Estadística para Ciencia de los Datos

Tarea #1

Realice los siguientes ejercicios y envíe sus respuestas, en un notebook de google colab, a través de TEC Digital a más tardar el jueves 2 de julio a las 5:00 PM. No se aceptarán entregas tardías.

Desarrolle todos los ejercicios con el mayor nivel de detalle posible. Se espera que el desarrollo algebraico sea formal. No será válido únicamente dar respuestas en prosa a las preguntas.

Para la parte 1 de la tarea es válido entregar las respuestas como un archivo PDF que sea un escaneo de la solución en papel de los ejercicios. Los estudiantes son responsables por asegurar que el archivo enviado sea legible.

Para la parte 2 de la tarea se espera que el código pueda ejecutarse con visualizaciones apropiadas, sino no se asignará puntaje.

Parte 1

1. Una agencia de bienes raíces tiene una base de datos de propiedades vendidas en la cuál rastrearon la cantidad de días que duró en concretarse la venta, además de su precio de venta (ver tabla). Basado en los datos responda los siguientes apartados. Responda todos los puntos utilizando ecuaciones únicamente.

   - Si A es el evento de vender una casa en más de 90 días, estime la probabilidad de A (5 puntos)
   - Si B es el evento de vender una casa en menos de 50,000, estime la probablidad de B (5 puntos)
   - ¿Cuál es la probabilidad de que A y B ocurran juntos? (5 puntos)
   - Si una casa se define que tiene un precio de menos de $50,000, ¿cuál es la probabilidad que tarde 90 o menos días en venderse? (5 puntos)
   - ¿Se puede considerar que los eventos A y B son independientes? (Demuéstrelo matemáticamente) (10 puntos)

	Menos de 30 días	De 31 a 90	Más de 90	Total
Menos de $50,000	50	40	10	100
50,000 a 99,999	20	150	80	250
100,000 a 149,999	20	280	100	400
Más de 150,000	10	10	30	50
Total	100	480	220	800

1. Suponga que se desea analizar la relación entre tres variables aleatorias X_1, X_2 y X_3, para las cuales se han recabado los siguientes arreglos de $N = 3$ observaciones:

   - Calcule la varianza de cada variable aleatoria (muestre todos los pasos) (10 puntos)
   - Calcule la matriz de correlación de Pearson (muestre todos los pasos) (10 puntos)

$$h_1 = \begin{bmatrix} 3 & 15 & 21 \end{bmatrix}$$
$$h_2 = \begin{bmatrix} 1 & 5 & 6 \end{bmatrix}$$
$$h_3 = \begin{bmatrix} 13 & 7 & 3 \end{bmatrix}$$

R/

La parte 1 fue realizada en papel y se adjunta en el **.zip** de la tarea.

Parte 2

1. Muestreo de estaturas. Los estudiantes deberán programar dos funciones para crear muestras de N personas (observaciones) con una sola variable aleatoria: estatura. Para efectos prácticos, ambas funciones pueden retornar una simple lista con valores numéricos que representan la estatura en centímetros de diferentes personas. Para ambas funciones deberán mostrarse ejemplos de creación de muestras con al menos 100 observaciones y mostrarlas en un histograma. La cantidad de cubetas deberá limitarse a un máximo de 9. Las funciones a programar serán las siguientes:

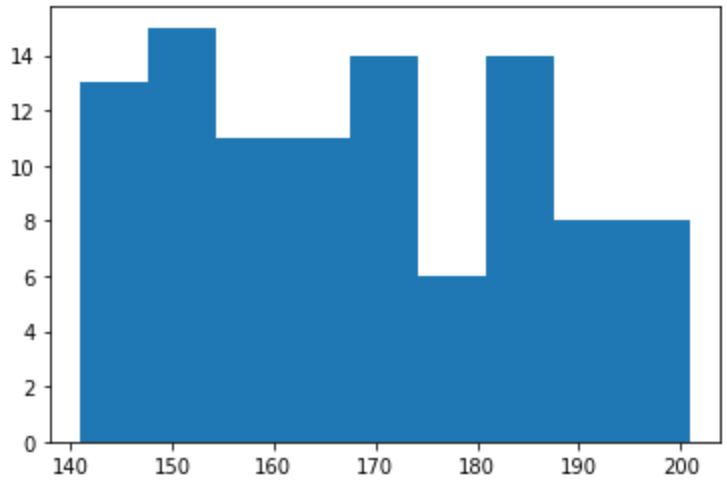
    - **Muestreo Uniforme**. Además de recibir la cantidad de observaciones a generar N, recibirá la estatura mínima y máxima por parámetro. Por ejemplo, si se pide una distribución uniforme entre 150 y 168, las estaturas entre ese rango deberán ser igual de probables de muestrear. (5 puntos)
    - **Muestreo Normal**. Además de recibir la cantidad de observaciones a generar N, recibirá la media y desviación estándar por parámetro. De esta forma, la función generará N observaciones provenientes de una distribución Gaussiana con la media y desviación estándar recibidas. (5 puntos)

1. **Evaluación de la función de verosimilitud normal**. Deberán proveer una función que reciba una de las **muestras** del punto 1 y evalúe la probabilidad que esa muestra provenga de una distribución normal con parámetros μ y σ (que deberán ser enviados a la función). Deberán mostrar ejemplos obtenidos bajo ambos esquemas de muestreo. Por ejemplo, si se crea una muestra normal centrada en $\mu_1 = 177$ con $\sigma_1 = 1$ y evaluamos la verosimilitud para $\mu = 176, \sigma = 3$ podría darnos una probabilidad alta. En contraposición, al crear una muestra uniforme entre 170 y 180 y evaluarla para esos mismos parámetros de verosimilitud, se esperaría que sea menos probable. Todo lo anterior debe reflejarse con ejemplos que muestren el correcto funcionamiento (10 puntos)
2. ¿Qué problema ocurre con el cálculo de la función de verosimilitud cuando aumentamos la cantidad de observaciones en varios órdenes de magnitud? y ¿Cual sería una forma de solucionar este problema? Programe una función con la posible solución y anote como comentarios las respuestas a las dos preguntas anteriores. De igual forma muestre un ejemplo de como invocar esta función (10 puntos)
3. Implemente una función para determinar los valores óptimos para maximizar la función de verosimilitud normal, según teoría vista en clase. Suponga que la cantidad de observaciones (N) siempre es alta. Igualmente se espera que se provean ejemplos en el código para mostrar el correcto funcionamiento (10 puntos)
4. Genere 10000 muestras de 30 observaciones cada una provenientes de una distribución normal con $\mu = 170$ y $\sigma = 1$. Calcule el valor óptimo de la varianza para maximizar la función de verosimilitud normal de cada muestra (estimador de la varianza sesgado) y luego obtenga el valor esperado de la varianza (promedio sobre las 10000 muestras). Compare este valor esperado con el valor de varianza usado para generar las muestras ($\sigma^2 = 1$) . Ejecute varias veces su código y responda ¿El valor esperado de la varianza aproxima de manera correcta la varianza original de la distribución o es necesario hacer alguna corrección en este caso para el cálculo del valor óptimo de la varianza? Justifique su respuesta en ambos casos y explique cual sería la corrección en caso afirmativo (10 puntos)

