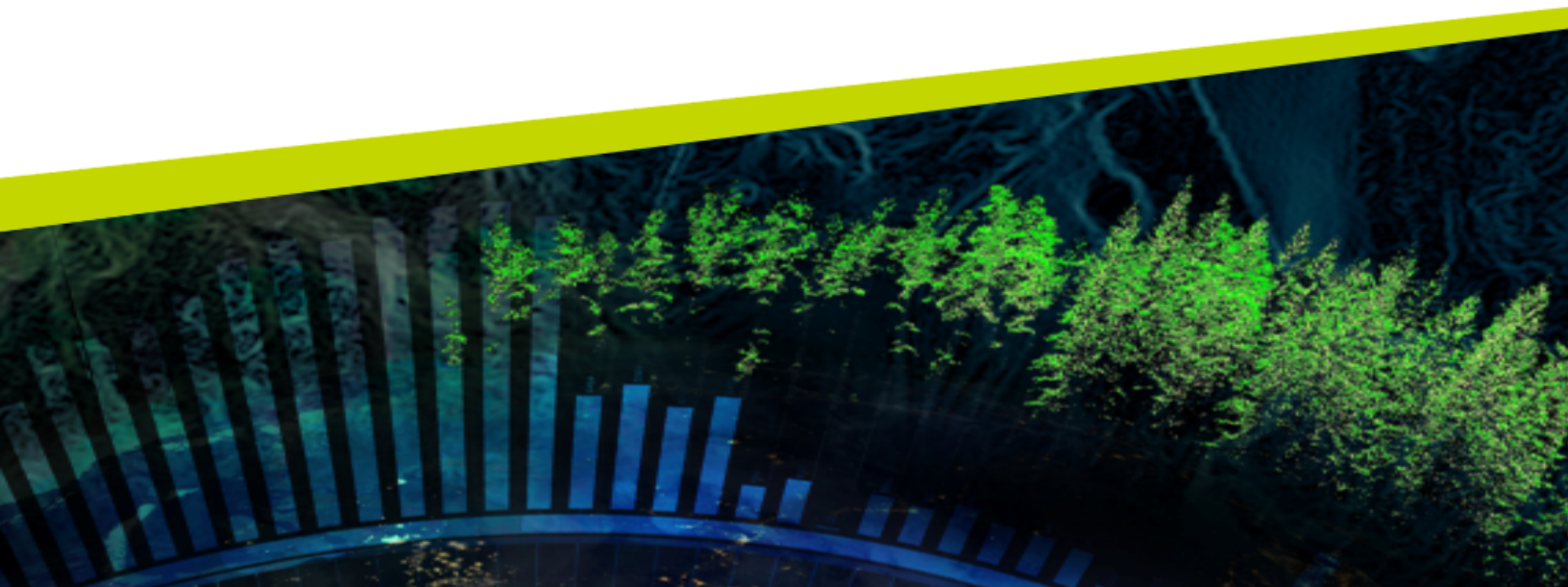
INFERENCIA Y MODELOS ESTADÍSTICOS

Jacqueline Köhler C. y José Luis Jara V.



CAPÍTULO 11. MÉTODOS MÁS CONTEMPORÁNEOS PARA ENFRENTAR 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 7 y 10) o transformar los datos a otras escalas (10). Nos hemos referido a estos métodos como "clásicos", puesto que fueron desarrollados hace décadas, cuando no existía acceso a los computadores. y que se han utilizado y enseñado por décadas.

El aumento en el acceso y el poder de cómputo, como en todas las áreas del saber, ha permitido el desarrollo de métodos más sofisticados para analizar datos problemáticos. En este capítulo abordaremos algunas de las estrategias que podemos usar en estos casos.

11.1 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 sugieren que esta, u otras condiciones, se estén cumpliendo.

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.

El capítulo anterior, nos ofrece dos opciones. La primera es aplicar ciertas 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! La segunda opción que vimos es hacer uso de pruebas no paramétricas que resultan útiles cuando las muestras son pequeñas y presentan distribuciones asimétricas, aunque estas pruebas igualmente requieren verificar ciertas condiciones.

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.

11.1.1 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.

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.

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)`.

Si bien hay otras medidas de tendencia central, como los estimadores-M basados en la máxima verosimilitud, en este capítulo solo trabajaremos con las dos opciones vistas anteriormente.

11.1.2 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 11.1, donde $\overline{x}_{ti}$ son las medias truncadas de cada muestra.

$$T_y = \frac{\overline{x}_{t1} - \overline{x}_{t2}}{\sqrt{d_1 + d_2}} \tag{11.1}$$

El denominador de la ecuación 11.1 corresponde al error estándar, donde d_i se calcula como señala la ecuación 11.2, con:

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

$$d_i = \frac{(n_i - 1)s_{wi}^2}{h_i(h_i - 1)} \tag{11.2}$$

El estadístico T_y sigue una distribución t cuyos grados de libertad se calculan mediante la ecuación 11.3.

$$\nu_y = \frac{(d_1 + d_2)^2}{\frac{d_1^2}{h_1 - 1} + \frac{d_2^2}{h_2 - 1}} \tag{11.3}$$

A su vez, la ecuación 11.4 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.

$$(\overline{x}_{t1} - \overline{x}_{t2}) \pm t\sqrt{d_1 + d_2} \tag{11.4}$$

Para una prueba de hipótesis bilateral, las hipótesis son:

Denotando como μ_{ti} a las medias truncadas de las poblaciones que dan origen a cada muestra trabajada, entonces:
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 que usa *técnicas de remuestreo*, que serán el tópico de la siguiente sección, implementada en la función `yuen(formula, data, tr, nboot)`, donde `nboot` señala la cantidad de submuestras a obtener mediante bootstrapping.

El script 11.1 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 19–22 del script 11.1 construyen gráficos Q-Q para comprobar el supuesto de normalidad requerido por la prueba t de Student, obteniéndose como resultado la figura 11.1, donde podemos observar que las muestras obtenidas no se distribuyen normalmente, especialmente para el caso del algoritmo A.

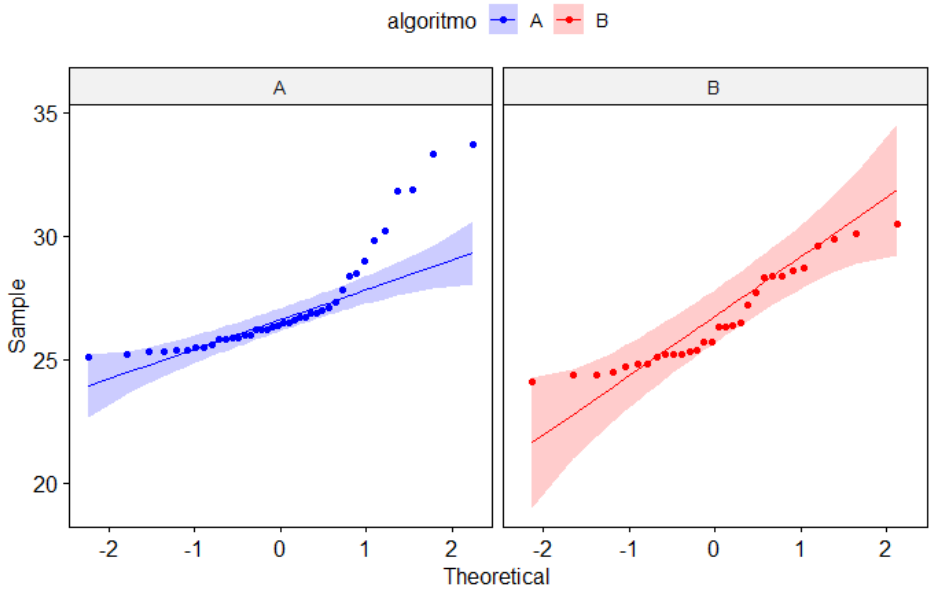


Figura 11.1: gráfico Q-Q de las muestras originales.

Para ilustrar los conceptos asociados a la prueba de Yuen, las líneas 28–45 del script 11.1 truncan ambas muestras considerando $\gamma = 0,2$ y construyen gráficos Q-Q para las muestras podadas (figura 11.2). Podemos apreciar que, tras la poda, la distribución de los datos se aproxima más a la normal.

Finalmente, las líneas 48–49 del script 11.1 efectúan la prueba de Yuen para ambas muestras, obteniéndose como resultado (figura 11.3) 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 no es posible descartar que ambos algoritmos tienen, en promedio, igual tiempo de ejecución.

Script 11.1: prueba de Yuen para dos muestras independientes.

```
1  library(WRS2)
2  library(ggpubr)
3
4  # Construir data frame.
5  a <- c(25.1, 25.2, 25.3, 25.3, 25.4, 25.4, 25.5, 25.5, 25.6, 25.8, 25.8,
6         25.9, 25.9, 26.0, 26.0, 26.2, 26.2, 26.2, 26.3, 26.4, 26.5, 26.5,
7         26.6, 26.7, 26.7, 26.9, 26.9, 27.0, 27.1, 27.3, 27.8, 28.4, 28.5,
8         29.0, 29.8, 30.2, 31.8, 31.9, 33.3, 33.7)
9
10 b <- c(24.1, 24.4, 24.4, 24.5, 24.7, 24.8, 24.8, 25.1, 25.2, 25.2, 25.2,
11        25.3, 25.4, 25.7, 25.7, 26.3, 26.3, 26.4, 26.5, 27.2, 27.7, 28.3,
12        28.4, 28.4, 28.6, 28.7, 29.6, 29.9, 30.1, 30.5)
13
```

