Actividad 1: M2003B

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

Equipo Frijoles Azufrados

- Joshua Alexander Chaidez Ochoa - A01742969
- Oliver Arturo Casas Pontanillo - A01645764
- Erik Abrahm Jajan Díaz - A01644648

Instrucciones:

- 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 [13]:
```
# cargar librerías básicas
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm, shapiro
import pandas as pd
```

In [14]:
```
# Inserte su código aquí
numero_de_lanzamientos = 100000 # número de eventos a simular

data = np.random.randint(1, 7, size=numero_de_lanzamientos) # lanzamientos d

# aqui graficamos
plt.hist(data, bins=np.arange(0.5, 7.5, 1), color='red', edgecolor='pink', d
plt.title("Experimento de dado")
plt.xlabel("Cara de dado")
plt.ylabel("Frecuencia")
plt.show()
```

Los resultados muestran una distribucion aproximada a una uniforme (dicha cercania o semejanza aumenta conforme aumentamos el numero de lanzamientos) la cual es anticipada debido a que se trata de un evento aleatorio donde cada cara del dado tiene la misma probabilidad de resultar selecccionada en cada evento de lanzamiento.

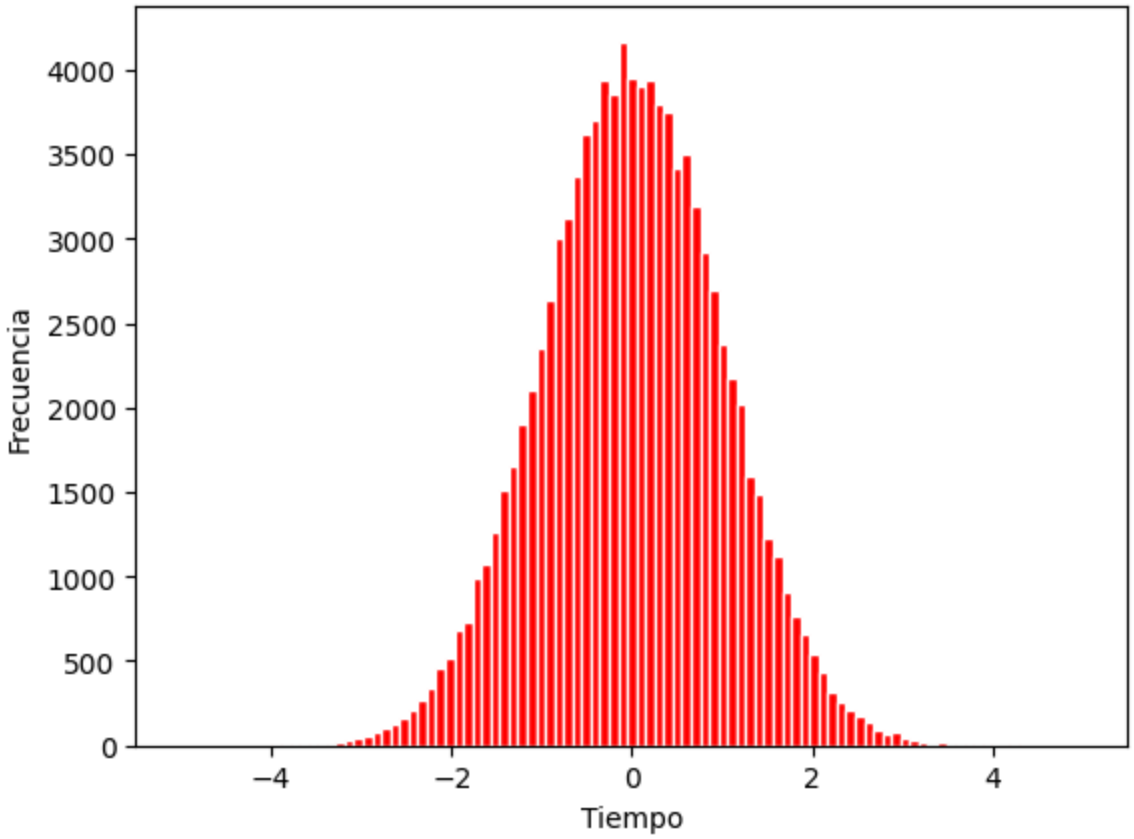
0.2. Distribuciones Gaussianas (normales)

Ejercicio:

- Escriba código para generar una distribución Gaussiana. Explique los parámetros de dicha función de probabilidad.
- Repita esta actividad para la distribución de Poisson y Bernoilli

In [15]:
```
# Parámetros de la distribución
media = 0 # media centro de la distribución
desviacion = 1 # desviación estándar ancho de la distribución o la dispersió
n_muestras = 100000 # cantidad de datos a generar y graficar posteriormente

data = np.random.normal(loc=media, scale=desviacion, size=n_muestras)

# aqui graficamos
plt.hist(data, bins=np.linspace(-5, 5, 100), color='red', edgecolor='white')
plt.title("")
plt.xlabel("Tiempo")
```

