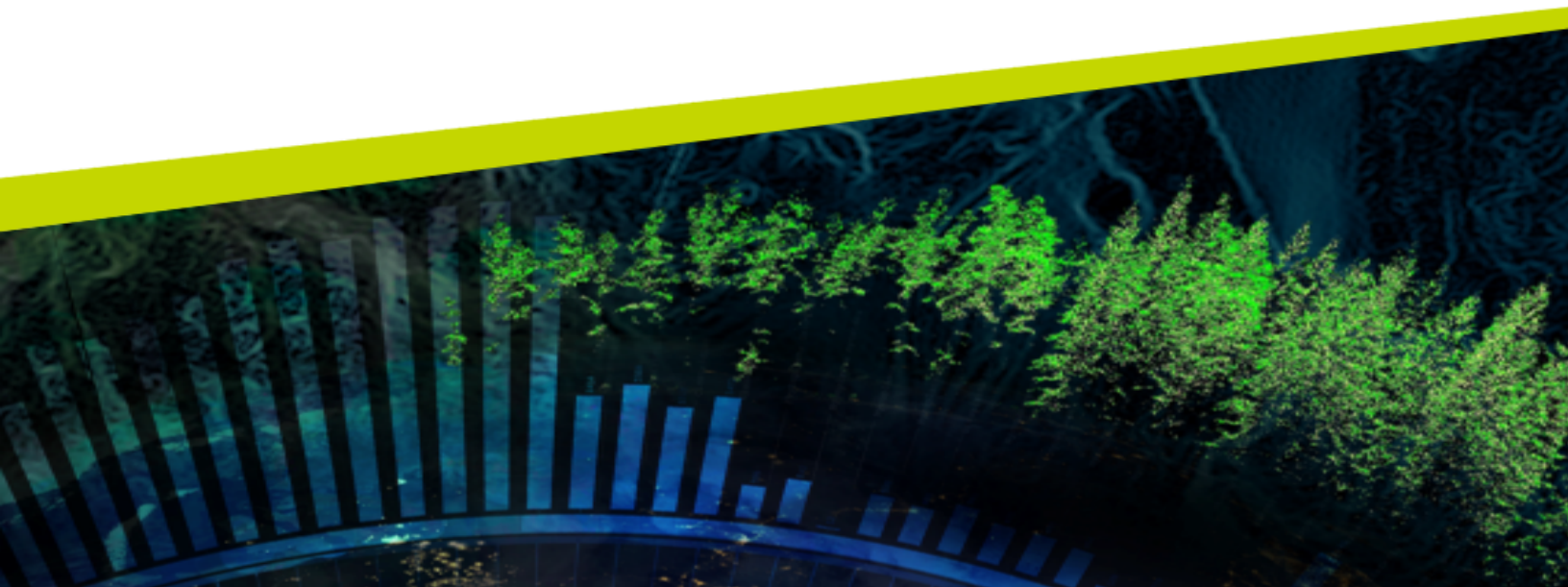
INFERENCIA Y MODELOS ESTADÍSTICOS

Jacqueline Köhler C. y José Luis Jara V.



CAPÍTULO 13. OTRAS ALTERNATIVAS PARA DATOS PROBLEMÁTICOS

Como ya sabemos, muchos procedimientos estadísticos requieren que los datos cumplan con ciertas propiedades o condiciones, lo que no siempre ocurre. En capítulos anteriores hemos visto que, ante este escenario, podemos usar como alternativa métodos no paramétricos (capítulos 8 y 11) o remuestreo (capítulo 12), aunque no son las únicas opciones. En este capítulo abordaremos otras estrategias que podemos usar cuando necesitamos analizar datos problemáticos.

13.1 TRANSFORMACIÓN DE DATOS

Otro fenómeno que ocurre a menudo en los estudios, además del incumplimiento de ciertas condiciones, es la necesidad de convertir los datos de una escala a otra diferente. Para hacer tales transformaciones, debemos aplicar una determinada función a una variable aleatoria X, lo que nos entrega como resultado una nueva variable aleatoria Y.

Como explica Lane (s.f., pp. 1, 16), existen diversos métodos que podemos usar para transformar datos, dependiendo de la forma que tengan los datos originales y la que deseemos obtener como resultado. En esta sección conoceremos, entonces, algunas transformaciones de uso frecuente en estadística.

13.1.1 Transformación lineal

Las **transformaciones lineales** son las más sencillas de las transformaciones, y para hacerlas nos basta con aplicar una función lineal, es decir, de la forma presentada en la ecuación 13.1, donde m y n son constantes.

$$y_i = m \cdot x_i + n \tag{13.1}$$

La física nos ofrece muchos escenarios en que es necesario aplicar este tipo de transformaciones, pues es el tipo de operación que realizamos cuando convertimos de una unidad a otra. A modo de ejemplo, consideremos la conversión de grados Celsius a grados Fahrenheit:

$$F = 1,8 \cdot C + 32$$

En R, podemos hacer este tipo de transformaciones de forma muy sencilla mediante operaciones aritméticas que se aplican vectorialmente, como muestra el script 13.1. En él se transforma un vector con 4 temperaturas en grados Celsius a grados Fahrenheit. El resultado se muestra en la figura 13.1.

Script 13.1: transformación lineal para convertir grados Celcius a grados Fahrenheit.

```
# Crear un vector con cuatro observaciones en grados Celsius.
Celsius <- c(-8, 0, 29.8, 100)

```

```r
# Aplicar transformación lineal para convertir a grados Fahrenheit.
Fahrenheit <- 1.8 * Celsius + 32

# Mostrar los resultados.
cat("Temperaturas en grados Celsius\n")
print(Celsius)
cat("\nTemperaturas en grados Fahrenheit\n")
print(Fahrenheit)
```

```
Temperaturas en grados Celsius
[1]  -8.0   0.0  29.8 100.0

Temperaturas en grados Fahrenheit
[1]  17.60  32.00  85.64 212.00
```

Figura 13.1: resultado de la transformación lineal del script 13.1.

13.1.2 Transformación logarítmica

La transformación logarítmica nos puede servir cuando tenemos distribuciones muy asimétricas, pues ayuda a reducir la desviación y así facilita el cumplimiento de la condición de normalidad requerida por muchas de las pruebas estadísticas que ya conocemos. Para ver este efecto de manera más clara, usaremos un conjunto de datos que registra el peso corporal (en kilogramos) y el peso del cerebro (en gramos) de diversos animales, algunos de ellos extintos (Rousseeuw & Leroy, 1987, p. 57). En R, esta transformación puede hacerse gracias a la función `log(x, base)`, aunque debemos tener cuidado con posibles valores iguales a 0. El script 13.2 aplica esta transformación al peso corporal y al peso cerebral de los animales (líneas 22–23). La figura 13.2 (script 13.2, líneas 28–43) muestra gráficamente el resultado de esta transformación para el peso cerebral de los animales.

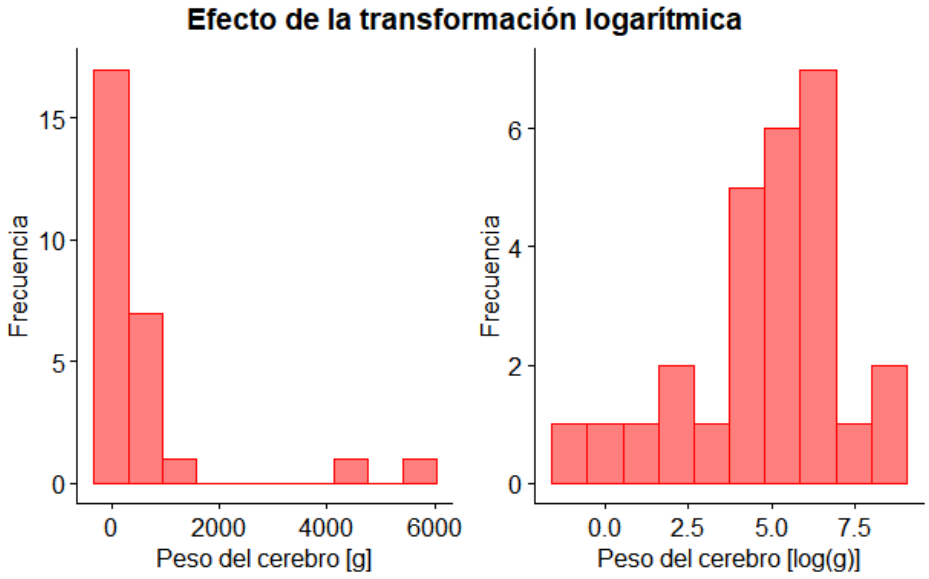


Figura 13.2: histogramas del peso cerebral antes y después de la transformación logarítmica.

Muchas veces la transformación logarítmica hace que nos sea más fácil interpretar los datos, evidenciando patrones más claros para la relación entre variables. En la figura 13.3 (script 13.2, líneas 47–60), a la derecha, se evidencia una fuerte relación entre el peso corporal y el peso del cerebro tras transformar ambas variables, relación que no podemos percibir con los datos originales (izquierda).

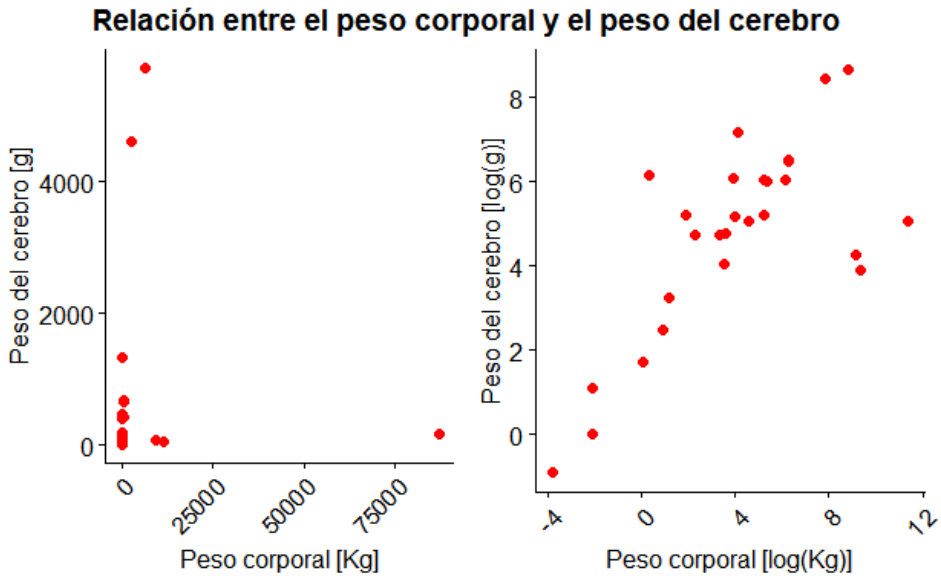


Figura 13.3: gráficos de dispersión para el peso corporal y el peso del cerebro antes y después de las transformaciones logarítmicas.