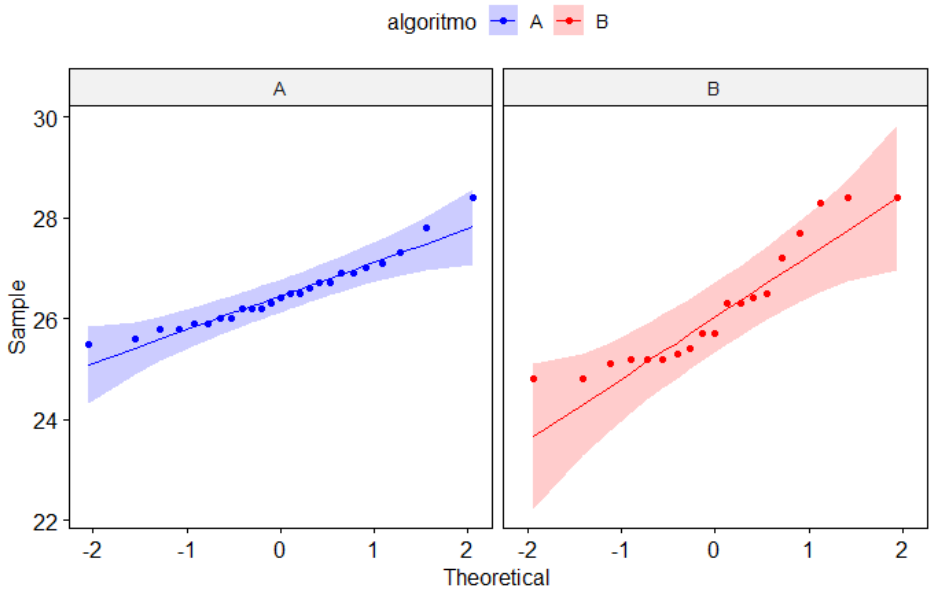


Figura 11.2: gráfico Q-Q de las muestras truncadas.

```
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 11.3: resultado de la prueba de Yuen para el ejemplo.

```
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)

# 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
```

```r
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 remuestreo con 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 11.2 muestra cómo aplicar la prueba de Yuen para el ejemplo, usando como estimadores la media y la mediana, obteniéndose los resultados de la figura 11.4. Podemos ver que, en ambos casos, el valor p obtenido es más bajo que al usar la media podada.

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

Script 11.2: 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)

# 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)
```

11.1.3 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 11.5, donde el nuevo término $d_{1,2}$ está dado por la ecuación 11.6. En este caso, ambas muestras tienen igual tamaño n y h corresponde al tamaño de la muestra combinada tras la poda.

$$T_y = \frac{\overline{x}_{t1} - \overline{x}_{t2}}{\sqrt{d_1 + d_2 - 2 \cdot d_{1,2}}} \tag{11.5}$$

$$d_{1,2} = \frac{1}{h(h-1)} \sum_{i=1}^{n} (x_{i1} - \overline{x}_1) \cdot (x_{i2} - \overline{x}_2) \tag{11.6}$$

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 11.3 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 11.5.

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

Script 11.3: 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, $p < 0,001$, 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).

11.1.4 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 remuestreo con 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 11.4 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 11.6 y 11.7 que las funciones `t1way()` y `t1waybt()` arrojan el mismo resultado. Puesto que el valor p obtenido, $p < 0,001$, 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 11.6 y 11.7). 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 11.4: 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)

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)

```

```
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 11.6: resultado de la comparación entre múltiples grupos independientes usando medias truncadas.

```r
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)

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)
```

```
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 11.7: resultado de la comparación entre múltiples grupos independientes usando medias truncadas con bootstrapping.

```r
}

# 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)
}
```

11.1.5 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 remuestreo con 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 11.5 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 11.8.

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.

```
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 11.8: resultado de las alternativa robusta para comparar entre múltiples grupos correlacionados.

Script 11.5: alternativa robusta para comparar entre múltiples grupos correlacionados.

```r
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,
       33.6, 33.9, 34.1, 34.2, 34.4, 34.4, 34.5, 34.6, 34.7, 36.3, 36.6,
       36.7, 38.9, 39.2)

instancia <- 1:length(X)
datos <- data.frame(instancia, X, Y, Z)

# Llevar data frame a formato largo.
datos <- datos %>% pivot_longer(c("X", "Y", "Z"), names_to = "algoritmo",
                                values_to = "tiempo")

datos[["algoritmo"]] <- factor(datos[["algoritmo"]])

# Fijar nivel de significación.
alfa <- 0.05

# Aplicar alternativa robusta para ANOVA de una vía con
# muestras correlacionadas.
gamma <- 0.2

prueba <- rmanova(y = datos[["tiempo"]], groups = datos[["algoritmo"]],
                  blocks = datos[["instancia"]], tr = gamma)

print(prueba)

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

  post_hoc <- rmmcp(y = datos[["tiempo"]], groups = datos[["algoritmo"]],
                    blocks = datos[["instancia"]], tr = gamma, alpha = alfa)