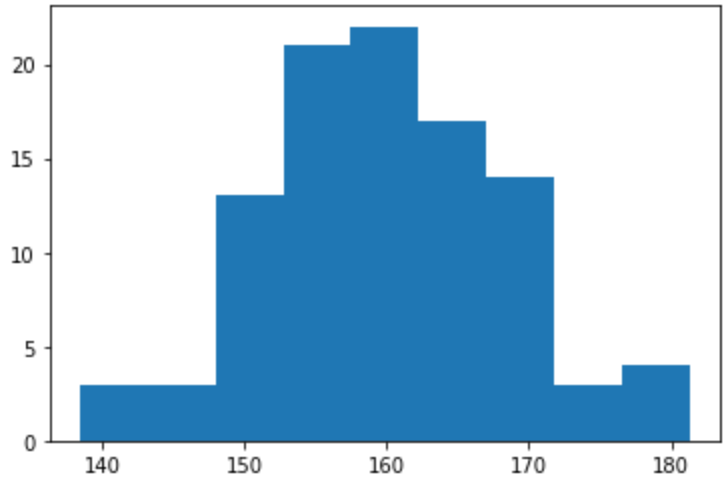
In [96]:
```
# EJERCICIO 2.1
import numpy as np
import matplotlib.pyplot as plt

'''
Returns a sample of random heights with uniform distribution

Parameters:
  n: number of samples (default: 100)
  min_height: sample's minimum height in cm
  max_height: sample's maximum height in cm

Returns:
  numpy vector with all samples in centimeters
'''
def get_height_samples_uniform(min_height=30, max_height=220, n=100):
  samples = np.random.uniform(min_height, max_height + 1, n)

  return samples

'''
Returns a sample of random heights with normal distribution

Parameters:
  n: number of samples (default: 100)
  mean: Mean of the sample's distribution
  std_dev: standard deviation of the sample's distribution

Returns:
  numpy vector with all samples in centimeters
'''
def get_height_samples_normal(mean=0, std_dev=0.1, n=100):
  samples = np.random.normal(mean, std_dev, n)

  return samples

'''
Plots a histogram from the samples vector.

Parameters:
  samples: Vector with all samples to plot
  bins: Number of bins to generate the histogram

Returns:
  None
'''
def plot_histogram(samples, bins=9):
  if bins > 9:
    bins = 9

  if bins <= 0:
    bins = 1

  plt.hist(samples, bins=bins)
  plt.show()

bins = 9

print(" 100 muestras aleatorias con distribución UNIFORME, con alturas entre 140c
m a 200cm")
uniform_samples = get_height_samples_uniform(min_height=140, max_height=200, n=10
```

```
0)
plot_histogram(uniform_samples, bins)

print(" 100 muestras aleatorias con distribución NORMAL, con media=160cm y desvia
ción estándar=8cm")
normal_samples = get_height_samples_normal(mean=160, std_dev=8, n=100)
plot_histogram(normal_samples, bins)
```

```
 100 muestras aleatorias con distribución UNIFORME, con alturas entre 140cm a 20
0cm
```




```
 100 muestras aleatorias con distribución NORMAL, con media=160cm y desviación e
stándar=8cm
```



Ejercicio 2.2

Evaluación de la función de verosimilitud normal. Deberán proveer una función que reciba una de las muestras del punto 1 y evalúe la probabilidad que esa muestra provenga de una distribución normal con parámetros μ y σ (que deberán ser enviados a la función). Deberán mostrar ejemplos obtenidos bajo ambos esquemas de muestreo. Por ejemplo, si se crea una muestra normal centrada en μ1=177 con σ1=1 y evaluamos la verosimilitud para μ=176,σ=3 podría darnos una probabilidad alta. En contraposición, al crear una muestra uniforme entre 170 y 180 y evaluarla para esos mismos parámetros de verosimilitud, se esperaría que sea menos probable. Todo lo anterior debe reflejarse con ejemplos que muestren el correcto funcionamiento (10 puntos)