Script 13.2: transformación logarítmica.

```
library(ggpubr)

# Cargar datos
animal <- c("Mountain beaver", "Cow", "Grey wolf", "Goat", "Guinea pig",
            "Dipliodocus", "Asian elephant", "Donkey", "Horse",
            "Potar monkey", "Cat", "Giraffe", "Gorilla", "Human",
            "African elephant", "Triceratops", "Rhesus monkey", "Kangaroo",
            "Golden hamster", "Mouse", "Rabbit", "Sheep", "Jaguar",
            "Chimpanzee", "Brachiosaurus", "Mole", "Pig")

body_weight <- c(1.35, 465, 36.33, 27.66, 1.04, 11700, 2547, 187.1, 521, 10,
                 3.3, 529, 207, 62, 6654, 9400, 6.8, 35, 0.12, 0.023, 2.5,
                 55.5, 100, 52.16, 87000, 0.122, 192)

brain_weight <- c(465, 423, 119.5, 115, 5.5, 50, 4603, 419, 655, 115, 25.6,
                  680, 406, 1320, 5712, 70, 179, 56, 1, 0.4, 12.1, 175, 157,
                  440, 154.5, 3, 180)

datos <- data.frame(animal, body_weight, brain_weight)

# Aplicar transformación logarítmica
log_cuerpo <- log(body_weight)
log_cerebro <- log(brain_weight)
datos <- data.frame(datos, log_cuerpo, log_cerebro)

# Histogramas para el peso cerebral antes y después de la transformación
# logarítmica.
g3 <- gghistogram(datos, x = "brain_weight", bins = 10,
```

```
                    xlab = "Peso del cerebro [g]", ylab = "Frecuencia",
                    color = "red", fill = "red")

g4 <- gghistogram(datos, x = "log_cerebro", bins = 10,
                    xlab = "Peso del cerebro [log(g)]", ylab = "Frecuencia",
                    color = "red", fill = "red")

# Crear una única figura con ambos histogramas.
histograma  <- ggarrange(g3, g4, ncol = 2, nrow = 1)

titulo  <- text_grob("Efecto de la transformación logarítmica",
                        face = "bold", size = 14)

histograma <- annotate_figure(histograma, top = titulo)
print(histograma)

# Gráficos de dispersión para la relación entre peso corporal y peso del
# cerebro, antes y después de aplicar la transformación logarítmica.
g1 <- ggscatter(datos, x = "body_weight", y = "brain_weight",
                  color = "red", xlab = "Peso corporal [Kg]",
                  ylab = "Peso del cerebro [g]") + rotate_x_text(45)

g2 <- ggscatter(datos, x = "log_cuerpo", y = "log_cerebro",
                  color = "red", xlab = "Peso corporal [log(Kg)]",
                  ylab = "Peso del cerebro [log(g)]") + rotate_x_text(45)

# Crear una única figura con los gráficos de dispersión.
dispersion  <- ggarrange(g1, g2, ncol = 2, nrow = 1)
texto <- "Relación entre el peso corporal y el peso del cerebro"
titulo  <- text_grob(texto, face = "bold", size = 14)
dispersion <- annotate_figure(dispersion, top = titulo)
print(dispersion)
```

Eso sí, tenemos que ser muy cuidadosos al usar esta transformación, porque cuando comparamos medias de datos tras una transformación logarítmica, ¡en realidad estamos comparando **medias geométricas**! Recordemos que la media geométrica se calcula de acuerdo a la ecuación 13.2 y suele ocuparse para representar tasas de crecimiento o de interés (Glen, 2021).

$$\left(\prod_{i=1}^{n} x_i\right)^{\frac{1}{n}} = \sqrt[n]{x_1 \cdot x_2 \cdot \ldots \cdot x_n} \tag{13.2}$$

Para ilustrar esta idea, supongamos que aplicamos una transformación logarítmica con base 10 al vector $[1, 10, 100]$, obteniendo como resultado $[0, 1, 2]$.

La media aritmética del vector transformado es:

$$\frac{0+1+2}{3} = 1$$

Al revertir la transformación para la media, tenemos que:

$$10^1 = 10$$

Lo que es distinto a la media aritmética de los datos originales:

$$\frac{1+10+100}{3} = 37$$

A su vez, la media geométrica del vector original es:

$$\sqrt[10]{1 \cdot 10 \cdot 100} = 10$$

Así, si dos variables a las que se ha aplicado la transformación logarítmica tienen igual media, entonces **las medias geométricas de las variables originales son iguales**.

13.1.3 Escalera de potencias de Tukey

Más general que la transformación logarítmica, la **escalera de potencias de Tukey** nos ayuda a cambiar la forma de una distribución asimétrica para que se asemeje a la normal. Este método consiste en explorar relaciones de la forma que muestra la ecuación 13.3, donde λ puede tomar cualquier valor real y se escoge de modo que la distribución de los datos transformados sea lo más cercana a la normal posible. También es útil al explorar la relación entre dos variables, en cuyo caso se busca obtener un gráfico de dispersión en que los puntos se asemejen a una recta.

$$y = x^{\lambda} \tag{13.3}$$

Formalmente, la transformación de Tukey se define según la ecuación 13.4, aunque (por la falta de computadores) suelen usarse únicamente aquellas que se muestran en la tabla 13.1. Fijémonos en que si $\lambda = 1$, no se realiza transformación alguna, y que para el caso de $\lambda = 0$, se tiene que $x^0 = 1$, por lo que se reemplaza en este caso por la transformación logarítmica.

$$\tilde{x}_{\lambda} = \begin{cases} x^{\lambda} & \lambda > 0 \\ \log(x) & \lambda = 0 \\ -(x^{\lambda}) & \lambda < 0 \end{cases} \tag{13.4}$$

λ	-2	-1	$-\frac{1}{2}$	0	$\frac{1}{2}$	1	2
$\tilde{x}$	$-\frac{1}{x^2}$	$-\frac{1}{x}$	$-\frac{1}{\sqrt{x}}$	$log(x)$	$\sqrt{x}$	x	x^2

Tabla 13.1: escalera de transformaciones de Tukey.

Usemos ahora la población total de Estados Unidos entre los años 1610 y 1850 (United States Census Bureau, 2004, 2021) como ejemplo para entender mejor esta transformación. La figura 13.4 (líneas 18–30 del script 13.3) muestra, a la izquierda, el histograma para la población (en millones de habitantes) y, a la derecha, un gráfico de dispersión para la población por año. Podemos ver claramente que la distribución de la población presenta una fuerte asimetría hacia la izquierda y que la población parece aumentar de manera exponencial durante ese periodo.

La figura 13.5 (líneas 46–75 del script 13.3) muestra los gráficos de dispersión para la población por año tras aplicar la transformación de Tukey con diferentes valores de λ a la población. En ella podemos observar que la curva cambia gradualmente de convexa a cóncava a medida que aumenta el valor de λ, lo que deja entrever que, para obtener un resultado que sea lo más cercano posible a una línea recta, tenemos que resolver un problema de optimización que minimice los valores residuales tras ajustar una recta a los puntos transformados. De las transformaciones presentadas en la figura 13.5, la más cercana a una recta se obtiene para $\lambda = 0$.

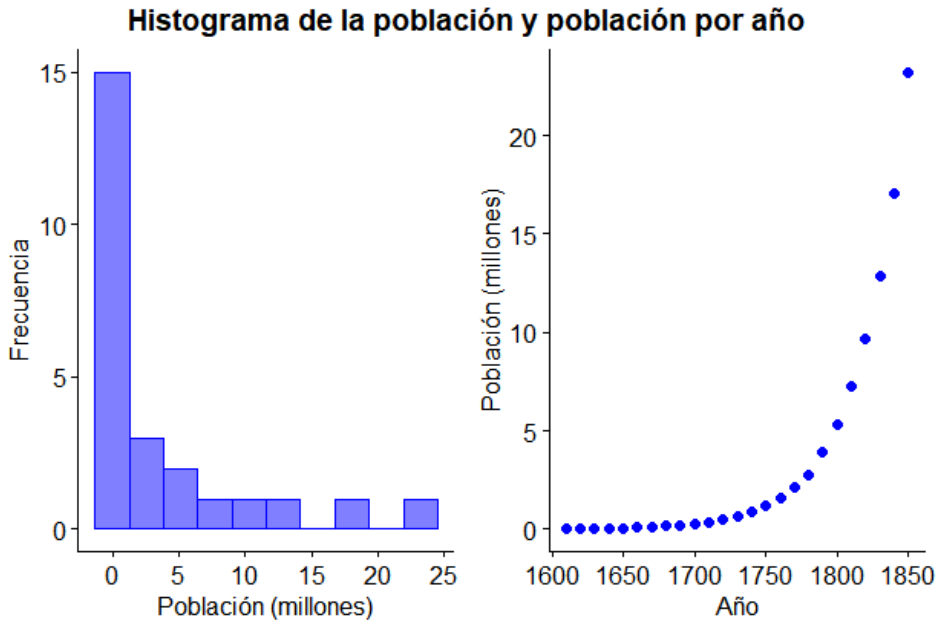


Figura 13.4: histograma de la población histórica de Estados Unidos y gráfico de dispersión de la población por año.

En párrafos anteriores mencionamos que la transformación de Tukey también permite reducir la asimetría en la distribución de los datos, como muestra la figura 13.6 (líneas 79–108 del script 13.3). En ella podemos notar que, a medida que λ aumenta, se reduce la asimetría negativa.

Como podemos suponer, reducir la asimetría nos ayuda a cumplir el requisito de normalidad que imponen muchas pruebas estadísticas, permitiéndonos así lograr resultados teóricamente más precisos. Sin embargo, una vez más tenemos que ser cuidadosos y tener en cuenta la transformación realizada al momento de interpretar los resultados. Si bien tenemos certeza que si se encuentran diferencias significativas en la variable transformada, estas diferencias **también existen en la variable original**, los estadísticos y los intervalos de confianza **¡no son los mismos** que arrojarían las pruebas con los datos originales!

En R, el paquete `rcompanion` incluye la función `transformTukey(x, start, end, int, plotit, verbose, quiet, statistic, returnLambda)`, donde:

- `x`: vector de valores a transformar.
- `start`: valor inicial de λ a evaluar.
- `end`: valor final de λ a evaluar.
- `int`: intervalo entre los valores de λ a evaluar.
- `plotit`: si toma valor `TRUE`, entrega los siguientes gráficos:
  - Estadístico de la prueba de normalidad versus λ.
  - Histograma de los valores transformados.
  - Gráfico Q-Q de los valores transformados.
- `verbose`: si toma valor `TRUE`, muestra información adicionar sobre la prueba de normalidad con respecto a λ.
- `quiet`: si toma valor `TRUE`, no muestra información alguna por pantalla.
- `statistic`: si toma valor 1, usa la prueba de normalidad de Shapiro-Wilk. Con valor 2, usa la prueba de Anderson-Darling.
- `returnLambda`: si toma valor `TRUE`, devuelve el valor de λ. Si toma valor `FALSE`, devuelve los datos transformados.