  print(post_hoc)
}
```

11.2 REMUESTREO

Los **métodos basados en remuestreo** son técnicas que construyen una distribución empírica de un estadístico a partir de un conjunto de observaciones, con el fin de cuantificar la incertidumbre de la estimación de un parámetro y mejorar su exactitud. Son una buena alternativa a emplear cuando necesitamos inferir sobre parámetros distintos a la media o la proporción, o bien cuando no se cumplen las condiciones requeridas por las pruebas ya conocidas. Además, algunos de estos métodos son más precisos que los tradicionales. Pese

a estas ventajas, los métodos basados en remuestreo realizan enormes cantidades de cómputos, por lo que en la práctica requieren de herramientas de software para su aplicación. Si bien existen varios métodos de remuestreo paramétricos y semiparamétricos, en este capítulo abordaremos las dos principales técnicas de remuestro no paramétricas, basándonos en las ideas descritas por Amat Rodrigo (2016) y Hesterberg y col. (2003).

11.3 BOOTSTRAPPING

A partir de lo que hemos aprendido hasta ahora, ya tenemos bastante claro que, en estadística, el ideal es contar con varias muestras grandes. Pero muchas veces solo disponemos de una muestra, relativamente pequeña. Sin embargo, si esta muestra es representativa de la población, esperaríamos que las observaciones que ella contiene aparecieran con frecuencias similares a las de la población. El método de **bootstrapping** se construye en torno a esta idea y, en términos generales, sigue los siguientes pasos:

1. Crear una gran cantidad B de nuevas muestras (cientos o miles) a partir de la muestra original. Cada muestra debe tener el mismo tamaño que la original y se construye mediante **muestreo con reposición**. Esto quiere decir que, al seleccionar un elemento de la muestra original, se devuelve a ella antes de tomar el siguiente, por lo que podría ser reelegido.
2. Calcular el estadístico de interés para cada una de las muestras, obteniendo así la llamada **distribución bootstrap** del estadístico.
3. Usar la distribución bootstrap, la cual entrega información acerca de la forma, el centro y la variabilidad de la distribución muestral del estadístico de interés.

A los lectores atentos les habrá llamado la atención que, a diferencia de las pruebas y procedimientos anteriores, el segundo paso del método de bootstrapping habla de un **estadístico de interés** en lugar de la media o la proporción (como las pruebas estudiadas hasta ahora). Esto se debe a que, en general, puede aplicarse para casi **cualquier estadístico**.

Esta técnica, además de contrastar hipótesis, permite construir intervalos de confianza para el parámetro estimado de la población.

11.3.1 Bootstrapping para una muestra

Supongamos que la investigadora Helen Chufe desea evaluar un nuevo algoritmo de clasificación y determinar el tiempo promedio de ejecución (en milisegundos) para instancias de tamaño fijo del problema. Para ello ha realizado pruebas con 10 instancias del problema y registrado los tiempos de ejecución, presentados en la tabla 11.1. La figura 11.9 muestra la distribución del tiempo de ejecución para la muestra.

Instancia	1	2	3	4	5	6	7	8	9	10
Tiempo (ms)	79	75	84	75	94	82	76	90	79	88

Tabla 11.1: tiempo de ejecución para cada instancia de la muestra.

Evidentemente, la muestra es pequeña ($n = 10$) y su distribución está fuertemente desviada hacia la izquierda, por lo que Chufe ha decidido emplear bootstrapping como alternativa para enfrentar estos datos problemáticos.

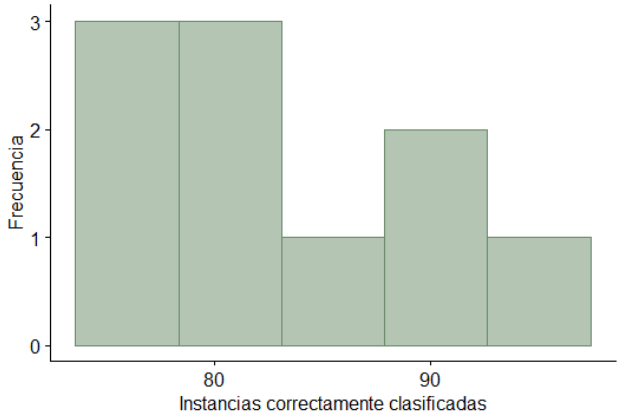


Figura 11.9: distribución del tiempo de ejecución para la muestra.

Para ilustrar el proceso paso a paso, consideremos inicialmente $B = 10$ remuestreos y calculemos la media para cada uno. La tabla 11.2 muestra en cada columna una de las muestras obtenidas, con sus respectivas medias en la última fila.

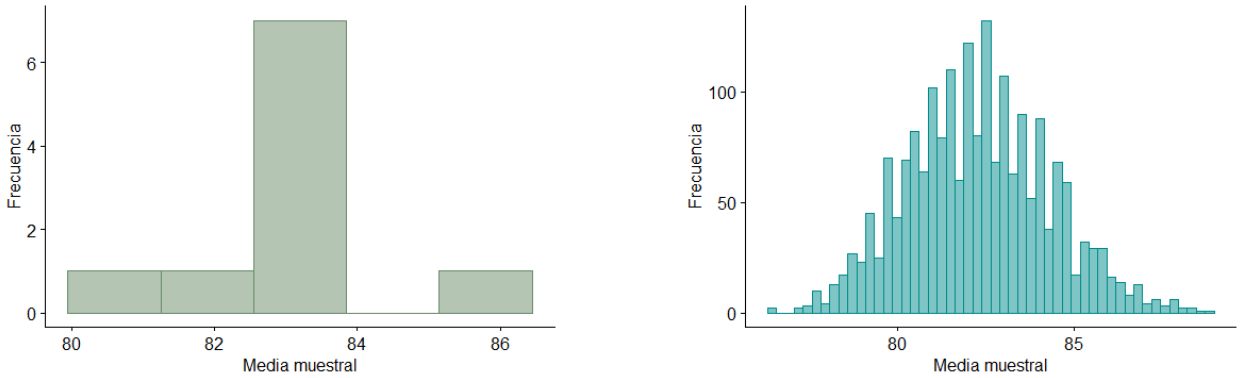
Original	**Bt 1**	**Bt 2**	**Bt 3**	**Bt 4**	**Bt 5**	**Bt 6**	**Bt 7**	**Bt 8**	**Bt 9**	**Bt 10**
79	84	84	84	94	94	94	76	82	75	79
75	94	75	82	88	75	75	88	88	82	79
84	79	82	84	90	90	94	79	79	94	94
75	79	88	79	90	82	94	76	75	94	82
94	88	79	79	90	82	76	84	75	79	84
82	75	84	79	75	76	75	75	79	76	82
76	88	82	84	75	76	79	75	90	88	94
90	88	79	79	75	82	75	79	88	88	79
79	84	79	90	94	90	88	75	75	79	75
88	75	94	88	82	88	76	94	90	82	82
82,2	**83,4**	**82,6**	**82,8**	**85,3**	**83,5**	**82,6**	**80,1**	**82,1**	**83,7**	**83,0**

Tabla 11.2: muestra original y remuestreos de bootstrap

La figura 11.10 muestra la distribución bootstrap de la media para los 10 remuestreos del ejemplo (figura 11.10a) y para 2.000 remuestreos (figura 11.10b). En ella podemos ver claramente que, a medida que la cantidad de muestras bootstrap crece, la distribución bootstrap de la media se asemeja cada vez más a la distribución normal, por lo que se acerca a la forma que esperaríamos para la distribución muestral. La figura 11.10b también nos da una idea acerca de la variabilidad de las medias de los diferentes remuestreos. Sin embargo, podemos conocer mejor esta variabilidad calculando el error estándar, dado por la ecuación 11.7, donde $\overline{x}_i^*$ es la media del i-ésimo remuestreo y B es la cantidad de remuestreos realizados.

$$SE_{b,\overline{x}} = \sqrt{\frac{1}{B-1} \cdot \sum_{i=1}^{B} \left(\overline{x}_i^* - \frac{1}{B} \cdot \sum_{i=1}^{B} \overline{x}_i^* \right)^2} \tag{11.7}$$

Fijémonos en que la subexpresión $\frac{1}{B} \cdot \sum_{i=1}^{B} \overline{x}_i^*$ de la ecuación 11.7 corresponde al promedio de las medias de los remuestreos (media de la distribución bootstrap), por lo que la subexpresión $\sum_{i=1}^{B} \left(\overline{x}_i^* - \frac{1}{B} \cdot \sum_{i=1}^{B} \overline{x}_i^* \right)^2$ es, a su vez, la suma de las desviaciones cuadradas, con lo que resulta evidente la semejanza con el cálculo del error estándar para la media de una muestra presentado en el capítulo 4:



(a) 10 remuestreos.

(b) 2.000 remuestreos.

Figura 11.10: distribución bootstrap de la media

$$SE_{\overline{x}} = \frac{s}{\sqrt{n}}$$

Para el ejemplo con $B = 10$, entonces, tenemos que la media de la distribución bootstrap es:

$$\frac{1}{B} \cdot \sum_{i=1}^{B} \overline{x}_i^* = 83,4 + 82,6 + 82,8 + 85,3 + 83,5 + 82,6 + 80,1 + 82,1 + 83,7 + 83,0 = 82,91$$

Con lo que la suma de las desviaciones cuadradas es:

$$\begin{aligned}\sum_{i=1}^{B} (\overline{x}_i^* - 82,91)^2 &= (83,4 - 82,91)^2 + (82,6 - 82,91)^2 + (82,8 - 82,91)^2 + \\ &+ (85,3 - 82,91)^2 + (83,5 - 82,91)^2 + (82,6 - 82,91)^2 + (80,1 - 82,91)^2 + \\ &+ (82,1 - 82,91)^2 + (83,7 - 82,91)^2 + (83,0 - 82,91)^2 = 15,689\end{aligned}$$

En consecuencia, el error estándar de la distribución bootstrap es:

$$SE_{B,\overline{x}} = \sqrt{\frac{1}{B-1} \cdot 15,689} = 1,320$$

Otra medida que suele emplearse para la distribución bootstrap es el **sesgo**, que indica cuánto se aleja el estadístico de interés de la muestra original (θ) de la media de la distribución bootstrap, como muestra la ecuación 11.8.

$$sesgo = \theta - \frac{1}{B} \cdot \sum_{i=1}^{B} \overline{x}_i^* \tag{11.8}$$

Para el ejemplo:

$$sesgo = \overline{x} - \frac{1}{B} \cdot \sum_{i=1}^{B} \overline{x}_i^* = 82,2 - 82,91 = -0,71$$

Ahora que ya conocemos la distribución bootstrap para la media, podemos entonces **construir un intervalo de confianza para la media de la población**, para lo que abordaremos diferentes alternativas.

Cuando la distribución bootstrap se asemeja a la normal y el sesgo es pequeño en comparación con el estimador calculado (como en este caso), podemos construir el intervalo de confianza usando la distribución t, del mismo modo que conocimos en el capítulo 4, como muestra la ecuación 11.9, usando el error estándar de la distribución bootstrap. El valor crítico de t, t^*, se obtiene para el nivel de significación establecido para el estudio y $\nu = n - 1$ grados de libertad (no olvidemos que n es el tamaño de la muestra original).

$$\theta \pm t^* \cdot SE_{B,\overline{x}} \tag{11.9}$$

Así, si consideramos para este ejemplo un nivel de significación $\alpha = 0,01$, el valor crítico de t (para dos colas) con 9 grados de libertad es $t^* = 3,25$. En consecuencia, el intervalo de confianza resultante para la media de la población es:

$$82,2 \pm 3,25 \cdot 1,320 = (77,910; 86,490)$$

Otra alternativa cuando la distribución bootstrap se asemeja a la normal, que tiene en cuenta posibles asimetrías, es construir el intervalo de confianza en base a cuantiles. En este caso, para $\alpha = 0,01$, los límites del intervalo están dados por los percentiles 1 y 99 de la distribución bootstrap:

$$(80,280; 85,156)$$

Cuando los intervalos de confianza obtenidos por ambos métodos son muy diferentes, es clara señal de que no podemos asumir que la distribución bootstrap se asemeja a la normal. En general, lo más recomendable es usar otro esquema, llamado BCa (del inglés *bias-corrected accelerated*), es decir, con sesgo corregido y acelerado. No se detalla aquí el procedimiento, pues requiere el empleo de software.

Desde luego, es inviable usar bootstrapping sin software. R ofrece el paquete `boot`, con las funciones `boot(data, statistic, R)` para generar la distribución bootstrap y `boot.ci(boot.out, conf, type)` para calcular los intervalos de confianza, donde:

- `data`: el conjunto de datos. En caso de matrices y data frames, se considera cada fila como una observación con múltiples variables.
- `statistic`: función que se aplica a los datos y devuelve un vector con el (o los) estadístico(s) de interés.
- `R`: cantidad de remuestreos bootstrap (B).
- `boot.out`: objeto de la clase `boot`, generado por la función `boot()`.
- `conf`: nivel de confianza ($1 - \alpha$).
- `type`: string o vector que indica los tipos de intervalo de confianza a construir ("`norm`" para el basado en la distribución normal, "`perc`" para el basado en los percentiles y "`bca`" para el método recomendado).

Debemos mencionar que la función `boot()` puede recibir otros muchos argumentos, los cuales escapan al alcance de los contenidos aquí expuestos. El script 11.6 construye intervalos de confianza mediante bootstrapping para el ejemplo, con $B = 2000$ y manteniendo el nivel de significación $\alpha = 0,01$. En las líneas 15–17 se construye la función para el estadístico de interés (en este caso la media), que luego usa la función `boot()` para generar la distribución bootstrap (líneas 19–20), obteniéndose el resultado que se presenta en la figura 11.11.

Podemos ver gráficamente esta distribución mediante un histograma y un gráfico Q-Q (figura 11.12), gracias a la llamada a la función `plot()` con el resultado entregado por `boot()` como argumento (línea 23).

En las líneas 26–39 se muestra el uso de `boot.ci()` para construir los intervalos de confianza mediante diferentes métodos, obteniéndose los siguientes resultados:

- Intervalo de confianza usando aproximación normal: $(77,03; 87,25)$.
- Intervalo de confianza usando percentiles: $(77,4; 87,7)$.
- Intervalo de confianza BCa: $(77,48; 87,90)$.

