

Actividad 3: M2003B

Author: A. Ramirez-Morales (andres.ramirez@tec.mx)

Instrucciones:

- Active el kernel proveniente de Anaconda
- Complete las funciones donde vea líneas de código inconclusas
- Use comentarios para documentar de manera integral sus funciones
- Pruebe sus funciones con distintos parámetros
- Aumente las explicaciones en el Markdown y en el código
- Procure NO usar chatGPT ú otra tecnología similar, usted tiene la capacidad intelectual suficiente para resolverlo por usted mismo
- Use la documentación oficial de las librerías que se utilizan
- Se entrega un archivo PDF CANVAS como lo indique el profesor

```
In [1]: # cargar librerías básicas
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm, shapiro
```

1. Regresión lineal - método analítico

1.1 Cargar y visualizar los datos

```
In [3]: import pandas as pd
data = pd.read_csv('C:/Users/diana/Documents/linear_data.csv')
X = data['X']
Y = data['Y']
```

1.2 Modelo lineal:

$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i$$

Ejercicio: Demuestre que:

$$\hat{\beta}_1 = \frac{\sum_{i=1}^n x_i y_i - n \bar{x} \bar{y}}{\sum_{i=1}^n x_i^2 - n \bar{x}^2}$$

Ejercicio: Complete la siguiente función para obtener los estimadores de β_0 , σ , y β_1 .

```
In [4]: def simple_linear_regression(x, y):
    """
        Estima los parámetros de una regresión lineal simple ( $y = \beta_0 + \beta_1*x + \text{error}$ ).

        Parámetros:
        x -- array-like, variable independiente
        y -- array-like, variable dependiente

        Retorna:
        beta_0_estimador -- intersección (ordenada al origen)
        beta_1_estimador -- pendiente (coeficiente de regresión)
        sigma_estimador -- estimación de la desviación estándar de los errores

    """
    # convertir listas de entrada a arrays de numpy
    x = np.array(x)
    y = np.array(y)

    # calcular promedios de x, y
    x_mean = np.mean(x)
    y_mean = np.mean(y)

    # calcular numerador y denominador para beta_1
    numerator = np.sum((x - x_mean) * (y - y_mean))
    denominator = np.sum((x - x_mean) ** 2)

    # Calcular beta_1 (pendiente)
    beta_1_estimador = numerator / denominator

    # Calcular beta_0 (intersección)
    beta_0_estimador = y_mean - beta_1_estimador * x_mean

    # Calcular valores predichos
    y_hat = beta_0_estimador + beta_1_estimador * x

    # Calcular residuos
    residuals = y - y_hat

    # Estimar sigma (desviación estándar de los errores)
    n = len(x)
    sigma_estimador = np.sqrt(np.sum(residuals ** 2) / (n - 2))

    return beta_0_estimador, beta_1_estimador, sigma_estimador
```

Pruebe su función

```
In [5]: beta_0_estimador, beta_1_estimador, sigma_estimador = simple_linear_regression(X, Y)
print(f"Estimador beta_0 (intercept): {beta_0_estimador}")
print(f"Estimador beta_1 (slope): {beta_1_estimador}")
print(f"Estimador sigma (error): {sigma_estimador}")
```

```
Estimador beta_0 (intercept): 5.260594723979204
Estimador beta_1 (slope): -2.011575418787928
Estimador sigma (error): 3.977648904150111
```

1.3 Errores para los estimadores

Ejercicio: Escriba una función que calcule los errores de β_0, β_1

```
In [8]: def calculate_standard_errors(x, sigma_hat):
    """
        Calcula los errores estándar de los estimadores beta_0 y beta_1
        en una regresión lineal simple.

    Parámetros:
    x -- array-like, variable independiente
    sigma_hat -- estimador de la desviación estándar de los errores (residuos)

    Retorna:
    se_beta_0 -- error estándar del intercepto (beta_0)
    se_beta_1 -- error estándar de la pendiente (beta_1)

    """

    # calcular los valores medios
    x_mean = np.mean(x)
    n = len(X) # numero de observaciones

    # estandard errors para beta_0 y beta_1
    se_beta_1 = sigma_hat / np.sqrt(np.sum((np.array(x) - x_mean)**2))
    se_beta_0 = sigma_hat * np.sqrt(1/n + (x_mean**2)) / np.sum((np.array(x) - x_mean)**2)

    return se_beta_0, se_beta_1
```

```
In [9]: se_beta_0, se_beta_1 = calculate_standard_errors(X, sigma_estimador)
print(f"Error on beta_0: {se_beta_0}")
print(f"Error on beta_1: {se_beta_1}")
```