Tras llamar a la función `transformTukey()` con los datos de la población de Estados Unidos (script 13.3, líneas 111–112), obtenemos que el valor óptimo de λ es 0,12 y los gráficos de la figura 13.7. El gráfico 13.7a nos

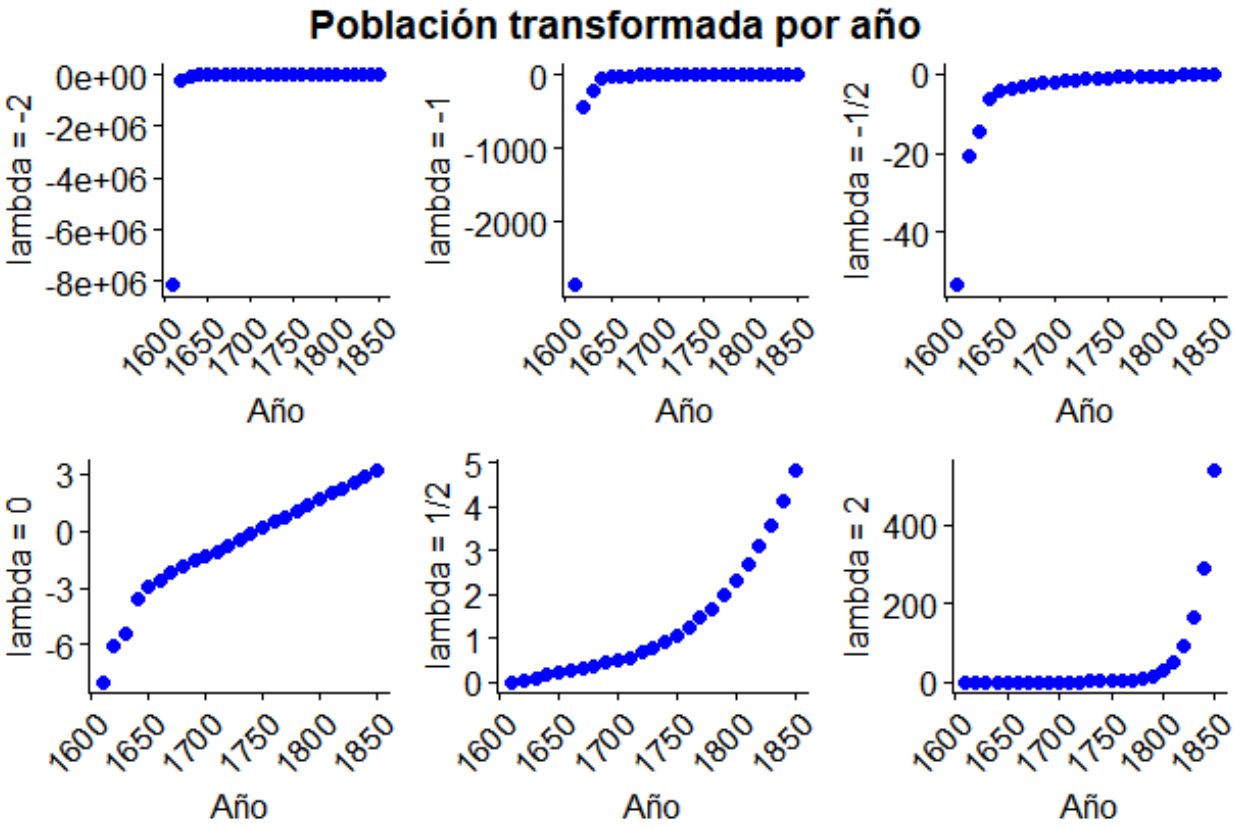


Figura 13.5: población de Estados Unidos por año tras aplicar la transformación de Tukey con distintos valores de λ.

muestra que el valor óptimo de λ es aquel que maximiza el estadístico entregado por la prueba de normalidad. A su vez, en la figura 13.7b vemos que la distribución obtenida con $\lambda = 0,12$ se asemeja mucho más a la normal que la de los datos originales o que cualquiera de las presentadas en la figura 13.6, lo que se ve confirmado por el gráfico Q-Q de la figura 13.7c.

Script 13.3: transformación de Tukey para la población total de Estados Unidos.

```
library(ggpubr)
library(rcompanion)

# Cargar datos
Year <- c(1610, 1620, 1630, 1640, 1650, 1660, 1670, 1680, 1690, 1700, 1710,
          1720, 1730, 1740, 1750, 1760, 1770, 1780, 1790, 1800, 1810, 1820,
          1830, 1840, 1850)

Population <- c(0.00035, 0.002302, 0.004646, 0.026634, 0.050368, 0.075058,
                0.111935, 0.151507, 0.210372, 0.250888, 0.331711, 0.466185,
                0.629445, 0.905563, 1.17076, 1.593625, 2.148076, 2.780369,
                3.929214, 5.308483, 7.239881, 9.638453, 12.86602, 17.069453,
                23.191876)

datos <- data.frame(Year, Population)

# Gráfico de dispersión e histograma.
g1 <- gghistogram(datos, x = "Population", bins = 10,
                  xlab = "Población (millones)", ylab = "Frecuencia",
```

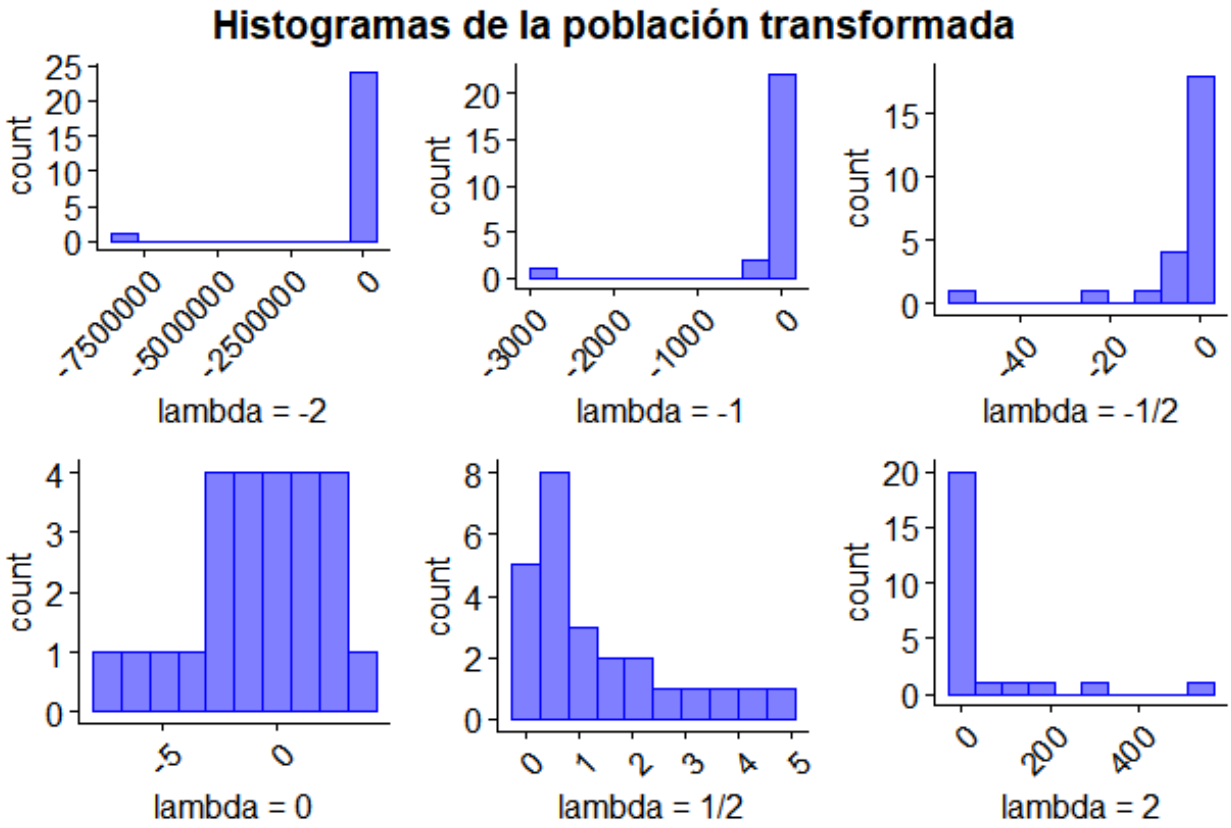
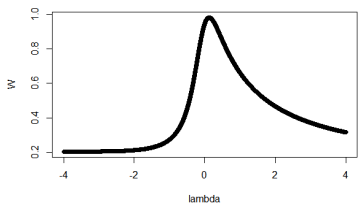
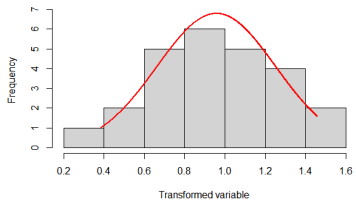


Figura 13.6: histograma de la población de Estados Unidos tras aplicar la transformación de Tukey con distintos valores de λ.

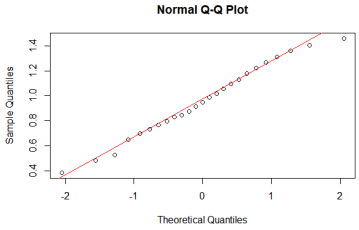
```
                    color = "blue", fill = "blue")

g2 <- ggscatter(datos, x = "Year", y = "Population", color = "blue",
                xlab = "Año", ylab = "Población (millones)")

# Histograma de la población y población por año
original  <- ggarrange(g1, g2, ncol = 2, nrow = 1)
texto <- "Histograma de la población y población por año"
titulo  <- text_grob(texto, face = "bold", size = 14)
original <- annotate_figure(original, top = titulo)
print(original)

# Transformaciones de la población
lambda_menos_dos <- -1 / (datos$Population ** 2)
lambda_menos_uno <- -1 / datos$Population
lambda_menos_un_medio <- -1 / sqrt(datos$Population)
lambda_cero <- log(datos$Population)
lambda_un_medio <- sqrt(datos$Population)
lambda_dos <- datos$Population ** 2

transformaciones <- data.frame(datos, lambda_menos_dos, lambda_menos_uno,
                               lambda_menos_un_medio, lambda_cero,
                               lambda_un_medio, lambda_dos)

# Gráficos de dispersión para la transformación de Tukey de la población y el
```



(a) estadístico W de la prueba de Shapiro-Wilk por cada valor de λ.



(b) histograma de la población transformada con el valor óptimo de λ.



(c) gráfico Q-Q de la población transformada con el valor óptimo de λ.