```
ORDINARY NONPARAMETRIC BOOTSTRAP


Call:
boot(data = muestra, statistic = media, R = B)


Bootstrap Statistics :
    original  bias    std. error
t1*     82.2 0.06125     1.98329
```

Figura 11.11: distribución bootstrap generada mediante `boot()` para la media.

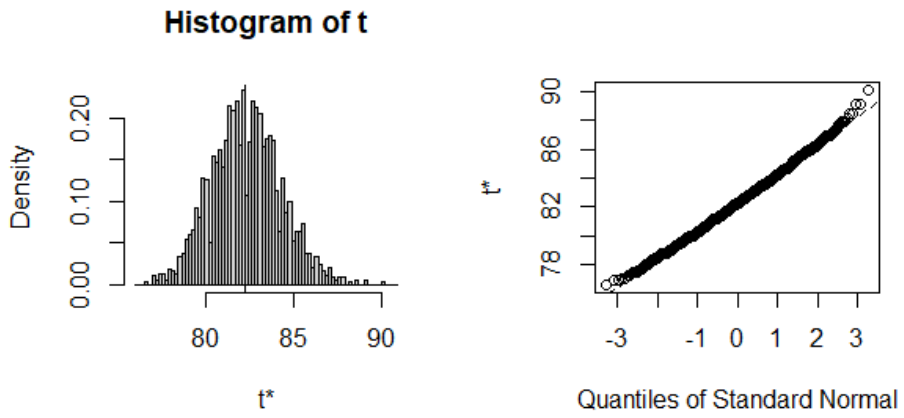


Figura 11.12: histograma y gráfico Q-Q de la distribución bootstrap generada mediante `boot()` para la media.

Las líneas 42–45 muestran otra alternativa para construir la distribución bootstrap por medio del paquete `bootES`, que ofrece la función `bootES(data, R, ci.type, ci.conf, plot, ...)`. Esta función realiza internamente una llamada a la función `boot()` descrita en los párrafos precedentes, pero no requiere implementar previamente la función para el cálculo de la media. Debemos tener en cuenta que aquí solo se muestran algunos de los argumentos, a saber:

- `data`: conjunto de datos.
- `R`: cantidad de remuestreos bootstrap (B).
- `ci.type`: tipo de intervalo de confianza a construir (opcional), con las mismas opciones descritas para `boot.ci()`.
- `ci.conf`: nivel de significación para el intervalo de confianza (opcional, por defecto 0.95).
- `plot`: por defecto con valor `FALSE`, cuando es `TRUE` genera una figura con el histograma y el gráfico Q-Q de la ditribución bootstrap.
- `...`: permite pasar otros argumentos para la función `boot()` subyacente.

Las figuras 11.13 y 11.14 muestran los resultados obtenidos, ligeramente diferentes a los anteriores. A partir de estos últimos podemos concluir que tenemos 95 % de confianza de que el algoritmo tarda entre 77,48 ms y 87,90 ms en ejecutar las instancias del tamaño seleccionado.

Script 11.6: construcción de un intervalo de confianza para la media poblacional mediante bootstrapping.

```
1 library(boot)
2 library(bootES)
```

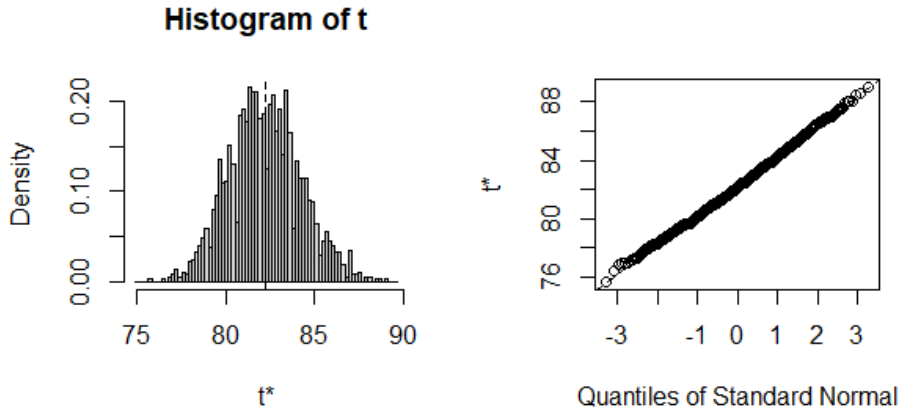


Figura 11.13: histograma y gráfico Q-Q de la distribución bootstrap generada mediante `bootES()` para la media.

```
99.00% bca Confidence Interval, 2000 replicates
Stat        CI (Low)    CI (High)   bias        SE
82.200      77.482      87.900      0.061       1.983
```

Figura 11.14: distribución bootstrap e intervalo de confianza para la media de la población generada mediante `bootES()`.

```

# Crear muestra inicial, mostrar su histograma y calcular la media.
muestra <- c(79, 75, 84, 75, 94, 82, 76, 90, 79, 88)
datos <- data.frame(muestra)

# Establecer cantidad de remuestreos y nivel de significación.
B = 2000
alfa <- 0.01

cat("Paquete boot\n")

# Construir distribución bootstrap usando el paquete boot.
media <- function(valores, i) {
  mean(valores[i])
}

set.seed(432)
distribucion_b <- boot(muestra, statistic = media, R = B)
print(distribucion_b)

# Graficar distribución bootstrap.
print(plot(distribucion_b))

# Construir intervalos de confianza.
intervalo_t <- boot.ci(distribucion_b, conf = 1 - alfa, type = "norm")

cat("\n\nIntervalo de confianza usando distribución t:\n")
print(intervalo_t)

```

```r
intervalo_per <- boot.ci(distribucion_b, conf = 1 - alfa, type = "perc")

cat("\n\nIntervalo de confianza usando percentiles:\n")
print(intervalo_per)

intervalo_bca <- boot.ci(distribucion_b, conf = 1 - alfa, type = "bca")

cat("\n\nIntervalo de confianza BCa:\n")
print(intervalo_bca)

# Construir distribución bootstrap usando el paquete bootES.
set.seed(432)

distribucion_bootstrapES <- bootES(muestra, R = B, ci.type = "bca",
                                   ci.conf = 1 - alfa, plot = TRUE)

print(distribucion_bootstrapES)
```

Supongamos ahora que Helen desea hacer una prueba de hipótesis para ver si el tiempo promedio de ejecución del algoritmo para instancias del tamaño seleccionado es mayor a 75 milisegundos. Así, tenemos que:

Denotando como μ al tiempo medio que tarda el algoritmo de Helen para resolver instancias de tamaño fijo del problema, entonces:
H_0: $\mu = 75$ [ms]
H_A: $\mu > 75$ [ms]

El contraste de hipótesis requiere siempre generar la distribución centrada en el valor nulo para, a partir de ella, obtener el valor p. Sabemos que la distribución bootstrap se centra *alrededor* del valor observado, por lo que debemos desplazarla para cumplir con esta condición. Para lograrlo, simplemente necesitamos restar a cada observación de la distribución bootstrap la diferencia entre su valor promedio y el valor nulo.

Para calcular el valor p, seguimos la fórmula señalada en la ecuación 11.10, donde:

- r: cantidad de observaciones en la distribución bootstrap (desplazada) a lo menos tan extremas como el estadístico observado.
- B: cantidad de repeticiones bootstrap consideradas en la simulación.

$$p = \frac{r+1}{B+1} \tag{11.10}$$

Tras hacer la prueba (script 11.7), obtenemos que $p = 0,001$, menor que el nivel de significación, por lo que la evidencia es suficientemente fuerte para rechazar la hipótesis nula en favor de la hipótesis alternativa. En consecuencia, concluimos con 99,5 % de confianza que el tiempo de ejecución promedio del algoritmo para instancias del tamaño seleccionado supera los 75 milisegundos.

Script 11.7: inferencia sobre la media de una muestra con bootstrapping.

```r
library(boot)

set.seed(432)

# Crear muestra inicial, mostrar su histograma y calcular la media.
muestra <- c(79, 75, 84, 75, 94, 82, 76, 90, 79, 88)
valor_observado <- mean(muestra)
datos <- data.frame(muestra)

# Construir distribución bootstrap.
B <- 2000

media <- function(valores, i) {
```

```
  mean(valores[i])
}

distribucion_b <- boot(muestra, statistic = media, R = B)

# Desplazar la distribución bootstrap para que se centre en
# el valor nulo.
valor_nulo <- 75
desplazamiento <- mean(distribucion_b[["t"]]) - valor_nulo
distribucion_nula <- distribucion_b[["t"]] - desplazamiento

# Determinar el valor p.
p <- (sum(distribucion_nula > valor_observado) + 1) / (B + 1)
cat("Valor p:", p)
```

11.3.2 Bootstrapping para dos muestras independientes

El proceso para comparar dos poblaciones mediante bootstraping es similar al que ya conocimos para una única población. Si tenemos dos muestras independientes A y B provenientes de dos poblaciones diferentes, de tamaños n_A y n_B respectivamente, los pasos a seguir son:

1. Fijar la cantidad B de repeticiones bootstrap.
2. En cada repetición, hacer un remuestreo con reposición de tamaño n_A a partir de la muestra A y otro de tamaño n_B a partir de la muestra B.
3. En cada repetición, calcular el estadístico de interés para generar la distribución bootstrap.
4. Construir el intervalo de confianza para el estadístico de interés.

Supongamos que una Universidad desea estudiar la diferencia entre las calificaciones finales de hombres y mujeres que rinden una asignatura inicial de programación por primera vez. Para ello, disponen de las notas (en escala de 1,0 a 7,0) de 27 hombres y 19 mujeres:

- Hombres: 1,3; 1,5; 1,6; 1,7; 1,7; 1,9; 2,3; 2,4; 2,6; 2,6; 2,7; 2,8; 3,2; 3,7; 4,1; 4,4; 4,5; 4,8; 5,2; 5,2; 5,3; 5,5; 5,5; 5,6; 5,6; 5,7; 5,7
- Mujeres: 3,5; 3,6; 3,8; 4,3; 4,5; 4,5; 4,9; 5,1; 5,3; 5,3; 5,5; 5,8; 6,0; 6,3; 6,3; 6,4; 6,4; 6,6; 6,7

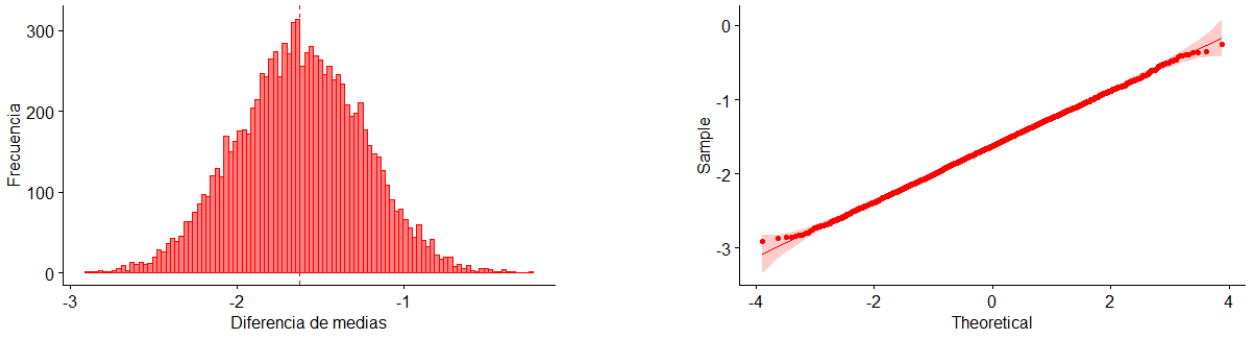
Tras aplicar pruebas de Shapiro-Wilk (figura 11.15), los investigadores han comprobado que las notas de los varones no siguen una distribución normal, por lo que han decidido usar bootstrapping para la prueba de hipótesis, con un nivel de significación $\alpha = 0,05$ y $B = 9999$ repeticiones.