In [30]:

```
# EJERCICIO 2.2

import math

'''
Calculates the value using a gaussian density function

Parameters:
  value: value to calculate
  mean: mean of the gaussian distribution
  std_dev: standard deviation of the gaussian distribution

Returns:
  number with the calculation on the gaussian density function
'''
def getGaussianValue(value, mean, std_dev):
  variance = std_dev ** 2
  norm = 1/(math.sqrt(2 * math.pi * variance))
  gaussian = norm * math.exp((-1/(2 * variance)) * (value - mean)**2 )

  return gaussian

'''
Checks the probability of likelihood from a sample with a gaussian density functi
on
This function can suffer the underflow issue with big samples

Parameters:
  samples: vector with all samples to compare
  mean: mean of the gaussian distribution
  std_dev: standard deviation of the gaussian distribution

Returns:
  number with the probability of likelihood
'''
def checkLikelihood(samples, mean, std_dev):
  likelihood = 1
  for x in samples:
    gaussian_x = getGaussianValue(x, mean, std_dev)
    likelihood *= gaussian_x

  return likelihood


likelihood_mean = 170
likelihood_std_dev = 5

print('Ejercicio 2.2')
print('Probabilidad de verosimilitud en diferentes muestras evaluado con media=17
0 y desviación estándar=5')

print('\n\n------\n\n')

print('Evaluando muestras de una distribución normal: \n\n')

print('1. Creados con media=160, desviación estándar=8 (la probabilidad debería s
er baja)')
normal_samples = get_height_samples_normal(mean=160, std_dev=8, n=100)
print(checkLikelihood(normal_samples, likelihood_mean, likelihood_std_dev))

print('\n\n2. Creados con media==171, desviación estándar=4 (la probabilidad debe
```

```
ría ser más alta)')
normal_samples = get_height_samples_normal(mean=171, std_dev=4, n=100)
print(checkLikelihood(normal_samples, likelihood_mean, likelihood_std_dev))

print('\n\n------\n\n')

print('Evaluando muestras de una distribución uniforme: \n\n')

print('1. Creados con altura mínima=140, altura máxima=185 (la probabilidad deber
ía ser mucho menor que comparando muestras de distribución normal)')
uniform_samples = get_height_samples_uniform(min_height=140, max_height=185, n=10
0)
print(checkLikelihood(uniform_samples, likelihood_mean, likelihood_std_dev))

print('\n\n2. Creados con altura mínima=160, altura máxima=175 (la probabilidad d
ebería ser mejor que la anterior)')
uniform_samples = get_height_samples_uniform(min_height=160, max_height=175, n=10
0)
print(checkLikelihood(uniform_samples, likelihood_mean, likelihood_std_dev))
```

```
Ejercicio 2.2
Probabilidad de verosimilitud en diferentes muestras evaluado con media=170 y de
sviación estándar=5


------


Evaluando muestras de una distribución normal:


1. Creados con media=160, desviación estándar=8 (la probabilidad debería ser baj
a)
7.000979927182151e-280


2. Creados con media==171, desviación estándar=4 (la probabilidad debería ser má
s alta)
7.721624171955924e-124


------


Evaluando muestras de una distribución uniforme:


1. Creados con altura mínima=140, altura máxima=185 (la probabilidad debería ser
mucho menor que comparando muestras de distribución normal)
2e-323


2. Creados con altura mínima=160, altura máxima=175 (la probabilidad debería ser
mejor que la anterior)
4.045574729114572e-133
```

Ejercicio 2.3

¿Qué problema ocurre con el cálculo de la función de verosimilitud cuando aumentamos la cantidad de observaciones en varios órdenes de magnitud? y ¿Cual sería una forma de solucionar este problema? Programe una función con la posible solución y anote como comentarios las respuestas a las dos preguntas anteriores. De igual forma muestre un ejemplo de como invocar esta función (10 puntos)