Figura 13.7: gráficos entregados por `transformTukey()`.

```
# año, usando distintos valores de lambda.
gt1 <- ggscatter(transformaciones, x = "Year", y = "lambda_menos_dos",
                 color = "blue", xlab = "Año",
                 ylab = "lambda = -2") + rotate_x_text(45)

gt2 <- ggscatter(transformaciones, x = "Year", y = "lambda_menos_uno",
                 color = "blue", xlab = "Año",
                 ylab = "lambda = -1") + rotate_x_text(45)

gt3 <- ggscatter(transformaciones, x = "Year", y = "lambda_menos_un_medio",
                 color = "blue", xlab = "Año",
                 ylab = "lambda = -1/2") + rotate_x_text(45)

gt4 <- ggscatter(transformaciones, x = "Year", y = "lambda_cero",
                 color = "blue", xlab = "Año",
                 ylab = "lambda = 0") + rotate_x_text(45)

gt5 <- ggscatter(transformaciones, x = "Year", y = "lambda_un_medio",
                 color = "blue", xlab = "Año",
                 ylab = "lambda = 1/2") + rotate_x_text(45)

gt6 <- ggscatter(transformaciones, x = "Year", y = "lambda_dos",
                 color = "blue", xlab = "Año",
                 ylab = "lambda = 2") + rotate_x_text(45)

# Crear una única figura con todos los gráficos de dispersión.
dispersion  <- ggarrange(gt1, gt2, gt3, gt4, gt5, gt6, ncol = 3, nrow = 2)
texto <- "Población transformada por año"
titulo  <- text_grob(texto, face = "bold", size = 14)
dispersion <- annotate_figure(dispersion, top = titulo)
print(dispersion)

# Histogramas para la transformación de Tukey de la población y el año,
# usando distintos valores de lambda.
h1 <- gghistogram(transformaciones, bins = 10, x = "lambda_menos_dos",
                  color = "blue", fill = "blue",
                  xlab = "lambda = -2") + rotate_x_text(45)

h2 <- gghistogram(transformaciones, bins = 10, x = "lambda_menos_uno",
                  color = "blue", fill = "blue",
                  xlab = "lambda = -1") + rotate_x_text(45)

h3 <- gghistogram(transformaciones, bins = 10, x = "lambda_menos_un_medio",
```

```
                    color = "blue", fill = "blue",
                    xlab = "lambda = -1/2") + rotate_x_text(45)

h4 <- gghistogram(transformaciones, bins = 10, x = "lambda_cero",
                    color = "blue", fill = "blue",
                    xlab = "lambda = 0") + rotate_x_text(45)

h5 <- gghistogram(transformaciones, bins = 10, x = "lambda_un_medio",
                    color = "blue", fill = "blue",
                    xlab = "lambda = 1/2") + rotate_x_text(45)

h6 <- gghistogram(transformaciones, bins = 10, x = "lambda_dos",
                    color = "blue", fill = "blue",
                    xlab = "lambda = 2") + rotate_x_text(45)

# Crear una única figura con todos los gráficos de dispersión.
histograma  <- ggarrange(h1, h2, h3, h4, h5, h6, ncol = 3, nrow = 2)
texto <- "Histogramas de la población transformada"
titulo  <- text_grob(texto, face = "bold", size = 14)
histograma <- annotate_figure(histograma, top = titulo)
print(histograma)

# Buscar la mejor transformación de Tukey usando una función de R.
transformacion <- transformTukey(datos$Population, start = -4, end = 4,
                                 int = 0.001, returnLambda = TRUE)
```

13.1.4 Transformaciones Box-Cox

La **transformación Box-Cox** es una versión escalada de la transformación de Tukey, dada por la ecuación 13.5:

$$x'_\lambda = \frac{x^\lambda - 1}{\lambda} \tag{13.5}$$

Si bien a primera vista parece muy diferente a la ecuación 13.4, podemos reescribirla como:

$$x'_\lambda = \frac{x^\lambda - 1}{\lambda} \approx \frac{\left(1 + \lambda \log(x) + \frac{1}{2}\lambda^2 \log(x)^2 + \ldots\right) - 1}{\lambda}$$

De donde:

$$\lim_{\lambda \to 0} \frac{\left(1 + \lambda \log(x) + \frac{1}{2}\lambda^2 \log(x)^2 + \ldots\right) - 1}{\lambda} = \frac{0}{0}$$

Tras aplicar la regla de l'Hôpital, obtenemos finalmente que:

$$\lim_{\lambda \to 0} x'_\lambda = \log(x)$$

Por lo que, al igual que en la escalera de potencias de Tukey, pero de forma más natural, empleamos la transformación logarítmica para $\lambda = 0$.

La figura 13.8 (creada con el script 13.4, líneas 39–64) muestra los gráficos de dispersión para la población total de Estados Unidos por año tras aplicar la transformación de Box-Cox con diferentes valores de λ. Podemos ver que el resultado se parece al que obtuvimos con la transformación de Tukey (figura 13.5).

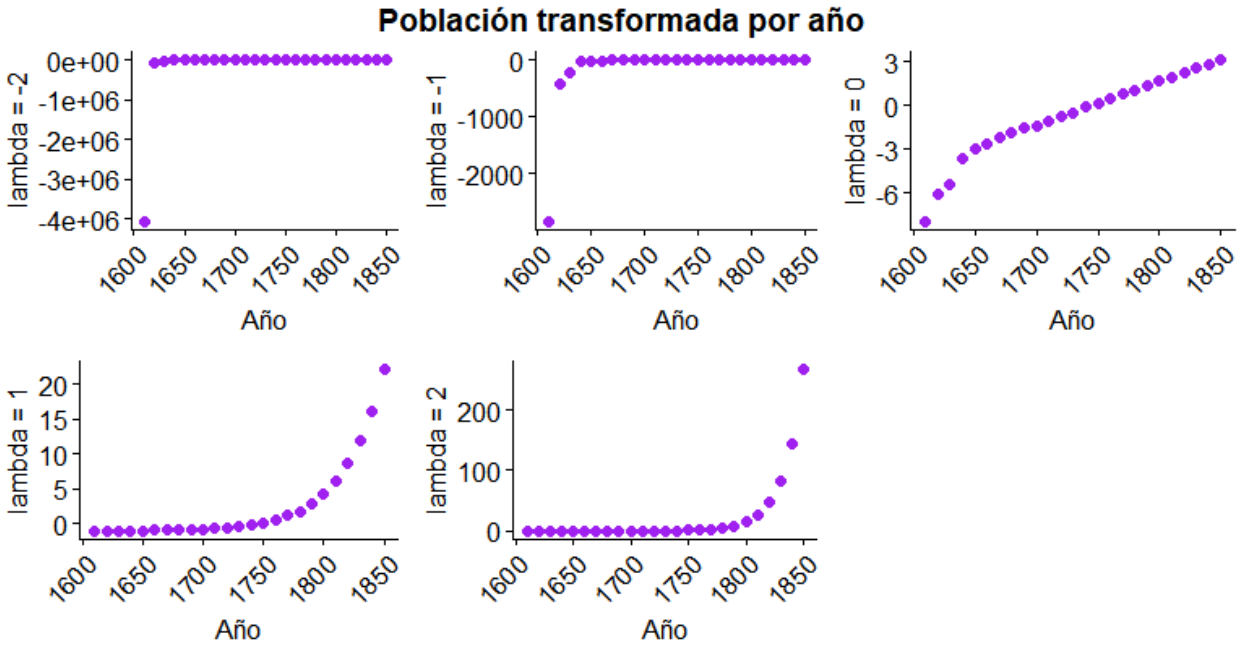


Figura 13.8: población de Estados Unidos por año tras aplicar la transformación de Box-Cox con distintos valores de λ.

Una característica interesante de esta transformación es que, para cualquier valor de λ, $x'_\lambda = 0$ cuando $x = 1$. Podemos observar esto claramente en la figura 13.9, que compara transformaciones Box-Cox con distintos valores de λ con $\log(x)$.

El paquete `DescTools` de R incluye varias funciones que permiten efectuar la transformación Box-Cox (Carchedi y col., s.f.). Destacan entre ellas:

- `BoxCoxLambda(x, lower, upper)`: devuelve el valor óptimo de λ para la transformación Box-Cox del vector `x`.
- `BoxCox(x, lambda)`: devuelve un vector correspondiente a la transformación Box-Cox de `x` con parámetro `lambda`.
- `BoxCoxInv(x, lambda)`: revierte la transformación Box-Cox del vector `x` con parámetro `lambda`.

Donde:

- `x`: vector numérico.
- `lower`: limite inferior para los posibles valores de λ.
- `upper`: limite superior para los posibles valores de λ.
- `lambda`: parámetro de la transformación.

En las líneas 67–70 del script 13.4 se determina el valor óptimo del parámetro λ, obteniéndose como resultado $\lambda = 0,09942656$, para luego efectuar la transformación Box-Cox correspondiente de la población de Estados Unidos. La figura 13.10 (script 13.4, líneas 84–87) muestra gráficamente el resultado de la transformación. En el gráfico 13.10a podemos ver que la relación entre la población transformada y el año se asemeja a una recta, mientras que el histograma de la figura 13.10b se parece bastante a una distribución normal, hecho que vemos confirmado por el gráfico Q-Q de la figura 13.10c.

Script 13.4: transformación de Box-Cox para la población total de Estados Unidos.

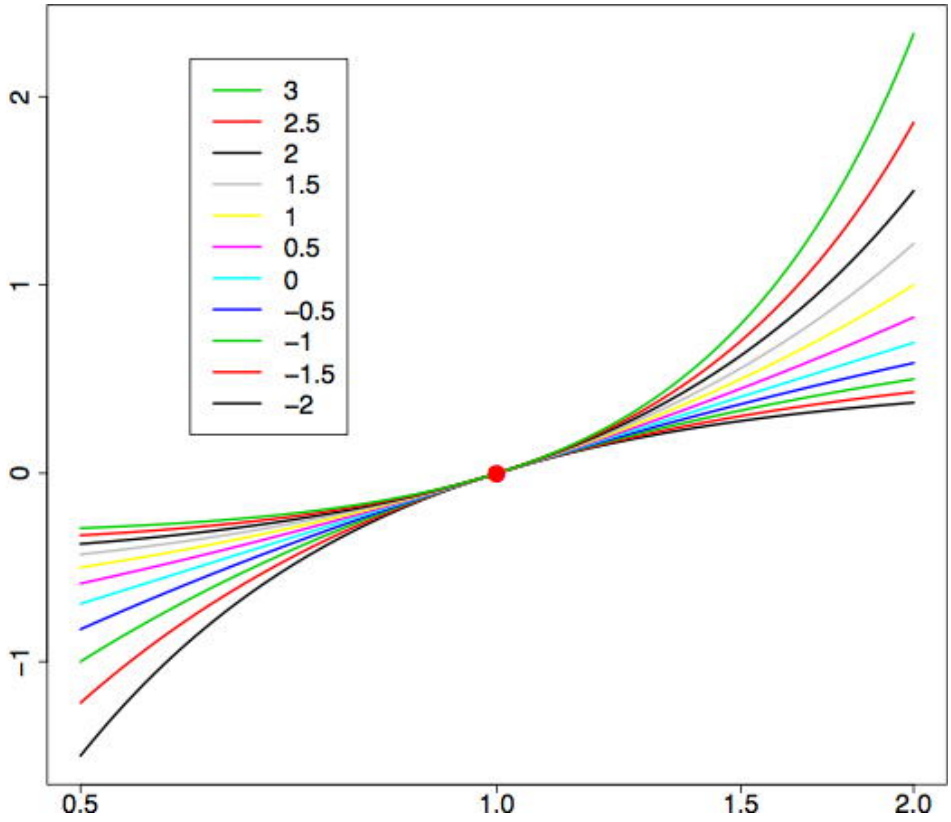
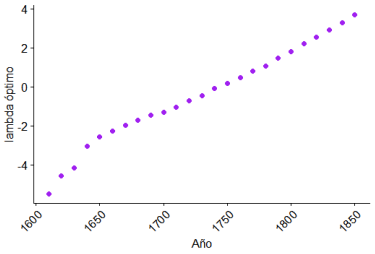
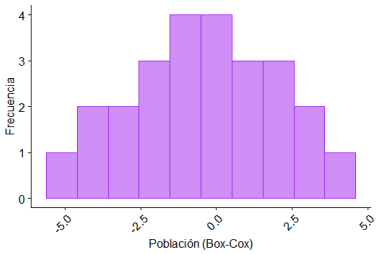


Figura 13.9: ejemplos de la transformación Box-Cox versus $\log(x)$. Fuente: (Lane, s.f., p. 16).