```
Shapiro-Wilk normality test

data:  hombres
W = 0.88357, p-value = 0.005742


Shapiro-Wilk normality test

data:  mujeres
W = 0.93022, p-value = 0.1748
```

Figura 11.15: pruebas de normalidad de Shappiro-Wilk para ambas muestras.

La media observada (en la muestra original) para la calificación final de las mujeres es $\overline{x}_m = 5,305$, mientras que para los hombres es $\overline{x}_h = 3,670$. Así, la diferencia observada es $\overline{x}_h - \overline{x}_m = -1,635$.

La distribución bootstrap de la diferencia de medias se asemeja a la normal (figura 11.16), con media $\overline{x} = -1,628$ y desviación estándar $s = 0,377$.



(a) histograma con la media añadida.

(b) gráfico Q-Q.

Figura 11.16: distribución bootstrap de la diferencia de medias.

Al construir el intervalo de confianza mediante el método BCa para la distribución bootstrap, R nos entrega como resultado el intervalo $(-2,372; -0,894)$. En consecuencia, concluimos con 95 % de confianza que las mujeres tienen, en promedio, mejor calificación final que los hombres, con una diferencia de entre 0,894 y 2,372 puntos.

El script 11.8 muestra el desarrollo de este ejemplo en R, el cual usa la función `two.boot(sample1, sample2, FUN, R)` del paquete `simpleboot`, donde:

- `sample1, sample2`: muestras originales.
- `FUN`: función que, para cada muestra, calcula el estadístico de interés θ.
- `R`: cantidad de remuestreos con repetición.

Esta función opera generando remuestreos para cada una de las muestras originales, y calculando en cada iteración el estadístico $\theta_1 - \theta_2$, donde los subíndices señalan la muestra correspondiente.

Script 11.8: bootstraping para la diferencia de medias.

```
library(simpleboot)
library(boot)
library(ggpubr)

set.seed(432)

# Ingresar datos originales
hombres <- c(1.3, 1.5, 1.6, 1.7, 1.7, 1.9, 2.3, 2.4, 2.6, 2.6, 2.7,
             2.8, 3.2, 3.7, 4.1, 4.4, 4.5, 4.8, 5.2, 5.2, 5.3, 5.5,
             5.5, 5.6, 5.6, 5.7, 5.7)

mujeres <- c(3.5, 3.6, 3.8, 4.3, 4.5, 4.5, 4.9, 5.1, 5.3, 5.3, 5.5,
             5.8, 6.0, 6.3, 6.3, 6.4, 6.4, 6.6, 6.7)

n_hombres <- length(hombres)
n_mujeres <- length(mujeres)

sexo <- c(rep("Hombre", n_hombres), rep("Mujer", n_mujeres))
nota <- c(hombres, mujeres)
datos <- data.frame(nota, sexo)

```

```r
# Comprobar normalidad de las muestras.
print(shapiro.test(hombres))
print(shapiro.test(mujeres))

# Calcular la diferencia observada entre las medias muestrales.
media_hombres <- mean(hombres)
media_mujeres <- mean(mujeres)
diferencia_observada <- media_hombres - media_mujeres

cat("diferencia observada:", media_hombres - media_mujeres, "\n\n")

# Establecer el nivel de significación.
alfa <- 0.05

# Crear la distribución bootstrap.
B <- 9999
distribucion_bootstrap <- two.boot(hombres, mujeres, FUN = mean, R = B)

# Examinar la distribución bootstrap.
valores <- data.frame(distribucion_bootstrap$t)
colnames(valores) <- "valores"

histograma <- gghistogram(valores, x = "valores", color = "red",
                          fill = "red", bins = 100,
                          xlab = "Diferencia de medias",
                          ylab = "Frecuencia", add = "mean")

print(histograma)

qq <- ggqqplot(valores, x = "valores", color = "red")
print(qq)

cat("Distribución bootstrap:\n")
cat("\tMedia:", mean(valores$valores), "\n")
cat("\tDesviación estándar:", sd(valores$valores), "\n\n")

# Construir el intervalo de confianza.
intervalo_bca <- boot.ci(distribucion_bootstrap, conf = 1 - alfa,
                         type = "bca")

print(intervalo_bca)
```

Supongamos ahora que el estudio del ejemplo desea determinar, con un nivel de significación $\alpha = 0,05$, si la diferencia entre las calificaciones finales de hombres y mujeres es igual a 1,5 puntos. Para ello, formulamos las siguientes hipótesis:

Sean μ_h y μ_m las calificaciones finales de hombres y mujeres, respectivamente, que rinden una asignatura inicial de programación por primera vez en la Universidad en estudio, entonces:
H_0: $\mu_h - \mu_m = 1,5$
H_A: $\mu_h - \mu_m \neq 1,5$

Tras aplicar bootstrapping para la prueba de hipótesis (script 11.9), obtenemos un valor p de $p = 0,364$, superior al nivel de significación, por lo que fallamos al rechazar la hipótesis nula. En consecuencia, concluimos con 95 % de confianza que no es posible descartar que la diferencia en la calificación final entre hombres y mujeres es de 1,5 puntos.

Script 11.9: bootstraping para inferir acerca de la diferencia de medias.

```r
library(simpleboot)
library(boot)
```

```r
library(ggpubr)

set.seed(432)

# Ingresar datos originales
hombres <- c(1.3, 1.5, 1.6, 1.7, 1.7, 1.9, 2.3, 2.4, 2.6, 2.6, 2.7,
             2.8, 3.2, 3.7, 4.1, 4.4, 4.5, 4.8, 5.2, 5.2, 5.3, 5.5,
             5.5, 5.6, 5.6, 5.7, 5.7)

mujeres <- c(3.5, 3.6, 3.8, 4.3, 4.5, 4.5, 4.9, 5.1, 5.3, 5.3, 5.5,
             5.8, 6.0, 6.3, 6.3, 6.4, 6.4, 6.6, 6.7)

n_hombres <- length(hombres)
n_mujeres <- length(mujeres)

sexo <- c(rep("Hombre", n_hombres), rep("Mujer", n_mujeres))
nota <- c(hombres, mujeres)
datos <- data.frame(nota, sexo)

# Calcular la diferencia observada entre las medias muestrales.
media_hombres <- mean(hombres)
media_mujeres <- mean(mujeres)
valor_observado <- media_hombres - media_mujeres

# Crear la distribución bootstrap.
B <- 9999
valor_nulo <- 1.5
distribucion_bootstrap <- two.boot(hombres, mujeres, FUN = mean, R = B)
desplazamiento <- mean(distribucion_bootstrap[["t"]]) - valor_nulo
distribucion_nula <- distribucion_bootstrap[["t"]] - desplazamiento

# Determinar el valor p.
p <- (sum(abs(distribucion_nula) > abs(valor_observado)) + 1) / (B + 1)
cat("Valor p:", p)
```

11.3.3 Bootstrapping para dos muestras pareadas

En este caso, el procedimiento resulta muy sencillo. A partir de las dos muestras originales, se crea una nueva muestra con la diferencia entre ambas, y luego se realiza el proceso especificado para la construcción de un intervalo de confianza que ya conocimos para el caso de una única muestra.

Supongamos ahora, que la Universidad del ejemplo anterior desea saber si existe diferencia entre las calificaciones obtenidas en la primera y la segunda prueba de un curso inicial de programación. Para ello, dispone de las calificaciones (en escala de 1,0 a 7,0) obtenidas en ambas pruebas para una muestra de 20 estudiantes, como presenta la tabla 11.3. Han decidido llevar a cabo el estudio mediante bootstrapping con $B = 3999$ repeticiones y un nivel de significación $\alpha = 0,05$, para lo cual han creado en R el script 11.10, obteniendo los resultados que se presentan en la figura 11.17.

A partir del resultado anterior, concluimos con 95 % de confianza que, en promedio, la diferencia de las medias para las calificaciones de la primera y la segunda evaluación se encuentra en el intervalo $(-0,656; 1,439)$, por lo que no podemos desechar la idea de que no existe una diferencia estadísticamente significativa entre ellas.

Script 11.10: bootstraping para la media de las diferencias.

Alumno	Prueba 1	Prueba 2
1	3,5	5,2
2	2,7	5,1
3	1,0	5,9
4	1,8	4,8
5	1,6	1,4
6	4,3	2,3
7	5,8	6,8
8	6,4	5,3
9	3,9	3,1
10	4,3	3,8
11	3,4	4,6
12	5,3	1,2
13	5,8	3,9
14	5,3	2,0
15	2,0	1,7
16	1,3	3,3
17	4,0	6,0
18	5,3	4,8
19	1,6	6,9
20	3,6	1,3

Tabla 11.3: calificaciones de los estudiantes en la primera y la segunda prueba de un curso inicial de programación.

```
95.00% bca Confidence Interval, 3999 replicates
Stat        CI (Low)    CI (High)   bias        SE
0.325       -0.656      1.439       0.001       0.541
```

Figura 11.17: intervalo de confianza BCa para la media de las diferencias.