In [36]:

```
likelihood_mean = 170
likelihood_std_dev = 5
sample_mean = 170
sample_std_dev = 5

print('Evaluando verosimilutud con media=160, desviación=5 en muestras aleatorias
normales creadas con media=165, desviación=4 \n')
normal_samples_10 = get_height_samples_normal(mean=sample_mean, std_dev=sample_st
d_dev, n=10)
print("La probabilidad con 10 muestras es de: ", checkLikelihood(normal_samples_1
0, likelihood_mean, likelihood_std_dev))

normal_samples_100 = get_height_samples_normal(mean=sample_mean, std_dev=sample_s
td_dev, n=100)
print("La probabilidad con 100 muestras es de: ", checkLikelihood(normal_samples_
100, likelihood_mean, likelihood_std_dev))

normal_samples_1000 = get_height_samples_normal(mean=sample_mean, std_dev=sample_
std_dev, n=1000)
print("La probabilidad con 1000 muestras es de: ", checkLikelihood(normal_samples
_1000, likelihood_mean, likelihood_std_dev))

normal_samples_10000 = get_height_samples_normal(mean=sample_mean, std_dev=sample
_std_dev, n=10000)
print("La probabilidad con 10000 muestras es de: ", checkLikelihood(normal_sample
s_10000, likelihood_mean, likelihood_std_dev))

'''
Checks the probability of likelihood from a sample with a gaussian density functi
on
This function does not suffer the underflow issue with big samples because
it's using the natural logarithm to do the calculations

Parameters:
  samples: vector with all samples to compare
  mean: mean of the gaussian distribution
  std_dev: standard deviation of the gaussian distribution

Returns:
  number with the probability of likelihood
'''
def checkLikelihoodWithLog(samples, mean, std_dev):
  likelihood = 0
  for x in samples:
    gaussian_x = getGaussianValue(x, mean, std_dev)
    likelihood += math.log(gaussian_x)

  return likelihood

print('\n\n Ahora evaluando la verosimilutud utilizando logaritmo natural (mismos
parámetros): \n')
print("La probabilidad con las mismas 10 muestras es de: ", checkLikelihoodWithLo
g(normal_samples_10, likelihood_mean, likelihood_std_dev))
print("La probabilidad con las mismas 100 muestras es de: ", checkLikelihoodWithL
og(normal_samples_100, likelihood_mean, likelihood_std_dev))
print("La probabilidad con las mismas 1000 muestras es de: ", checkLikelihoodWith
Log(normal_samples_1000, likelihood_mean, likelihood_std_dev))
print("La probabilidad con las mismas 10000 muestras es de: ", checkLikelihoodWit
hLog(normal_samples_10000, likelihood_mean, likelihood_std_dev))
```

```
Evaluando verosimilutud con media=160, desviación=5 en muestras aleatorias norma
les creadas con media=165, desviación=4

La probabilidad con 10 muestras es de:  9.011479593528677e-125
La probabilidad con 100 muestras es de:  9.837894154327515e-136
La probabilidad con 1000 muestras es de:  0.0
La probabilidad con 10000 muestras es de:  0.0


 Ahora evaluando la verosimilutud utilizando logaritmo natural (mismos parámetro
s):

La probabilidad con las mismas 10 muestras es de:  -285.6246373492998
La probabilidad con las mismas 100 muestras es de:  -310.8653309677358
La probabilidad con las mismas 1000 muestras es de:  -3039.045486753261
La probabilidad con las mismas 10000 muestras es de:  -30194.53807663531
```

R/

Conforme aumenta la cantidad de observaciones se presenta el problema de *underflow*, que significa que computacionalmente la fracción decimal es tan pequeña que no se puede representar más y termina siendo 0.

La solución es evaluar la función de verosimilutud utilizando logaritmo natural para maximizar la función y facilitar los cálculos.

Como se presenta en los resultados anteriores, ya se puede calcular en muestras de 1.000 y 10.000 observaciones lo cual era imposible utilizando el método normal de verosimilitud sin usar logaritmo.

Ejercicio 2.4

Implemente una función para determinar los valores óptimos para maximizar la función de verosimilitud normal, según teoría vista en clase. Suponga que la cantidad de observaciones (N) siempre es alta. Igualmente se espera que se provean ejemplos en el código para mostrar el correcto funcionamiento (10 puntos)