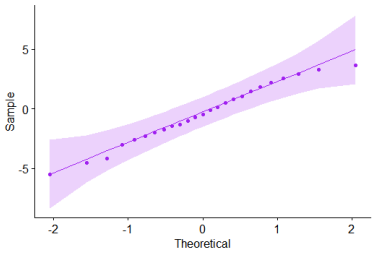
```r
library(ggpubr)
library(DescTools)

# Cargar datos
Year <- c(1610, 1620, 1630, 1640, 1650, 1660, 1670, 1680, 1690, 1700, 1710,
          1720, 1730, 1740, 1750, 1760, 1770, 1780, 1790, 1800, 1810, 1820,
          1830, 1840, 1850)

Population <- c(0.00035, 0.002302, 0.004646, 0.026634, 0.050368, 0.075058,
                0.111935, 0.151507, 0.210372, 0.250888, 0.331711, 0.466185,
                0.629445, 0.905563, 1.17076, 1.593625, 2.148076, 2.780369,
                3.929214, 5.308483, 7.239881, 9.638453, 12.86602, 17.069453,
                23.191876)

datos <- data.frame(Year, Population)

# Transformación de Box-cox
box_cox <- function(x, lambda) {
  if(lambda == 0) {
    return(log(x))
  }

  resultado <- (x ** lambda -1) / lambda
  return(resultado)
}

# Transformaciones de la población
```



(a) población de Estados Unidos por año tras aplicar la transformación de Box-Cox con λ óptimo.



(b) histograma de la población transformada con el valor óptimo de λ.



(c) gráfico Q-Q de la población transformada con el valor óptimo de λ.

Figura 13.10: gráficos de población de Estados Unidos por año usando la transformación de Box-Cox.

```r
lambda_menos_dos <- box_cox(datos$Population, -2)
lambda_menos_uno <- box_cox(datos$Population, -1)
lambda_cero <- box_cox(datos$Population, 0)
lambda_uno <- box_cox(datos$Population, 1)
lambda_dos <- box_cox(datos$Population, 2)

transformaciones <- data.frame(datos, lambda_menos_dos, lambda_menos_uno,
                               lambda_cero, lambda_uno, lambda_dos)

# Gráficos de dispersión para la transformación de Box-Cox de la población y
# el año, usando distintos valores de lambda.
gt1 <- ggscatter(transformaciones, x = "Year", y = "lambda_menos_dos",
                 color = "purple", xlab = "Año",
                 ylab = "lambda = -2") + rotate_x_text(45)

gt2 <- ggscatter(transformaciones, x = "Year", y = "lambda_menos_uno",
                 color = "purple", xlab = "Año",
                 ylab = "lambda = -1") + rotate_x_text(45)

gt3 <- ggscatter(transformaciones, x = "Year", y = "lambda_cero",
                 color = "purple", xlab = "Año",
                 ylab = "lambda = 0") + rotate_x_text(45)

gt4 <- ggscatter(transformaciones, x = "Year", y = "lambda_uno",
                 color = "purple", xlab = "Año",
                 ylab = "lambda = 1") + rotate_x_text(45)

gt5 <- ggscatter(transformaciones, x = "Year", y = "lambda_dos",
                 color = "purple", xlab = "Año",
                 ylab = "lambda = 2") + rotate_x_text(45)

# Crear una única figura con todos los gráficos de dispersión.
dispersion  <- ggarrange(gt1, gt2, gt3, gt4, gt5, ncol = 3, nrow = 2)
texto <- "Población transformada por año"
titulo  <- text_grob(texto, face = "bold", size = 14)
dispersion <- annotate_figure(dispersion, top = titulo)
print(dispersion)

# Buscar la mejor transformación Box-Cox usando funciones de R.
lambda <- BoxCoxLambda(datos$Population, lower = -4, upper = 4)
cat("Lambda óptimo:", lambda)
transformacion <- BoxCox(datos$Population, lambda)
```

```r
datos <- data.frame(datos, transformacion)

# Graficar los datos transformados.
g1 <- ggqqplot(transformacion, color = "purple")
print(g1)

g2 <- gghistogram(datos, bins = 10, x = "transformacion", color = "purple",
                  fill = "purple", xlab = "Población (Box-Cox)",
                  ylab = "Frecuencia") + rotate_x_text(45)

print(g2)

# Gráfico de dispersión para la transformación de Box-Cox de la población y
# el año, usando lambda óptimo.
g3 <- ggscatter(datos, x = "Year", y = "transformacion", color = "purple",
                xlab = "Año", ylab = "lambda óptimo") + rotate_x_text(45)

print(g3)
```

13.2 MÉTODOS ROBUSTOS

Hemos mencionado varias veces en este libro que muchas de las pruebas estadísticas clásicas requieren que los datos sigan una distribución cercana a la normal, pero que es frecuente que los datos disponibles no cumplan esta u otras condiciones.

Pensemos como ejemplo en la prueba t de Student (capitulo 5), que se usa para inferir acerca de la media de una población. Sin embargo, como mencionamos en el capítulo 2, esta medida de tendencia central tiene el problema de ser sensible a la presencia de valores atípicos, a distribuciones asimétricas o a muestras muy pequeñas. En términos generales, el incumplimiento de las condiciones del supuesto de normalidad puede causar diversos problemas:

- Resultados sesgados.
- Intervalos de confianza calculados de manera inadecuada.
- Reducción del poder estadístico de la prueba.

Hemos visto que muchas veces las alternativas no paramétricas resultan útiles cuando las muestras son pequeñas y presentan distribuciones asimétricas, aunque estas pruebas igualmente requieren verificar ciertas condiciones.

Otra alternativa es usar métodos de remuestreo para establecer la distribución muestral y poder emplear métodos paramétricos. Sin embargo, sabemos que el remuestreo suele ser muy costoso en términos de la cantidad de cálculos realizados.

La sección anterior, por su parte, nos ofrece la opción de aplicar ciertas transformaciones a de transformaciones a los datos de modo que se asemejen más a la distribución normal, suponiendo que el tamaño de la(s) muestra(s) sea adecuado. No obstante, este enfoque tiene la dificultad de que, tras la transformación, en realidad estamos infiriendo acerca de un nuevo parámetro, lo cual podría alejarse significativamente de la pregunta de investigación original. ¡Recordemos que al aplicar la transformación logarítmica, por ejemplo, la prueba t de Student infiere sobre la media geométrica en lugar de la media aritmética!

En el capítulo 2 mencionamos la existencia de estimadores robustos, poco sensibles a asimetrías muestrales o valores atípicos. No obstante, el paradigma estadístico tradicional no suele considerarlos. En esta sección, basada en las ideas expuestas por Mair y Wilcox (2020), aborda, en consecuencia, alternativas robustas para muchas de las pruebas estudiadas hasta ahora, disponibles en el paquete `WRS2` de R.

13.2.1 Alternativas robustas para la media

En el capítulo 2 conocimos distintas medidas de tendencia central. Entre ellas vimos que la mediana, correspondiente al valor central (o el promedio de los dos valores centrales) de la muestra ordenada, es una alternativa robusta a la media. No obstante, existen otras opciones que nos pueden ser útiles.

13.2.1.1 Media truncada

La **media truncada** es bastante similar a la media aritmética que ya conocemos, con la diferencia de que se calcula descartando un determinado porcentaje (γ) de los valores en ambos extremos (colas) de la distribución. Tomemos como ejemplo una muestra x con 10 elementos, los cuales han sido ordenados por simplicidad:

$$x = [1, 4, 37, 38, 40, 43, 43, 45, 87, 91]$$

Si calculamos la media para la muestra anterior, tenemos que es $\overline{x} = 42,9$. No obstante, si observamos la muestra del ejemplo con detención, podemos darnos cuenta de que los valores extremos parecen ser atípicos y, en consecuencia, pueden tener una gran influencia en el valor resultante para la media. Así, podría ser más adecuado calcular la media truncada con $\gamma = 0,2$, es decir, podando el 20 % de los valores más pequeños y el 20 % de los valores más grandes, con lo que obtendremos:

$$\overline{x}_t = \frac{37 + 38 + 40 + 43 + 43 + 45}{6} = 41$$

Así, si $\gamma = 0,5$, se obtiene la mediana.

En R, podemos calcular la media truncada mediante la ya conocida función `mean()` del paquete base, agregando el argumento adicional `trim` con la proporción γ de los datos extremos a descartar, esto es `mean(x, trim = 0.2)` para el ejemplo.

13.2.1.2 Media Winsorizada

Un problema de la media truncada es que, al usarla, descartamos muchos datos, lo que puede causar problemas en algunos casos. Otra opción puede ser, en lugar de descartar los valores extremos en cada cola, reemplazarlos por los valores extremos que no serían descartados al usar la media truncada y luego calcular la media con la muestra modificada. A esta medida se le conoce como **media Winsorizada**. Si retomamos nuestro ejemplo para la media truncada, los valores extremos tras la operación de truncado son 37 y 45. Así, reemplazamos los valores truncados en la muestra original por estos nuevos extremos, con lo que nuestra muestra Winsorizada es:

$$x = [37, 37, 37, 38, 40, 43, 43, 45, 45, 45]$$

Con lo que la media Winsorizada es:

$$\overline{x}_W = \frac{37+37+37+38+40+43+43+45+45+45}{10} = 41$$

En R, podemos hacer este cálculo mediante la función `winmean(x, tr)` del paquete `WRS2`, donde:

- `x`: vector con la muestra original.
- `tr`: proporción de los datos en cada cola a Winsorizar.