```r
library(bootES)

set.seed(432)

# Ingresar datos originales.
alumno <- 1:20

prueba_1 <- c(3.5, 2.7, 1.0, 1.8, 1.6, 4.3, 5.8, 6.4, 3.9, 4.3, 3.4,
              5.3, 5.8, 5.3, 2.0, 1.3, 4.0, 5.3, 1.6, 3.6)

prueba_2 <- c(5.2, 5.1, 5.9, 4.8, 1.4, 2.3, 6.8, 5.3, 3.1, 3.8, 4.6,
              1.2, 3.9, 2.0, 1.7, 3.3, 6.0, 4.8, 6.9, 1.3)

# Establecer nivel de significación.
alfa <- 0.05

# Calcular la diferencia entre ambas observaciones.
diferencia <- prueba_2 - prueba_1

# Generar la distribución bootstrap y su intervalo de confianza.
B <- 3999

distribucion_bootstrapES <- bootES(diferencia, R = B, ci.type = "bca",
                                   ci.conf = 1 - alfa, plot = FALSE)
```

```r

print(distribucion_bootstrapES)
```

Ahora la Universidad del ejemplo desea saber si la diferencia promedio entre las calificaciones obtenidas en la primera y la segunda prueba de un curso inicial de programación es de 5 décimas. Así, considerando un nivel de significación $\alpha = 0,05$, los investigadores formulan las siguientes hipótesis:

Sea μ_{dif} la media de las diferencias en las calificaciones obtenidas por los estudiantes en la primera y la segunda prueba del curso inicial de programación de la Universidad, entonces:

H_0: $\mu_{dif} = 0,5$
H_0: $\mu_{dif} \neq 0,5$

Tras efectuar la prueba de hipótesis mediante bootstrapping (script 11.11) obtienen un valor p de $p = 0,573$, por lo que la evidencia no es suficientemente fuerte como para rechazar la hipótesis nula. En consecuencia, los investigadores concluyen con 95 % de confianza que no hay razones para no pensar que la diferencia de las calificaciones obtenidas en ambas evaluaciones es de 5 décimas

Script 11.11: bootstraping para inferir acerca de la media de las diferencias.

```r
library(bootES)

set.seed(432)

# Ingresar datos originales.
alumno <- 1:20

prueba_1 <- c(3.5, 2.7, 1.0, 1.8, 1.6, 4.3, 5.8, 6.4, 3.9, 4.3, 3.4,
              5.3, 5.8, 5.3, 2.0, 1.3, 4.0, 5.3, 1.6, 3.6)

prueba_2 <- c(5.2, 5.1, 5.9, 4.8, 1.4, 2.3, 6.8, 5.3, 3.1, 3.8, 4.6,
              1.2, 3.9, 2.0, 1.7, 3.3, 6.0, 4.8, 6.9, 1.3)

# Establecer nivel de significación.
alfa <- 0.05

# Calcular la diferencia entre ambas observaciones.
diferencia <- prueba_2 - prueba_1

# Calcular la media observada de las diferencias.
valor_observado <- mean(diferencia)

# Generar la distribución bootstrap y su intervalo de confianza.
B <- 3999
valor_nulo <- 0.5

distribucion_bootstrapES <- bootES(diferencia, R = B, ci.type = "bca",
                                   ci.conf = 1 - alfa, plot = FALSE)

distribucion_nula <- distribucion_bootstrapES[["t"]] - valor_nulo


# Determinar el valor p.
p <- (sum(abs(distribucion_nula) > abs(valor_observado)) + 1) / (B + 1)
cat("Valor p:", p)
```

11.4 PRUEBAS DE PERMUTACIONES

En el capítulo 7 conocimos la prueba exacta de Fisher, la cual obtiene un valor p exacto tras calcular todas las permutaciones de los datos con iguales valores marginales en una tabla de contingencia como alternativa para muestras pequeñas de la prueba y considerar únicamente aquellas permutaciones que ocurren con igual o menor probabilidad que la obtenida para los datos del estudio.

La prueba exacta de Fisher es lo que se conoce como una **prueba exacta de permutaciones**, cuyo único requisito es la **intercambiabilidad**: si se cumple la hipótesis nula, todas las permutaciones pueden ocurrir con igual probabilidad. En la práctica, este tipo de métodos puede emplearse para diversos estadísticos, tales como la proporción, la media y la varianza. Puesto que el valor p entregado por las pruebas de permutaciones es exacto, no es posible obtener un intervalo de confianza.

En términos generales, las pruebas exactas de permutaciones para la diferencia entre dos grupos A y B (puede extenderse esta idea para más grupos) de tamaños n_A y n_B, respectivamente, sigue los siguientes pasos:

1. Calcular la diferencia entre el estadístico de interés observado para ambos grupos.
2. Juntar ambas muestras en una muestra combinada.
3. Obtener todas las permutaciones de la muestra combinada en que se pueden distribuir las observaciones en dos grupos de tamaños n_A y n_B.
4. Construir la distribución de las posibles diferencias, calculando la diferencia entre el estadístico de interés obtenido para ambos grupos en cada una de las permutaciones.
5. Calcular el valor p exacto, dado por la proporción de permutaciones en que el valor (absoluto, si es bilateral) de la diferencia calculada es menor/mayor o igual al valor (absoluto si es bilateral) de la diferencia observada.

Puesto que las pruebas exactas de permutaciones requieren calcular todas las permutaciones, solo resultan adecuadas para muestras pequeñas, pues requieren de una enorme cantidad de cómputos. En consecuencia, si la muestra es grande, suele tomarse una **muestra aleatoria de las permutaciones** posibles, procedimiento que suele denominarse **simulación de Monte Carlo**, y a partir de ella calcular un valor p aproximado (puesto que una permutación se podría seleccionar más de una vez) por medio de la ecuación 11.10. Podemos ver que en la ecuación 11.10 se suma 1 tanto al numerador como al denominador. Esto corresponde a una corrección que debemos aplicar puesto que el método de Monte Carlo no es insesgado.

De los párrafos anteriores se desprende que las pruebas de permutaciones (exactas o no) son adecuadas para el contraste de hipótesis con dos o más muestras, pues determinan una significación estadística (valor p). En términos generales, el procedimiento para efectuar una prueba de permutaciones usando simulaciones de Monte Carlo no es muy distinto al de bootstrapping, aunque hay algunas diferencias fundamentales en el trasfondo:

1. Formular las hipótesis a contrastar (e identificar el estadístico de interés θ).
2. Crear una gran cantidad P de permutaciones (equivalente a B en la ecuación 11.10, y generalmente terminada en 9 para simplificar los cómputos) a partir de las muestras originales, usando **muestreo sin reposición sobre la muestra combinada**, y obtener el estadístico θ para cada una de las muestras.
3. Generar la distribución que el estadístico θ tendría si la hipótesis nula fuese cierta.
4. Determinar la probabilidad de encontrar un valor de θ al menos tan extremo como el observado en la distribución generada.

Debemos fijarnos en que, a diferencia de bootstrapping, las pruebas de permutaciones usan muestreo sin reposición puesto que, si la hipótesis nula fuera cierta, cada permutación de los valores obtenidos en la muestra combinada sería igualmente probable. Así, lo que se hace en cada repetición es tomar una muestra sin repetición de la muestra original (es decir, "reordenar" las observaciones) y asignar aleatoriamente cada observación a uno de los grupos, respetando los tamaños n_A y n_B de las muestras originales.

11.4.1 Prueba de permutaciones para comparar una variable continua en dos muestras independientes

El profesor de una asignatura inicial de programación, que se imparte para estudiantes de primer año de Ingeniería y estudiantes de último año de otras carreras que pueden cursar dicha asignatura como electivo, desea estudiar si existen diferencias en el rendimiento académico de ambos grupos. Para ello, considera una muestra de $n_A = 20$ estudiantes de primer año de Ingeniería y $n_B = 12$ estudiantes de último año de otras carreras.

El profesor ha decidido comparar el promedio de calificaciones finales de ambos grupos, usando para ello una prueba de permutaciones con $P = 5999$ repeticiones y un nivel de significación $\alpha = 0,05$. La diferencia observada para las muestras originales es $\overline{x}_A - \overline{x}_B = -0,017$, sugiriendo que los estudiantes de Ingeniería tienen peores calificaciones. Así, las hipótesis a contrastar son:

Denotando como μ_A al promedio de calificaciones finales de estudiantes de primer año de Ingeniería en el curso inicial de programación bajo estudio, y como μ_B al promedio de calificaciones finales de estudiantes de último año de otras carreras en el mismo curso, entonces:
H_0: $\mu_A - \mu_B = 0$
H_A: $\mu_A - \mu_B \neq 0$

Tras hacer la prueba, la distribución generada se asemeja bastante a la normal, aunque con una ligera asimetría hacia la derecha (figura 11.18), y el valor p obtenido para el contraste de hipótesis es $p = 0,969$, por lo que concluye con 95 % de confianza que no hay evidencia suficiente para creer que existe diferencia entre los promedios de las calificaciones finales de ambos grupos de estudiantes.

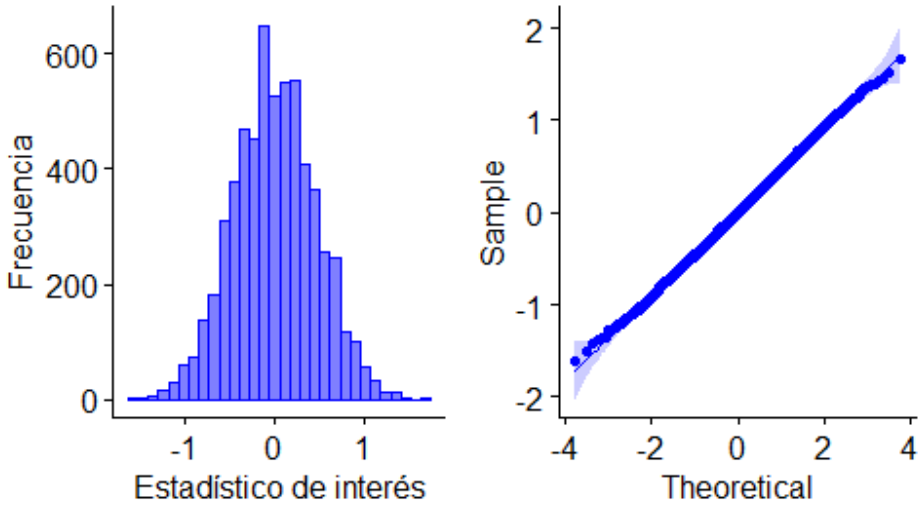


Figura 11.18: histograma y gráfico Q-Q de la distribución para la diferencia de medias generada mediante permutaciones.

Intrigado por este resultado, pues el profesor tiene la fuerte sensación de que, en general, los estudiantes de Ingeniería tienen más calificaciones deficientes que los estudiantes de otras carreras, ha decidido hacer un nuevo estudio con las mismas muestras, comparando ahora la diferencia en la variabilidad (manteniendo la misma cantidad de repeticiones e igual nivel de significación). Así:

Denotando como σ_A a la varianza de las calificaciones finales de estudiantes de primer año de Ingeniería en el curso inicial de programación bajo estudio, y como σ_B a la varianza de las calificaciones finales de estudiantes de último año de otras carreras en el mismo curso, entonces:
H_0: $\sigma_A - \sigma_B = 0$
H_A: $\sigma_A - \sigma_B \neq 0$

La diferencia observada entre las varianzas de la muestra original es $\sigma x_A - \sigma x_B = 2,560$, sugiriendo que la variabilidad de las calificaciones obtenidas por los estudiantes de ingeniería es mayor. Tras efectuar el contraste de hipótesis, obtiene como resultado $p = 0,003$, evidencia suficiente para rechazar la hipótesis nula en favor de la hipótesis alternativa. Así, el profesor concluye que su percepción no es del todo errada, puesto que la variabilidad de las calificaciones es significativamente mayor para los estudiantes de Ingeniería.

Para hacer estos estudios, el profesor desarrolló en R el script 11.12. A pesar de que existen algunos paquetes de R para realizar pruebas de permutaciones, hemos decidido en esta ocasión implementar el procedimiento creando la función `contrastar_hipotesis_permutaciones()`, cuya especificación puede leerse en el script 11.12, la cual realiza el proceso y arroja como resultado el valor p resultante. Podemos ver que esta función opera usando como estadístico de interés la diferencia de un estadístico θ entre dos muestras y que la función que calcula dicho estadístico θ para una muestra se entrega como argumento.

Script 11.12: pruebas de permutaciones para variables numéricas.