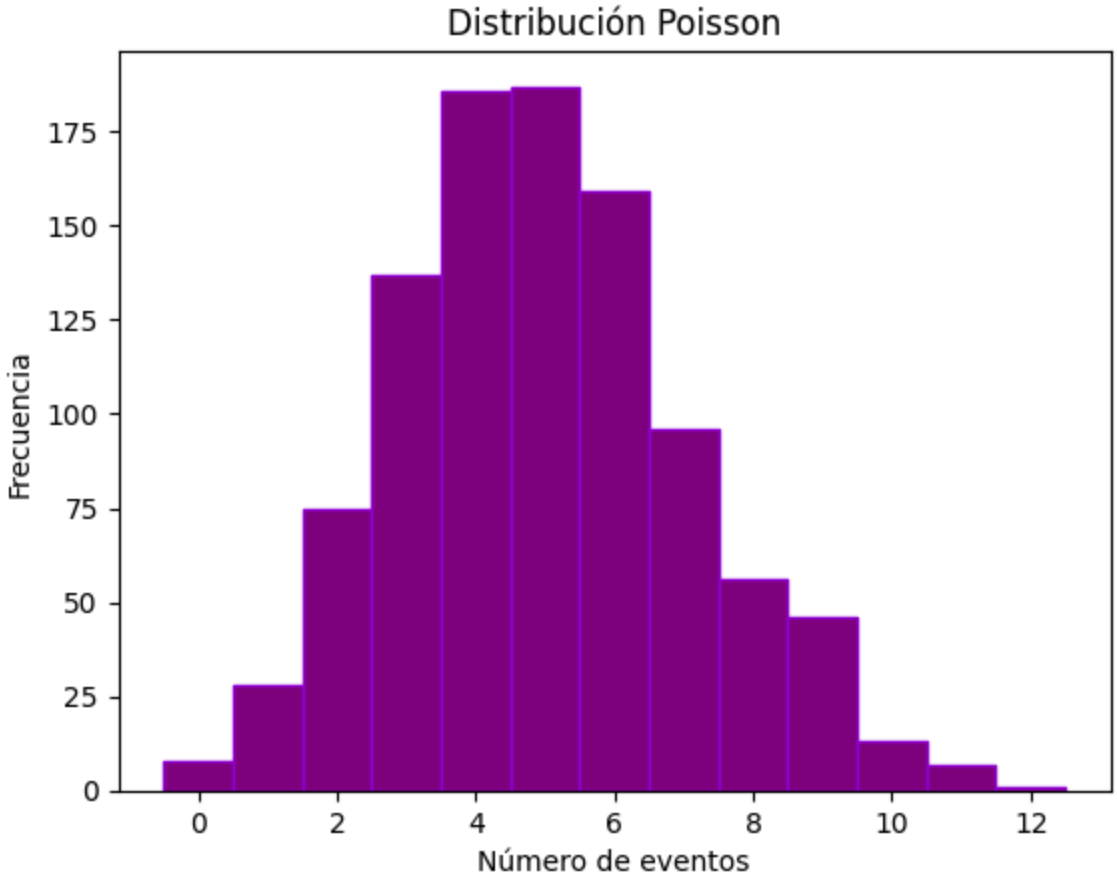
```
plt.ylabel("Frecuencia")
plt.show()
```



El código genera una distribución normal estándar. Con una media de 0 y una desviación de 1, crea 100,000 puntos de datos que luego se visualizan en un histograma. Este gráfico muestra la frecuencia de los datos, con las barras más altas concentradas en el centro, lo que representa la forma de campana de Gauss.