Así, la llamada para el ejemplo sería `winmean(x, tr = 0.2)`.

13.2.2 Prueba de Yuen para dos muestras independientes

La **prueba de Yuen** es una buena alternativa a la prueba t de Student para muestras independientes cuando las varianzas de ambas muestras son muy diferentes o los tamaños de las muestras son muy dispares. Utiliza las medias truncadas y las medias Winsorizadas, aunque no se recomienda usar esta prueba si las muestras se truncan cerca del nivel de medianas ($\gamma \approx 0,5$).

Cuando tenemos dos muestras independientes, el estadístico de prueba está dado por la ecuación 13.6, donde $\overline{x}_{ti}$ son las medias truncadas de cada muestra.

$$T_y = \frac{\overline{x}_{t1} - \overline{x}_{t2}}{\sqrt{d_1 + d_2}} \tag{13.6}$$

El denominador de la ecuación 13.6 corresponde al error estándar, donde d_i se calcula como señala la ecuación 13.7, con:

- s_{w1}: desviación estándar de la muestra Winsorizada.
- n_i: tamaño de la muestra original.
- h_i: tamaño de la muestra truncada.

$$d_i = \frac{(n_i - 1)s_{wi}^2}{h_i(h_i - 1)} \tag{13.7}$$

El estadístico T_y sigue una distribución t cuyos grados de libertad se calculan mediante la ecuación 13.8.

$$\nu_y = \frac{(d_1 + d_2)^2}{\frac{d_1^2}{h_1 - 1} + \frac{d_2^2}{h_2 - 1}} \tag{13.8}$$

A su vez, la ecuación 13.9 muestra cómo se construye el intervalo de confianza, donde t corresponde al cuantil $1 - \alpha/2$ de la distribución t con ν_y grados de libertad.

$$(\overline{x}_{t1} - \overline{x}_{t2}) \pm t\sqrt{d_1 + d_2} \tag{13.9}$$

Para una prueba de hipótesis bilateral, las hipótesis son:

H_0: $\mu_{t1} = \mu_{t2}$
H_A: $\mu_{t1} \neq \mu_{t2}$

En R, podemos aplicar la prueba de Yuen para muestras independientes mediante la función `yuen(formula, data, tr)` del paquete `WRS2`, donde:

- `formula`: tiene la forma <variable dependiente>∼<variable independiente>. Note que la variable independiente debe tener dos niveles, a fin de determinar a qué muestra pertenece cada observación de la variable dependiente.
- `data`: matriz de datos.
- `tr`: parámetro γ de la poda.

Para pruebas unilaterales, sin embargo, se recomienda usar la variante con bootstrap, implementada en la función `yuen(formula, data, tr, nboot)`, donde `nboot` señala la cantidad de muestras a obtener mediante bootstrapping.

El script 13.5 muestra un ejemplo de uso de la prueba de Yuen. Como contexto, se desea comparar el tiempo promedio de ejecución (en milisegundos) de dos algoritmos. Se han seleccionado aleatoriamente 70 instancias de igual tamaño del problema, las cuales han sido asignadas al azar a cada uno de los algoritmos. En consecuencia, contamos con $n_a = 40$ observaciones para el algoritmo A y $n_b = 30$ observaciones para el algoritmo B. Se ha establecido para este estudio un nivel de significación $\alpha = 0,05$.

Las líneas 19–20 del script 13.5 construyen gráficos Q-Q para comprobar el supuesto de normalidad de la prueba t de Student, obteniéndose como resultado la figura 13.11, donde podemos observar que las muestras obtenidas no se distribuyen normalmente, especialmente para el caso del algoritmo A.

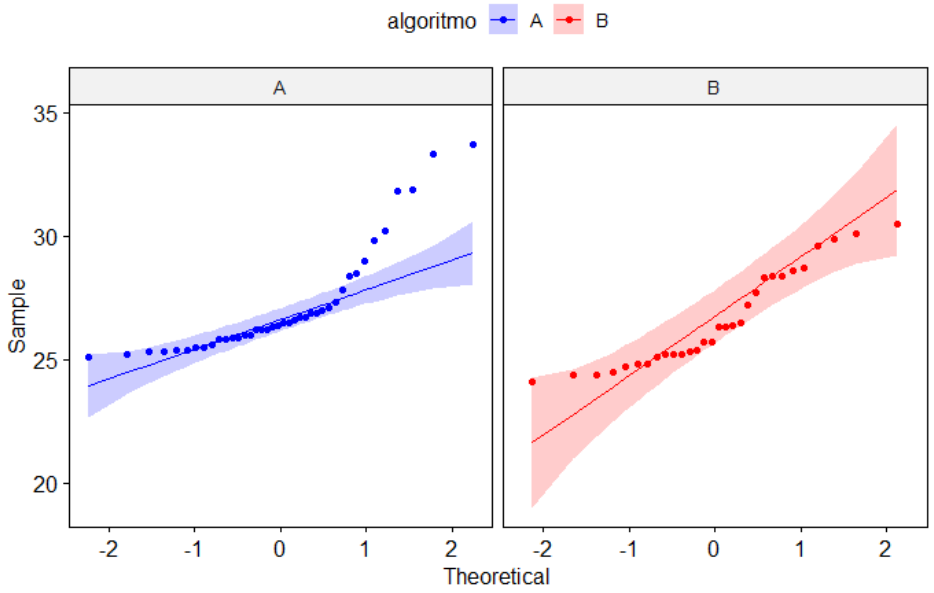


Figura 13.11: gráfico Q-Q de las muestras originales.

Para ilustrar los conceptos asociados a la prueba de Yuen, las líneas 28–45 del script 13.5 truncan ambas muestras considerando $\gamma = 0,2$ y construyen gráficos Q-Q para las muestras podadas (figura 13.12). Podemos apreciar que, tras la poda, la distribución de los datos se aproxima más a la normal.

Finalmente, las líneas 48–49 del script 13.5 efectúan la prueba de Yuen para ambas muestras, obteniéndose como resultado (figura 13.13) una diferencia entre las medias truncadas de $0,246$, con intervalo de 95 % de confianza $(-0,859; 1,351)$ y tamaño del efecto de $0,090$. El valor p obtenido, $p = 0,653$, no es significativo al nivel de significación establecido, por lo que concluimos con 95 % de confianza que ambos algoritmos tienen, en promedio, igual tiempo de ejecución.

Script 13.5: prueba de Yuen para dos muestras independientes.

```
1 library(WRS2)
2 library(ggpubr)
3
```

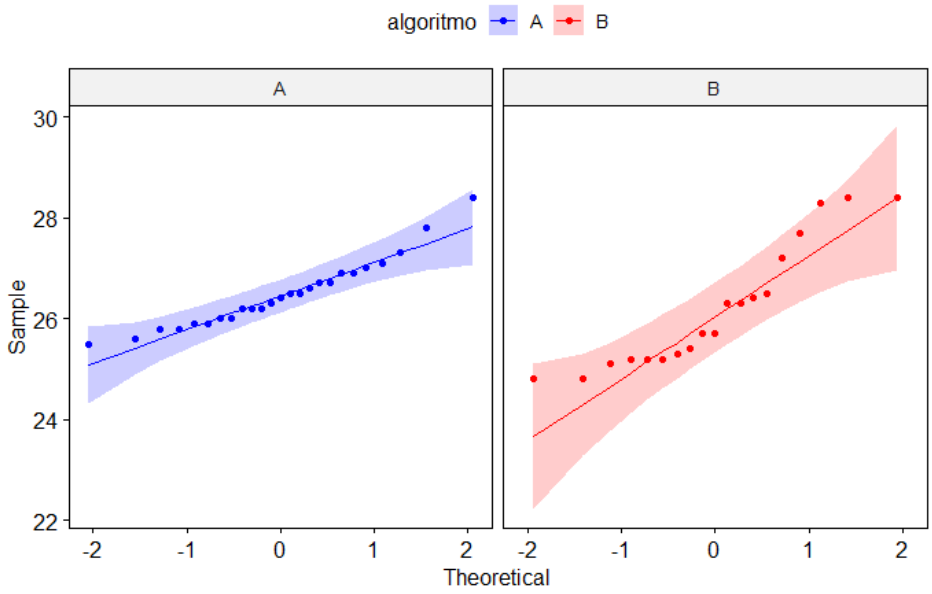


Figura 13.12: gráfico Q-Q de las muestras truncadas.

```
Call:
yuen(formula = tiempo ~ algoritmo, data = datos, tr = gamma)

Test statistic: 0.455 (df = 29.05), p-value = 0.65252

Trimmed mean difference:  0.24583
95 percent confidence interval:
-0.8592     1.3509

Explanatory measure of effect size: 0.09
```

Figura 13.13: resultado de la prueba de Yuen para el ejemplo.

```r
# Construir data frame.
a <- c(25.1, 25.2, 25.3, 25.3, 25.4, 25.4, 25.5, 25.5, 25.6, 25.8, 25.8,
       25.9, 25.9, 26.0, 26.0, 26.2, 26.2, 26.2, 26.3, 26.4, 26.5, 26.5,
       26.6, 26.7, 26.7, 26.9, 26.9, 27.0, 27.1, 27.3, 27.8, 28.4, 28.5,
       29.0, 29.8, 30.2, 31.8, 31.9, 33.3, 33.7)

b <- c(24.1, 24.4, 24.4, 24.5, 24.7, 24.8, 24.8, 25.1, 25.2, 25.2, 25.2,
       25.3, 25.4, 25.7, 25.7, 26.3, 26.3, 26.4, 26.5, 27.2, 27.7, 28.3,
       28.4, 28.4, 28.6, 28.7, 29.6, 29.9, 30.1, 30.5)

tiempo <- c(a, b)
algoritmo <- c(rep("A", length(a)), rep("B", length(b)))
datos <- data.frame(tiempo, algoritmo)

# Comprobar normalidad.
g <- ggqqplot(datos, x = "tiempo", facet.by = "algoritmo",
              palette = c("blue", "red"), color = "algoritmo")

print(g)
```

```r

# Establecer nivel de significación.
alfa <- 0.05

# Ver poda del 20%.
gamma <- 0.2
n_a <- length(a)
n_b <- length(b)

poda_a <- n_a * gamma
poda_b <- n_b * gamma

a_truncada <- a[poda_a:(n_a - poda_a)]
b_truncada <- b[poda_b:(n_b - poda_b)]

tiempo <- c(a_truncada, b_truncada)
algoritmo <- c(rep("A", length(a_truncada)), rep("B", length(b_truncada)))
datos_truncados <- data.frame(tiempo, algoritmo)

g <- ggqqplot(datos_truncados, x = "tiempo", facet.by = "algoritmo",
              palette = c("blue", "red"), color = "algoritmo")

print(g)

# Aplicar prueba de Yuen.
prueba <- yuen(tiempo ~ algoritmo, data = datos, tr = gamma)
print(prueba)
```

El paquete `WRS2` incluye también la función `pb2gen(formula, data, est, nboot)`, que usa bootstrapping para aplicar la prueba de Yuen usando otras medidas robustas de tendencia central, donde:

- `formula`: tiene la misma forma descrita para la prueba de Yuen.
- `data`: matriz de datos.
- `est`: medida a emplear. Puede tomar las opciones `"mean"` para la media y `"median"` (mediana), entre otras opciones que escapan a los alcances de este curso.
- `nboot`: cantidad de muestras a generar mediante bootstrapping.