```r
library(ggpubr)

# Crear muestras iniciales.
a <- c(5.4, 4.7, 6.3, 2.9, 5.9, 5.1, 2.1, 6.2, 1.6, 6.7, 3.0, 3.3,
       5.0, 4.1, 3.3, 3.4, 1.2, 3.8, 5.8, 4.2)

b <- c(4.0, 4.1, 4.3, 4.3, 4.3, 4.2, 4.3, 4.3, 4.4, 4.1, 4.3, 4.0)

# Establecer semilla y cantidad de repeticiones.
R = 5999
set.seed(432)

# Función para obtener una permutación.
# Argumentos:
# - i: iterador (para llamadas posteriores).
# - muestra_1, muestra_2: muestras.
# Valor:
# - lista con las muestras resultantes tras la permutación.
obtiene_permutacion <- function(i, muestra_1, muestra_2) {
  n_1 <- length(muestra_1)
  combinada <- c(muestra_1, muestra_2)
  n <- length(combinada)
  permutacion <- sample(combinada, n, replace = FALSE)
  nueva_1 <- permutacion[1:n_1]
  nueva_2 <- permutacion[(n_1+1):n]
  return(list(nueva_1, nueva_2))
}

# Función para calcular la diferencia de un estadístico de interés entre las
# dos muestras.
# Argumentos:
# - muestras: lista con las muestras.
# - FUN: nombre de la función que calcula el estadístico de interés.
# Valor:
# - diferencia de un estadístico para dos muestras.
calcular_diferencia <- function(muestras, FUN) {
  muestra_1 <- muestras[[1]]
  muestra_2 <- muestras[[2]]
  diferencia <- FUN(muestra_1) - FUN(muestra_2)
  return(diferencia)
}

# Función para calcular el valor p.
# Argumentos:
```

```r
# - distribucion: distribución nula del estadístico de interés.
# - valor_observado: valor del estadístico de interés para las muestras
#   originales.
# - repeticiones: cantidad de permutaciones a realizar.
# - alternative: tipo de hipótesis alternativa. "two.sided" para
#   hipótesis bilateral, "greater" o "less" para hipótesis unilaterales.
# Valor:
# - el valorp calculado.
calcular_valor_p <- function(distribucion, valor_observado,
                             repeticiones, alternative) {
  if(alternative == "two.sided") {
    numerador <- sum(abs(distribucion) > abs(valor_observado)) + 1
    denominador <- repeticiones + 1
    valor_p <- numerador / denominador
  }
  else if(alternative == "greater") {
    numerador <- sum(distribucion > valor_observado) + 1
    denominador <- repeticiones + 1
    valor_p <- numerador / denominador
  }
  else {
    numerador <- sum(distribucion < valor_observado) + 1
    denominador <- repeticiones + 1
    valor_p <- numerador / denominador
  }

  return(valor_p)
}

# Función para graficar una distribución.
# Argumentos:
# - distribucion: distribución nula del estadístico de interés.
# - ...: otros argumentos a ser entregados a gghistogram y ggqqplot.
graficar_distribucion <- function(distribucion, ...) {
  observaciones <- data.frame(distribucion)

  histograma <- gghistogram(observaciones, x = "distribucion",
                            xlab = "Estadístico de interés",
                            ylab = "Frecuencia", bins = 30, ...)

  qq <- ggqqplot(observaciones, x = "distribucion", ...)

  # Crear una única figura con todos los gráficos de dispersión.
  figura  <- ggarrange(histograma, qq, ncol = 2, nrow = 1)
  print(figura)
}

# Función para hacer la prueba de permutaciones.
# Argumentos:
# - muestra_1, muestra_2: vectores numéricos con las muestras a comparar.
# - repeticiones: cantidad de permutaciones a realizar.
# - FUN: función del estadístico E para el que se calcula la diferencia.
# - alternative: tipo de hipótesis alternativa. "two.sided" para
#   hipótesis bilateral, "greater" o "less" para hipótesis unilaterales.
# - plot: si es TRUE, construye el gráfico de la distribución generada.
# - ...: otros argumentos a ser entregados a graficar_distribucion.
contrastar_hipotesis_permutaciones <- function(muestra_1, muestra_2,
                                               repeticiones, FUN,
                                               alternative, plot, ...) {
```

```r
  cat("Prueba de permutaciones\n\n")
  cat("Hipótesis alternativa:", alternative, "\n")
  observado <- calcular_diferencia(list(muestra_1, muestra_2), FUN)
  cat("Valor observado:", observado, "\n")

  n_1 <- length(muestra_1)

  # Generar permutaciones.
  permutaciones <- lapply(1:repeticiones, obtiene_permutacion, muestra_1,
                          muestra_2)

  # Generar la distribución.
  distribucion <- sapply(permutaciones, calcular_diferencia, FUN)

  # Graficar la distribución.
  if(plot) {
    graficar_distribucion(distribucion, ...)
  }

  # Calcular el valor p.
  valor_p <- calcular_valor_p(distribucion, observado, repeticiones,
                              alternative)

  cat("Valor p:", valor_p, "\n\n")
}



# Hacer pruebas de permutaciones para la media y la varianza.
contrastar_hipotesis_permutaciones(a, b, repeticiones = R, FUN = mean,
                                   alternative = "two.sided", plot = TRUE,
                                   color = "blue", fill = "blue")

contrastar_hipotesis_permutaciones(a, b, repeticiones = R, FUN = var,
                                   alternative = "two.sided", plot = FALSE)
```

11.4.2 Prueba de permutaciones para comparar medias de más de dos muestras correlacionadas

Supongamos ahora que un estudiante de un curso de programación necesita comparar la eficiencia de tres algoritmos de ordenamiento: *quicksort*, *bubblesort* y *mergesort*. Para ello, ha seleccionado aleatoriamente 6 arreglos de igual tamaño y registrado para cada uno de ellos el tiempo de ejecución utilizado por cada algoritmo (en milisegundos) bajo iguales condiciones, como muestra la tabla 11.4.

Instancia	Quicksort	Bubblesort	Mergesort
1	11,2	15,7	12,0
2	22,6	29,3	25,7
3	23,4	30,7	25,7
4	23,3	30,8	23,7
5	21,8	29,8	25,5
6	40,1	50,3	44,7

Tabla 11.4: tiempos de ejecución para las diferentes instancias con cada algoritmo del ejemplo.

Tras comprobar mediante la figura 11.19 que no se cumple la condición de normalidad, el estudiante ha decidido usar permutaciones para resolver su problema. Para ello, ha considerado un nivel de significación $\alpha = 0,01$ y un total de 2999 repeticiones, obteniendo como resultado un valor $p = 0,0003$, mucho menor que el nivel de significación. En consecuencia, concluye con 99 % de confianza que el tiempo de ejecución promedio es significativamente diferente para al menos uno de los algoritmos.

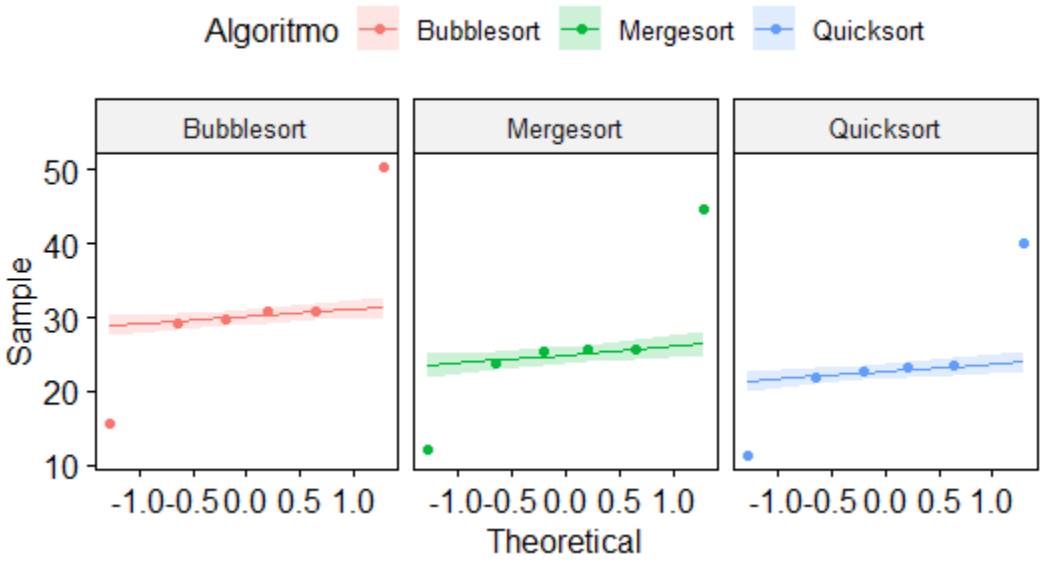


Figura 11.19: gráfico Q-Q para comprobar el supuesto de normalidad para el ejemplo.

A fin de determinar qué algoritmos difieren en su tiempo promedio de ejecución, ha decidido llevar a cabo un procedimiento post-hoc, calculando los valores p para las medias de las diferencias entre cada par de grupos para las diferentes permutaciones, obteniendo los resultados que se presentan en la figura 11.20. En consecuencia, el estudiante concluye con 99 % de confianza, que existen diferencias significativas en el tiempo promedio de ejecución entre los algoritmos Quicksort y Bubblesort y los algoritmos Bubblesort y Mergesort. Al estudiar las diferencias observadas, puede ver que Bubblesort es menos eficiente que los dos algoritmos restantes.