Error on beta_0: 0.2516997113776405
Error on beta_1: 0.010593434625272392

1.4 Distribuciones para los estimadores

Dado que los estimadores son no-cesgados y además por el TLC tienen distribuciones Gaussianas, es posible construir distribuciones para ellos.

Ejercicio: Escriba una función que obtenga dichas distribuciones usando los valores calculados en los ejercicios anteriores.

```
In [10]: def generate_distributions(mean1, std1, mean2, std2, num_samples=1000):
    """
        Genera tres distribuciones aleatorias:
        1. Una normal con media = mean1 y desviación estándar = std1.
        2. Una normal con media = mean2 y desviación estándar = std2.
        3. Una chi-cuadrada con nu grados de libertad.

    Parámetros:
    mean1, std1, mean2, std2 -- parámetros para las distribuciones
    num_samples -- número de muestras a generar
```

```

mean1 -- media de la primera normal
std1 -- desviación estándar de la primera normal
mean2 -- media de la segunda normal
std2 -- desviación estándar de la segunda normal
num_samples -- número de muestras a generar (default = 1000)

Retorna:
gaussian1 -- muestras de la primera normal
gaussian2 -- muestras de la segunda normal
chi2_dist -- muestras de la chi-cuadrada
"""

# generar Las dos distribuciones normales
gaussian1 = np.random.normal(mean1, std1, num_samples)
gaussian2 = np.random.normal(mean2, std2, num_samples)

# generar distribución chi-cuadrada [Sugerencia: usar np.random.chisquare()]
nu = num_samples # numero de grados de libertad (ejemplo: igual al número de m
chi2_dist = np.random.chisquare(nu, num_samples)

return gaussian1, gaussian2, chi2_dist

```

Ejecute el código y haga gráficas de sus las distribuciones de los estimadores.

```

In [11]: import numpy as np
import matplotlib.pyplot as plt

# --- generar distribuciones ---
def generate_distributions(mean1, std1, mean2, std2, num_samples=1000):
    """
    Genera tres distribuciones aleatorias:
    1. Una normal con media = mean1 y desviación estándar = std1.
    2. Una normal con media = mean2 y desviación estándar = std2.
    3. Una chi-cuadrada con nu grados de libertad.
    """
    gaussian1 = np.random.normal(mean1, std1, num_samples)
    gaussian2 = np.random.normal(mean2, std2, num_samples)
    nu = 5 # grados de libertad fijos (puedes cambiarlo)
    chi2_dist = np.random.chisquare(nu, num_samples)

    return gaussian1, gaussian2, chi2_dist

# --- Generar distribuciones ---
g1, g2, chi2 = generate_distributions(mean1=0, std1=1, mean2=2, std2=0.5, num_sampl

# --- Graficar histogramas ---
plt.figure(figsize=(12, 6))

# Normal 1
plt.subplot(1, 3, 1)
plt.hist(g1, bins=30, density=True, alpha=0.7, color='C0')
plt.title("Distribución Normal 1")
plt.xlabel("Valor")
plt.ylabel("Densidad")

```

```
# Normal 2
plt.subplot(1, 3, 2)
plt.hist(g2, bins=30, density=True, alpha=0.7, color='C1')
plt.title("Distribución Normal 2")
plt.xlabel("Valor")
plt.ylabel("Densidad")

# Chi-cuadrada
plt.subplot(1, 3, 3)
plt.hist(chi2, bins=30, density=True, alpha=0.7, color='C2')
plt.title("Distribución Chi-cuadrada")
plt.xlabel("Valor")
plt.ylabel("Densidad")

plt.tight_layout()
plt.show()
```