El script 13.6 muestra cómo aplicar la prueba de Yuen para el ejemplo, usando como estimadores la media y la mediana, obteniéndose los resultados de la figura 13.14. Podemos ver que, en ambos casos, el valor p obtenido es más alto que al usar la media podada.

Script 13.6: prueba de Yuen con bootstrapping para dos muestras independientes usando la media y la mediana.

```r
library(WRS2)

# Construir data frame.
a <- c(25.1, 25.2, 25.3, 25.3, 25.4, 25.4, 25.5, 25.5, 25.6, 25.8, 25.8,
       25.9, 25.9, 26.0, 26.0, 26.2, 26.2, 26.2, 26.3, 26.4, 26.5, 26.5,
       26.6, 26.7, 26.7, 26.9, 26.9, 27.0, 27.1, 27.3, 27.8, 28.4, 28.5,
       29.0, 29.8, 30.2, 31.8, 31.9, 33.3, 33.7)

b <- c(24.1, 24.4, 24.4, 24.5, 24.7, 24.8, 24.8, 25.1, 25.2, 25.2, 25.2,
       25.3, 25.4, 25.7, 25.7, 26.3, 26.3, 26.4, 26.5, 27.2, 27.7, 28.3,
       28.4, 28.4, 28.6, 28.7, 29.6, 29.9, 30.1, 30.5)

tiempo <- c(a, b)
algoritmo <- c(rep("A", length(a)), rep("B", length(b)))
datos <- data.frame(tiempo, algoritmo)

```

```
Resultado al usar la media como estimador

Call:
pb2gen(formula = tiempo ~ algoritmo, data = datos, est = "mean",
    nboot = bootstrap)

Test statistic: 0.61, p-value = 0.21321
95% confidence interval:
-0.3008    1.5617

Resultado al usar la mediana como estimador

Call:
pb2gen(formula = tiempo ~ algoritmo, data = datos, est = "median",
    nboot = bootstrap)

Test statistic: 0.45, p-value = 0.47147
95% confidence interval:
-0.95    1.35
```

Figura 13.14: resultado de la prueba de Yuen con bootstrapping para el ejemplo, usando como estimadores la media y la mediana.

```r
# Establecer nivel de significación y cantidad de muestras a generar
# con bootstrapping.
alfa <- 0.05
bootstrap <- 999

# Aplicar prueba con la media
set.seed(135)

prueba_media <- pb2gen(tiempo ~ algoritmo,
                       data = datos,
                       est = "mean",
                       nboot = bootstrap)

cat("\n\nResultado al usar la media como estimador\n\n")
print(prueba_media)

# Aplicar prueba con la mediana
set.seed(135)

prueba_mediana <- pb2gen(tiempo ~ algoritmo,
                         data = datos,
                         est = "median",
                         nboot = bootstrap)

cat("Resultado al usar la mediana como estimador\n\n")
print(prueba_mediana)
```

13.2.3 Prueba de Yuen para dos muestras pareadas

Para el caso de dos muestras pareadas, podemos generalizar el estadístico de la prueba de Yuen con medias truncadas como muestra la ecuación 13.10, donde el nuevo término d_{12} está dado por la ecuación 13.11. En este caso, ambas muestras tienen igual tamaño n y h corresponde al tamaño de la muestra combinada tras la poda.

$$T_y = \frac{\overline{x}_{t1} - \overline{x}_{t2}}{\sqrt{d_1 + d_2 - 2 \cdot d_{12}}} \tag{13.10}$$

$$d_{12} = \frac{1}{h(h-1)} \sum_{i=1}^{n} (x_{i1} - \overline{x}_1) \cdot (x_{i2} - \overline{x}_2) \tag{13.11}$$

Supongamos ahora que queremos comparar el rendimiento de dos algoritmos X e Y, para lo cual hemos seleccionado aleatoriamente 25 instancias del problema y registrado su tiempo de ejecución en milisegundos con cada uno de los algoritmos. El script 13.7 ilustra el uso de la función `yuend(x, y, tr)` del paquete `WRS2`, que compara las medias truncadas, obteniéndose, para este ejemplo, el resultado que muestra la figura 13.15.

```
Call:
yuend(x = x, y = y, tr = gamma)

Test statistic: -4.5915 (df = 14), p-value = 0.00042

Trimmed mean difference:  -0.76
95 percent confidence interval:
-1.115     -0.405

Explanatory measure of effect size: 0.44
```

Figura 13.15: resultado de la prueba de Yuen para el ejemplo, usando como estimadores la media y la mediana.

Script 13.7: prueba de Yuen para dos muestras pareadas.

```
library(WRS2)

# Construir data frame.
x <- c(32.0, 32.0, 32.0, 32.0, 32.1, 32.1, 32.1, 32.2, 32.3, 32.3, 32.5,
       32.7, 32.7, 32.7, 33.1, 33.4, 33.9, 34.1, 34.2, 34.5, 36.0, 36.6,
       36.7, 37.2, 38.0)

y <- c(33.0, 33.0, 33.0, 33.0, 33.0, 33.0, 33.3, 33.3, 33.3, 33.3, 33.5,
       33.6, 33.7, 33.9, 33.9, 34.2, 34.2, 34.3, 34.3, 34.4, 34.5, 34.6,
       36.4, 38.9, 40.2)

# Fijar nivel de significación.
alfa <- 0.05

# Aplicar prueba de Yuen para muestras pareadas.
gamma <- 0.2
prueba <- yuend(x = x, y = y, tr = gamma)
print(prueba)
```

Puesto que el valor p obtenido, $p < 0,00042$, es menor que el nivel de significación, la evidencia es suficientemente fuerte como para rechazar la hipótesis nula en favor de la hipótesis alternativa. En consecuencia,

podemos afirmar con 95 % de confianza que existe una diferencia estadísticamente significativa en el desempeño de ambos algoritmos, siendo el algoritmo X el más eficiente (puesto que la diferencia estimada entre las medias tiene signo negativo).

13.2.4 Comparaciones de una vía para múltiples grupos independientes

El paquete `WRS2` ofrece diferentes alternativas a ANOVA de una vía para muestras independientes que podemos usar cuando los tamaños muestrales son muy diferentes o no se cumple la condición de homocedasticidad.

La función `t1way(formula, data, tr, alpha)` efectúa un procedimiento similar a ANOVA usando medias truncadas. A su vez, la función `lincon(formula, data, tr, alpha)` permite realizar el procedimiento post-hoc correspondiente.

De manera similar, `t1waybt(formula, data, tr, nboot)` realiza un procedimiento análogo al anterior incorporando bootstrapping. En este caso, el procedimiento post-hoc puede realizarse mediante la función `mcppb20(formula, data, tr, nboot)`.

Una tercera opción es la función `med1way(formula, data, iter)`, que emplea la mediana y sigue un proceso iterativo. No obstante, en este caso el paquete no ofrece funciones que permitan realizar el procedimiento post-hoc.

Los argumentos asociados a las funciones mencionadas en los párrafos anteriores son:

- `formula`: de la forma <variable dependiente>∼<variable independiente>.
- `data`: matriz de datos.
- `tr`: parámetro γ de la poda.
- `alpha`: nivel de significación.
- `nboot`: cantidad de muestras a generar mediante bootstrapping.
- `iter`: cantidad de iteraciones a realizar.

El script 13.8 ilustra el funcionamiento de algunas de las funciones descritas en los párrafos precedentes, suponiendo que ahora deseamos comparar el tiempo promedio de ejecución (en milisegundos) de tres algoritmos, contando con $n_a = 40$ observaciones para el algoritmo A, $n_b = 30$ observaciones para el algoritmo B y $n_c = 35$ observaciones para el algoritmo C. Se ha establecido para este estudio un nivel de significación $\alpha = 0,05$. Podemos ver en las figuras 13.16 y 13.17 que las funciones `t1way()` y `t1waybt()` arrojan el mismo resultado. Puesto que el valor p obtenido, $p = 0,00021$, es menor que el nivel de significación, rechazamos la hipótesis nula en favor de la hipótesis alternativa. Concluimos, entonces, con 95 % de confianza, que existe una diferencia estadísticamente significativa entre los tiempos promedio de ejecución de los algoritmos.

Al efectuar los procedimientos post-hoc respectivos, los valores p obtenidos son ligeramente diferentes para ambos métodos (figuras 13.16 y 13.17). No obstante, en ambos casos podemos concluir que el algoritmo C presenta un tiempo de ejecución promedio diferente.

Si examinamos las medias de cada grupo, tenemos que $\overline{x}_A = 27,19$ [ms], $\overline{x}_B = 26,58$ [ms] y $\overline{x}_C = 25,52$ [ms], por lo que el algoritmo C es más rápido.

Script 13.8: alternativas robustas para comparar entre múltiples grupos independientes.

```
library(WRS2)

# Construir data frame.
a <- c(25.1, 25.2, 25.3, 25.3, 25.4, 25.4, 25.5, 25.5, 25.6, 25.8, 25.8,
       25.9, 25.9, 26.0, 26.0, 26.2, 26.2, 26.2, 26.3, 26.4, 26.5, 26.5,
       26.6, 26.7, 26.7, 26.9, 26.9, 27.0, 27.1, 27.3, 27.8, 28.4, 28.5,
       29.0, 29.8, 30.2, 31.8, 31.9, 33.3, 33.7)
```

```
Comparación entre grupos usando medias truncadas

Call:
t1way(formula = tiempo ~ algoritmo, data = datos, tr = gamma,
    alpha = alfa)

Test statistic: F = 10.9813
Degrees of freedom 1: 2
Degrees of freedom 2: 34.39
p-value: 0.00021

Explanatory measure of effect size: 0.48
Bootstrap CI: [0.28; 0.68]


Procedimiento post-hoc

Call:
lincon(formula = tiempo ~ algoritmo, data = datos, tr = gamma,
    alpha = alfa)

         psihat ci.lower ci.upper p.value
A vs. B 0.24583 -1.11867  1.61033 0.65252
A vs. C 1.49583  0.65571  2.33596 0.00022
B vs. C 1.25000 -0.02757  2.52757 0.03909
```