```
Análisis post-hoc (permutaciones) para la diferencia de las medias
----------------------------------------------------------
Valores p:
Quicksort - Bubblesort: 0.000
Quicksort - Mergesort: 0.116
Bubblesort - Mergesort: 0.003

Diferencias observadas:
Quicksort - Bubblesort: -7.367
Quicksort - Mergesort: -2.483
Bubblesort - Mergesort: 4.883
```

Figura 11.20: resultado del procedimiento post-hoc.

El script 11.13 corresponde a la solución desarrollada por el estudiante.

Script 11.13: prueba de permutaciones para muestras correlacionadas.

```
1 library(ggpubr)
2 library(ez)
```

```r
library(tidyverse)

# Crear el data frame.
Quicksort <- c(11.2, 22.6, 23.4, 23.3, 21.8, 40.1)
Bubblesort <- c(15.7, 29.3, 30.7, 30.8, 29.8, 50.3)
Mergesort <- c(12.0, 25.7, 25.7, 23.7, 25.5, 44.7)
Instancia <- factor(1:6)
datos_anchos <- data.frame(Instancia, Quicksort, Bubblesort, Mergesort)

datos_largos <- datos_anchos %>% pivot_longer(c("Quicksort", "Bubblesort",
                                                "Mergesort"),
                                              names_to = "Algoritmo",
                                              values_to = "Tiempo")

datos_largos[["Algoritmo"]] <- factor(datos_largos[["Algoritmo"]])

# Verificar condición de normalidad.
g <- ggqqplot(datos_largos, "Tiempo", facet.by = "Algoritmo",
              color = "Algoritmo")

print(g)

# Establecer nivel de significación.
alfa <- 0.01

# Obtener el valor observado, correspondiente al estadístico F entregado
# por ANOVA para la muestra original.
anova <- ezANOVA(datos_largos, dv = Tiempo, within = Algoritmo,
                 wid = Instancia, return_aov = TRUE)

valor_observado <- anova[["ANOVA"]][["F"]]

# Generar permutaciones.
R = 2999
# copia_ancha <- data.frame(datos_anchos)

set.seed(432)

# Función para obtener una permutación.
# Devuelve una matriz de datos con formato ancho.
obtiene_permutacion <- function(i, df_ancho) {
  df_ancho[, 2:4] <- t(apply(df_ancho[, 2:4], 1, sample))
  return(df_ancho)
}

# Obtiene permutaciones
permutaciones <- lapply(1:R, obtiene_permutacion, datos_anchos)

# Función para obtener el estadístico F para una matriz de datos con formato
# ancho.
obtiene_F <- function(df_ancho) {
  df_largo <- df_ancho %>% pivot_longer(c("Quicksort", "Bubblesort",
                                          "Mergesort"),
                                        names_to = "Algoritmo",
                                        values_to = "Tiempo")

  df_largo[["Algoritmo"]] <- factor(df_largo[["Algoritmo"]])

  anova <- ezANOVA(df_largo, dv = Tiempo, within = Algoritmo, wid = Instancia,
```

```
                    return_aov = TRUE)
  return(anova[["ANOVA"]][["F"]])
}

# Genera distribución de estadísticos F con las permutaciones.
distribucion <- sapply(permutaciones, obtiene_F)

# Obtener valor p.
p <- (sum(distribucion > valor_observado) + 1) / (R + 1)
cat("ANOVA de una vía para muestras pareadas con permutaciones\n")
cat("p =", p, "\n\n")

# Análisis post-hoc.

# Función para calcular la media de las diferencias para dos columnas de una
# matriz de datos en formato ancho.
obtiene_media_difs <- function(df_ancho, columna_1, columna_2) {
  media <- mean(df_ancho[[columna_1]] - df_ancho[[columna_2]])
  return(media)
}

# Obtiene las las medias de las diferencias observadas
dif_obs_quick_bubble <- obtiene_media_difs(datos_anchos, "Quicksort",
                                           "Bubblesort")

dif_obs_quick_merge <- obtiene_media_difs(datos_anchos, "Quicksort",
                                          "Mergesort")

dif_obs_bubble_merge <- obtiene_media_difs(datos_anchos, "Bubblesort",
                                           "Mergesort")

# Obtiene las distribuciones de las medias de las diferencias permutadas
dist_medias_difs_quick_bubble <- sapply(permutaciones, obtiene_media_difs,
                                        "Quicksort", "Bubblesort")

dist_medias_difs_quick_merge <- sapply(permutaciones, obtiene_media_difs,
                                       "Quicksort", "Mergesort")

dist_medias_difs_bubble_merge <- sapply(permutaciones, obtiene_media_difs,
                                        "Bubblesort", "Mergesort")

# Obtener valores p.
num <- sum(abs(dist_medias_difs_quick_bubble) > abs(dif_obs_quick_bubble)) + 1
den <- R + 1
p_quick_bubble <- num / den

num <- sum(abs(dist_medias_difs_quick_merge) > abs(dif_obs_quick_merge)) + 1
den <- R + 1
p_quick_merge <- num / den

num <- sum(abs(dist_medias_difs_bubble_merge) > abs(dif_obs_bubble_merge)) + 1
den <- R + 1
p_bubble_merge <- num / den

cat("\n\n")
cat("Análisis post-hoc (permutaciones) para la diferencia de las medias\n")
cat("-----------------------------------------------------------\n")
cat("Valores p:\n")

```

```r
cat(sprintf("Quicksort - Bubblesort: %.3f\n", p_quick_bubble))
cat(sprintf("Quicksort - Mergesort: %.3f\n", p_quick_merge))
cat(sprintf("Bubblesort - Mergesort: %.3f\n", p_bubble_merge))

cat("\nDiferencias observadas:\n")
cat(sprintf("Quicksort - Bubblesort: %.3f\n", dif_obs_quick_bubble))
cat(sprintf("Quicksort - Mergesort: %.3f\n", dif_obs_quick_merge))
cat(sprintf("Bubblesort - Mergesort: %.3f\n", dif_obs_bubble_merge))
```

11.5 EJERCICIOS PROPUESTOS

1. En tus palabras, ¿qué es un estadístico robusto?
2. Explica, en tus propias palabras, las dos medidas de tendencia central robustas presentadas en este capítulo.
3. ¿Cómo funciona y para qué sirve la prueba de Yuen?
4. ¿Se pueden comparar dos medianas en vez de dos medias de grupos independientes?
5. 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.
6. ¿Existe una alternativa para medidas repetidas de la prueba de Yuen?
7. ¿Existe una alternativa robusta para hacer un análisis de varianza con medidas repetidas?
8. En tus palabras, ¿qué son las técnicas de remuestreo?
9. Explica si ¿podría considerarse que la prueba exacta de Fisher usa técnicas de remuestreo?
10. ¿En qué se parecen y en qué se diferencias las técnicas de bootstrapping y permutación?
11. En tus palabras, ¿qué es una simulación Monte Carlo?
12. ¿Cómo realizarías bootstrapping para determinar si la estatura media de estudiantes de Las Condes es igual a la estatura media de estudiantes de La Pintana?
13. ¿Cómo usarías Monte Carlo para verificar que un medicamento para bajar el colesterol funciona en un grupo de 30 personas escogidas al azar?
14. ¿Cómo usarías bootstrapping para analizar si tres algoritmos necesitan tiempos similares en procesar 25 instancias de prueba del problema de la mochila?
15. ¿Cómo usarías Monte Carlo para conocer si un medicamento para bajar la presión funciona al administrarlo a un grupo de personas hipertensas, comparando con un grupo de personas hipertensas recibiendo placebo y un grupo de control de personas no hipertensas?

REFERENCIAS

Amat Rodrigo, J. (2016). *Resampling: test de permutación, simulación de Monte Carlo y Bootstrapping.* Consultado el 31 de mayo de 2021, desde https://www.cienciadedatos.net/documentos/23_resampling_test_permutacion_simulacion_de_monte_carlo_bootstrapping

Hesterberg, T., Monaghan, S., Moore, D. S., Clipson, A. & Epstein, R. (2003). *Bootstrap Methods and Permutation Tests.* Consultado el 3 de junio de 2021, desde https://statweb.stanford.edu/~tibs/stat315a/Supplements/bootstrap.pdf

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