In [85]:
```
'''
Calculates the optimum mean and standard deviation from a sample

Parameters:
  samples: vector with all samples to calculate the values
  bias: boolean to indicate if the variance estimator should include a bias.
        The recommendation is to include the bias only with big samples

Returns:
  tuple of (optimum mean, optimum standard deviation)
'''
def get_optimum_values(samples, bias=True):
  M = len(samples)

  mean_sum = 0
  for x in samples:
    mean_sum += x
  mean = mean_sum / M

  variance_sum = 0
  for x in samples:
    variance_sum += ((x - mean) ** 2)
  variance = variance_sum / (M if bias else (M - 1))

  return mean, math.sqrt(variance)

size_samples = 100
mean = 170
std_dev = 5

normal_samples = get_height_samples_normal(mean=mean, std_dev=std_dev, n=size_sam
ples)

optimum_mean, optimum_std_dev = get_optimum_values(normal_samples, bias=True)
_, nobias_optimum_std_dev = get_optimum_values(normal_samples, bias=False)

print('Con una muestra de 10.000 observaciones, creadas con media=170, desviación
=5, los valores optimos estimados son:')
print('\nMedia óptima: ', optimum_mean)
print('\nDesviación estándar óptima: ', optimum_std_dev)
print('\nDesviación estándar óptima (usando estimador de varianza sin sesgo): ',
nobias_optimum_std_dev)
```

```
Con una muestra de 10.000 observaciones, creadas con media=170, desviación=5, lo
s valores optimos estimados son:

Media óptima:  169.63108396156923

Desviación estándar óptima:  4.979644761691183

Desviación estándar óptima (usando estimador de varianza sin sesgo):  5.00473129
20570866
```

Ejercicio 2.5

Genere 10000 muestras de 30 observaciones cada una provenientes de una distribución normal con $\mu = 170$ y $\sigma = 1$. Calcule el valor óptimo de la varianza para maximizar la función de verosimilitud normal de cada muestra (estimador de la varianza sesgado) y luego obtenga el valor esperado de la varianza (promedio sobre las 10000 muestras). Compare este valor esperado con el valor de varianza usado para generar las muestras ($\sigma^2 = 1$) . Ejecute varias veces su código y responda ¿El valor esperado de la varianza aproxima de manera correcta la varianza original de la distribución o es necesario hacer alguna corrección en este caso para el cálculo del valor óptimo de la varianza? Justifique su respuesta en ambos casos y explique cual sería la corrección en caso afirmativo (10 puntos)