Figura 13.16: resultado de la comparación entre múltiples grupos independientes usando medias truncadas.

```r

b <- c(24.1, 24.4, 24.4, 24.5, 24.7, 24.8, 24.8, 25.1, 25.2, 25.2, 25.2,
       25.3, 25.4, 25.7, 25.7, 26.3, 26.3, 26.4, 26.5, 27.2, 27.7, 28.3,
       28.4, 28.4, 28.6, 28.7, 29.6, 29.9, 30.1, 30.5)

c <- c(24.5, 24.5, 24.5, 24.5, 24.5, 24.5, 24.6, 24.6, 24.6, 24.6, 24.6,
       24.6, 24.7, 24.7, 24.7, 24.7, 24.8, 25.0, 25.0, 25.0, 25.2, 25.2,
       25.2, 25.2, 25.5, 25.7, 25.9, 26.2, 26.5, 26.5, 26.7, 27.0, 29.2,
       29.9, 30.1)

tiempo <- c(a, b, c)
algoritmo <- c(rep("A", length(a)), rep("B", length(b)), rep("C", length(c)))
datos <- data.frame(tiempo, algoritmo)

# Fijar nivel de significación.
alfa <- 0.05

# Comparar los diferentes algoritmos usando medias truncadas.
cat("Comparación entre grupos usando medias truncadas\n\n")
gamma <- 0.2

set.seed(666)

medias_truncadas <- t1way(tiempo ~ algoritmo, data = datos, tr = gamma,
                          alpha = alfa)

print(medias_truncadas)
```

```
Comparación entre grupos usando bootstrap

Call:
t1way(formula = tiempo ~ algoritmo, data = datos, tr = gamma,
    alpha = alfa)

Test statistic: F = 10.9813
Degrees of freedom 1: 2
Degrees of freedom 2: 34.39
p-value: 0.00021

Explanatory measure of effect size: 0.48
Bootstrap CI: [0.28; 0.68]


Procedimiento post-hoc

Call:
mcppb20(formula = tiempo ~ algoritmo, data = datos, tr = gamma,
    nboot = muestras)

         psihat ci.lower ci.upper p-value
A vs. B 0.24583 -0.91667  1.61389 0.55656
A vs. C 1.49583  0.73690  2.44464 0.00000
B vs. C 1.25000  0.16270  2.34206 0.00801
```

Figura 13.17: resultado de la comparación entre múltiples grupos independientes usando medias truncadas con bootstrapping.

```r

if(medias_truncadas$p.value < alfa) {
  cat("\nProcedimiento post-hoc\n\n")

  set.seed(666)

  post_hoc <- lincon(tiempo ~ algoritmo, data = datos, tr = gamma,
                     alpha = alfa)

  print(post_hoc)
}

# Comparar los diferentes algoritmos usando bootstrap.
cat("Comparación entre grupos usando bootstrap\n\n")
muestras <- 999

set.seed(666)

bootstrap <- t1waybt(tiempo ~ algoritmo, data = datos, tr = gamma,
                     nboot = muestras)

print(medias_truncadas)

if(medias_truncadas$p.value < alfa) {
  cat("\nProcedimiento post-hoc\n\n")

```

```
  set.seed(666)

  post_hoc <- mcppb20(tiempo ~ algoritmo, data = datos, tr = gamma,
                      nboot = muestras)

  print(post_hoc)
}
```

13.2.5 Comparaciones de una vía para múltiples grupos correlacionados

Desde luego, el paquete `WRS2` también ofrece opciones robustas para reemplazar el procedimiento ANOVA de una vía para muestras correlacionadas, que podemos usar cuando los datos disponibles violan la condición de esfericidad.

La función `rmanova(y, groups, blocks, tr)` efectúa un procedimiento similar a ANOVA usando medias truncadas, mientras que la función `rmmcp(y, groups, blocks, tr, alpha)` implementa el procedimiento post-hoc para dicha prueba. Por otra parte, `rmanovab(y, groups, blocks, tr, nboot)` realiza la misma tarea que `rmanova()`, incorporando bootstrapping. En este caso, el procedimiento post-hoc está dado por la función `pairdepb(y, groups, blocks, tr, nboot)`. Los argumentos para esta familia de funciones son:

- `formula`: de la forma <variable dependiente>∼<variable independiente>.
- `y`: vector con la variable dependiente.
- `groups`: vector que indica los grupos.
- `blocks`: vector que identifica los sujetos o bloques.
- `tr`: parámetro γ de la poda.
- `alpha`: nivel de significación.
- `nboot`: cantidad de muestras a generar mediante bootstrapping.

El script 13.9 muestra el uso de las funciones robustas sin bootstrapping para ANOVA de una vía con muestras correlacionadas. El ejemplo presentado aborda, una vez más, la comparación del desempeño de tres algoritmos, X, Y y Z. Para ello, se han seleccionado aleatoriamente 25 instancias del problema y se registra su tiempo de ejecución (en milisegundos) con cada uno de los algoritmos. Para este estudio consideraremos un nivel de significación $\alpha = 0,05$. Podemos ver los resultados obtenidos en la figura 13.18.

El valor p resultante, $p = 1 \cdot 10^{-5}$, indica que la evidencia es suficientemente fuerte para rechazar la hipótesis nula en favor de la hipótesis alternativa, por lo que realizamos el procedimiento post-hoc correspondiente. La conclusión, con 95 % de confianza, es que no todos los algoritmos tienen el mismo rendimiento promedio, siendo el algoritmo X más eficiente que los algoritmos Y y Z.

Script 13.9: alternativa robusta para comparar entre múltiples grupos correlacionados.

```
library(WRS2)
library(tidyverse)

# Construir data frame.
X <- c(32.0, 32.0, 32.0, 32.0, 32.1, 32.1, 32.1, 32.2, 32.3, 32.3, 32.5,
       32.7, 32.7, 32.7, 33.1, 33.4, 33.9, 34.1, 34.2, 34.5, 36.0, 36.6,
       36.7, 37.2, 38.0)

Y <- c(33.0, 33.0, 33.0, 33.0, 33.0, 33.0, 33.3, 33.3, 33.3, 33.3, 33.5,
       33.6, 33.7, 33.9, 33.9, 34.2, 34.2, 34.3, 34.3, 34.4, 34.5, 34.6,
       36.4, 38.9, 40.2)

Z <- c(32.0, 32.2, 32.5, 32.6, 32.7, 32.7, 32.7, 33.0, 33.2, 33.4, 33.6,
```

```
Call:
rmanova(y = tiempo, groups = algoritmo, blocks = instancia, tr = gamma)

Test statistic: F = 24.1706
Degrees of freedom 1: 1.5
Degrees of freedom 2: 20.96
p-value: 1e-05


Procedimiento post-hoc

Call:
rmmcp(y = tiempo, groups = algoritmo, blocks = instancia, tr = gamma,
    alpha = alfa)

          psihat ci.lower ci.upper p.value p.crit   sig
X vs. Y -0.85333 -1.16837 -0.53830 0.00000 0.0169  TRUE
X vs. Z -0.68667 -0.98245 -0.39089 0.00002 0.0250  TRUE
Y vs. Z -0.00667 -0.26776  0.25443 0.94566 0.0500 FALSE
```

Figura 13.18: resultado de las alternativa robusta para comparar entre múltiples grupos correlacionados.

```
14        33.6, 33.9, 34.1, 34.2, 34.4, 34.4, 34.5, 34.6, 34.7, 36.3, 36.6,
15        36.7, 38.9, 39.2)
16
17 instancia <- 1:length(X)
18 datos <- data.frame(instancia, X, Y, Z)
19
20 # Llevar data frame a formato largo.
21 datos <- datos %>% pivot_longer(c("X", "Y", "Z"), names_to = "algoritmo",
22                                 values_to = "tiempo")
23
24 datos[["algoritmo"]] <- factor(datos[["algoritmo"]])
25
26 # Fijar nivel de significación.
27 alfa <- 0.05
28
29 # Aplicar alternativa robusta para ANOVA de una vía con
30 # muestras correlacionadas.
31 gamma <- 0.2
32
33 prueba <- rmanova(y = datos[["tiempo"]], groups = datos[["algoritmo"]],
34                   blocks = datos[["instancia"]], tr = gamma)
35
36 print(prueba)
37
38 if(prueba$p.value < alfa) {
39   cat("\nProcedimiento post-hoc\n\n")
40
41   post_hoc <- rmmcp(y = datos[["tiempo"]], groups = datos[["algoritmo"]],
42                     blocks = datos[["instancia"]], tr = gamma, alpha = alfa)
43
44   print(post_hoc)
45 }
```

13.3 EJERCICIOS PROPUESTOS

1. ¿Para qué se usa la transformación logarítmica?
2. Explica por qué comparar medias aritméticas de datos en escala logarítmica compara las medias geométricas en la escala normal de la variable.
3. El paquete `rcompanion` proporciona una función para aplicar la escala de potencias de Tukey. Experimenta con su uso.
4. Explica la relación entre la escala de potencias de Tukey y la transformación Box-Cox.
5. El paquete `DescTools` proporciona funciones para aplicar la transformación Box-Cox. Experimenta con su uso.
6. En tus palabras, ¿qué es un estadístico robusto?
7. Menciona tres situaciones que complican las pruebas de hipótesis paramétricas tradicionales.
8. ¿Qué alternativas se mencionan para tratar datos problemáticos? En particular, ¿qué recomienda el capítulo para muestras pequeñas que muestran desviaciones de normalidad?
9. Explica, en tus propias palabras, las dos medidas de tendencia central robustas presentadas en este capítulo.
10. ¿Cómo funciona y para qué sirve la prueba de Yuen?
11. ¿Se pueden comparar dos medianas en vez de dos medias de grupos independientes?
12. Describe las funciones disponibles en el paquete `WRS2` para hacer análisis de varianza robusta (con medias y con medianas, e incluyendo procedimientos post-hoc) para grupos independientes.
13. ¿Existe una alternativa para medidas repetidas de la prueba de Yuen?
14. ¿Existe una alternativa robusta para hacer un análisis de varianza con medidas repetidas?

REFERENCIAS

Carchedi, N., De Mesmaeker, D. & Vannoorenberghe, L. (s.f.). RDocumentation. Consultado el 2 de abril de 2021, desde https://www.rdocumentation.org/

Glen, S. (2021). *Geometric Mean Definition and Formula*. Consultado el 27 de mayo de 2021, desde https://www.statisticshowto.com/geometric-mean-2/

Lane, D. (s.f.). *Online Statistics Education: A Multimedia Course of Study*. Consultado el 4 de mayo de 2021, desde https://onlinestatbook.com/

Mair, P. & Wilcox, R. (2020). Robust statistical methods in R using the WRS2 package. *Behavior Research Methods*, *52*(2), 464-488.

Rousseeuw, P. J. & Leroy, A. M. (1987). *Robust regression and outlier detection*. John wiley & sons.

United States Census Bureau. (2004). *CT1970p2-13: Colonial and Pre-Federal Statistics*. Consultado el 26 de mayo de 2021, desde https://www2.census.gov/prod2/statcomp/documents/CT1970p2-13.pdf

United States Census Bureau. (2021). Decennial Census of Population and Housing. Consultado el 26 de mayo de 2021, desde https://www.census.gov/programs-surveys/decennial-census/decade.html