In [89]:

```
'''
Performs a test execution to compare the average variance obtained
from samples created with specified mean and standard deviation

Parameters:
  iterations: number of iterations to test
  n: Size of the samples
  mean: mean of the samples with normal distribution
  std_dev: mean of the samples with normal distribution
  bias: boolean to indicate if the variance estimator should include a bias.
        The recommendation is to include the bias only with big samples

Returns:
  number with the average variance from all iterations
'''
def eval_mean_variance(iterations, n, mean, std_dev, bias=True):
  variances = []
  for i in range(iterations):
    samples = get_height_samples_normal(mean=mean, std_dev=std_dev, n=n)
    optimum_mean, optimum_std_dev = get_optimum_values(samples, bias=bias)
    variances.append(optimum_std_dev ** 2)

  return np.mean(variances)

print('Ejecución #1 de 10.000 iteraciones con 30 observaciones, media=170, desvia
ción=1 (usando estimador de varianza sesgado)')
print('Varianza promedio: ', eval_mean_variance(iterations=10000, n=30, mean=170,
std_dev=1, bias=True))

print('\nEjecución #2 de 10.000 iteraciones con 30 observaciones, media=170, desv
iación=1 (usando estimador de varianza sesgado)')
print('Varianza promedio: ', eval_mean_variance(iterations=10000, n=30, mean=170,
std_dev=1, bias=True))

print('\nEjecución #3 de 10.000 iteraciones con 30 observaciones, media=170, desv
iación=1 (usando estimador de varianza sesgado)')
print('Varianza promedio: ', eval_mean_variance(iterations=1000, n=30, mean=170,
std_dev=1, bias=True))

print('\nEjecución #4 de 10.000 iteraciones con 1000 observaciones, media=170, de
sviación=1 (usando estimador de varianza sesgado)')
print('Varianza promedio: ', eval_mean_variance(iterations=10000, n=1000, mean=17
0, std_dev=1, bias=True))

print('\n\nAhora usando estimador de varianza sin sesgo')

print('\nEjecución #5 de 10.000 iteraciones con 30 observaciones, media=170, desv
iación=1 (usando estimador de varianza SIN sesgo)')
print('Varianza promedio: ', eval_mean_variance(iterations=10000, n=30, mean=170,
std_dev=1, bias=False))

print('\nEjecución #6 de 10.000 iteraciones con 30 observaciones, media=170, desv
iación=1 (usando estimador de varianza SIN sesgo)')
print('Varianza promedio: ', eval_mean_variance(iterations=10000, n=30, mean=170,
std_dev=1, bias=False))

print('\nEjecución #7 de 10.000 iteraciones con 30 observaciones, media=170, desv
iación=1 (usando estimador de varianza SIN sesgo)')
print('Varianza promedio: ', eval_mean_variance(iterations=10000, n=30, mean=170,
std_dev=1, bias=False))
```

```
print('\nEjecución #8 de 10.000 iteraciones con 1000 observaciones, media=170, de
sviación=1 (usando estimador de varianza SIN sesgo)')
print('Varianza promedio: ', eval_mean_variance(iterations=10000, n=1000, mean=17
0, std_dev=1, bias=False))
```

```
Ejecución #1 de 10.000 iteraciones con 30 observaciones, media=170, desviación=1
(usando estimador de varianza sesgado)
Varianza promedio:  0.9675525648412849

Ejecución #2 de 10.000 iteraciones con 30 observaciones, media=170, desviación=1
(usando estimador de varianza sesgado)
Varianza promedio:  0.9655483197601874

Ejecución #3 de 10.000 iteraciones con 30 observaciones, media=170, desviación=1
(usando estimador de varianza sesgado)
Varianza promedio:  0.9666944624130336

Ejecución #4 de 10.000 iteraciones con 1000 observaciones, media=170, desviación
=1 (usando estimador de varianza sesgado)
Varianza promedio:  0.9994400152861576


Ahora usando estimador de varianza sin sesgo

Ejecución #5 de 10.000 iteraciones con 30 observaciones, media=170, desviación=1
(usando estimador de varianza SIN sesgo)
Varianza promedio:  1.0000957158704022

Ejecución #6 de 10.000 iteraciones con 30 observaciones, media=170, desviación=1
(usando estimador de varianza SIN sesgo)
Varianza promedio:  1.002535412075098

Ejecución #7 de 10.000 iteraciones con 30 observaciones, media=170, desviación=1
(usando estimador de varianza SIN sesgo)
Varianza promedio:  0.9982685330832359

Ejecución #8 de 10.000 iteraciones con 1000 observaciones, media=170, desviación
=1 (usando estimador de varianza SIN sesgo)
Varianza promedio:  0.9991069288745901
```

R/

En las 3 primeras ejecuciones donde se utilizó el estimador de varianza sesgado, el valor de la varianza promedio obtenida no aproxima de mejor manera el utilizado para crear las muestras.

Al haber pocos datos (solo 30), este estimador se ajusta mucho a los pocos datos que hay y se deberia compensar quitando dicho sesgo. Hay que resaltar que si funcionaría si la cantidad de datos es muy grande. Por ejemplo, en la **ejecución #4**, se sigue utilizando el mismo estimador con sesgo pero con observaciones de 1000 elementos. Como se puede notar, la varianza ahora si aproxima casi por completo la utilizada inicialmente.

Para trabajar con pocos datos podemos utilizar el estimador sin sesgo que se adecua mejor. Podemos ver en las **ejecuciones #5, #6 y #7** que con observaciones de 30 datos, la varianza estimada obtenida despues de 10.000 iteraciones si se aproxima muy bien al utilizado en la creación de las muestras.

Así mismo, se realiza en la **ejecución #8** una prueba con el estimador sin sesgo en muestras de 1000 observaciones. Esta también se adecua de buena manera.