Poisson

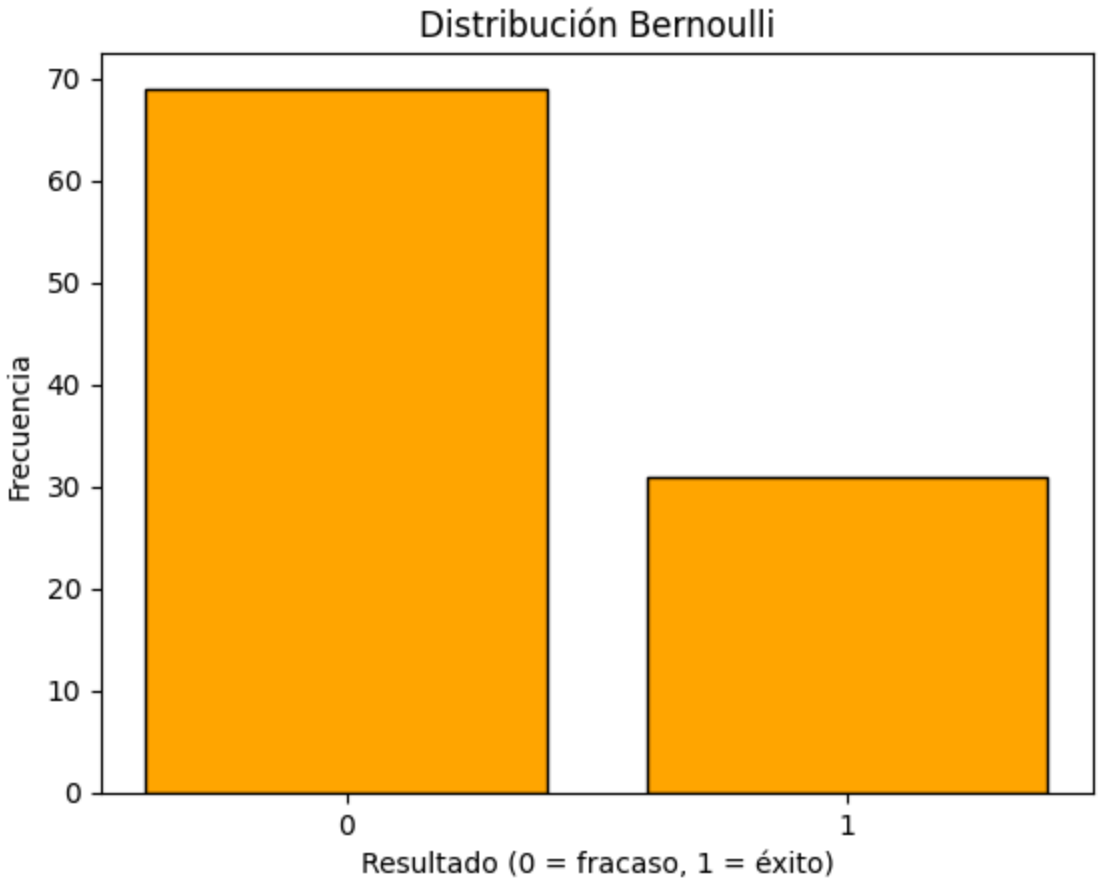
In [16]:
```
lambda_poisson = 5  # Lambda de la distribucion Poisson
n_muestras = 1000

data = np.random.poisson(lam=lambda_poisson, size=n_muestras)
bins = np.arange(data.min(), data.max() + 1) - 0.5  # centrado en enteros

plt.hist(data, bins=bins, color='purple', edgecolor='darkviolet')
plt.title("Distribución Poisson")
plt.xlabel("Número de eventos")
plt.ylabel("Frecuencia")
plt.show()
```



El código simula y grafica una distribución de Poisson con un valor de lambda igual a 5. Genera 1,000 puntos de datos que representan el número de eventos que ocurren en un intervalo fijo de tiempo o espacio. El histograma resultante, con barras moradas, muestra la frecuencia con la que se observa cada número de eventos. El pico de la distribución se encuentra cerca del valor de lambda (5), y las frecuencias disminuyen simétricamente a medida que se alejan de ese valor.

Bernoilli

In [17]:
```
p = 0.3  # Probabilidades de éxito
n_muestras = 100 # numero de intentos generados

data = np.random.binomial(n=1, p=p, size=n_muestras)

plt.hist(data, bins=[-0.5, 0.5, 1.5], color='orange', edgecolor='black', rwi
plt.title("Distribución Bernoulli")
plt.xlabel("Resultado (0 = fracaso, 1 = éxito)")
plt.ylabel("Frecuencia")
plt.xticks([0, 1])
plt.show()
```



El código que proporcionaste simula y grafica una distribución de Bernoulli con una probabilidad de éxito (p) de 0.3. Genera 100 resultados aleatorios, donde 1 representa el éxito y 0 el fracaso. El histograma resultante, con barras naranjas, muestra la frecuencia de estos dos posibles resultados. La barra del valor 1 (éxito) es más baja que la barra del valor 0 (fracaso), lo cual refleja la probabilidad de 0.3.

0.3. Crear, guardar y cargar datos

Ejercicio:

- Crear, guardar y cargar dichos datos
- Asegúrese que los datos coinciden con lo que espera
- Siga los "leads" en el código siguiente

In [18]:
```
# aqui simulamos los datos, por ejemplo gaussianos (como en el ejemplo anter
n_muestras = 1000
media = 0
desviacion = 1

data = np.random.normal(loc=media, scale=desviacion, size=n_muestras)

# aqui creamos un pandas data frame para guardar los datos
df = pd.DataFrame({
        'data': data,
    })
```

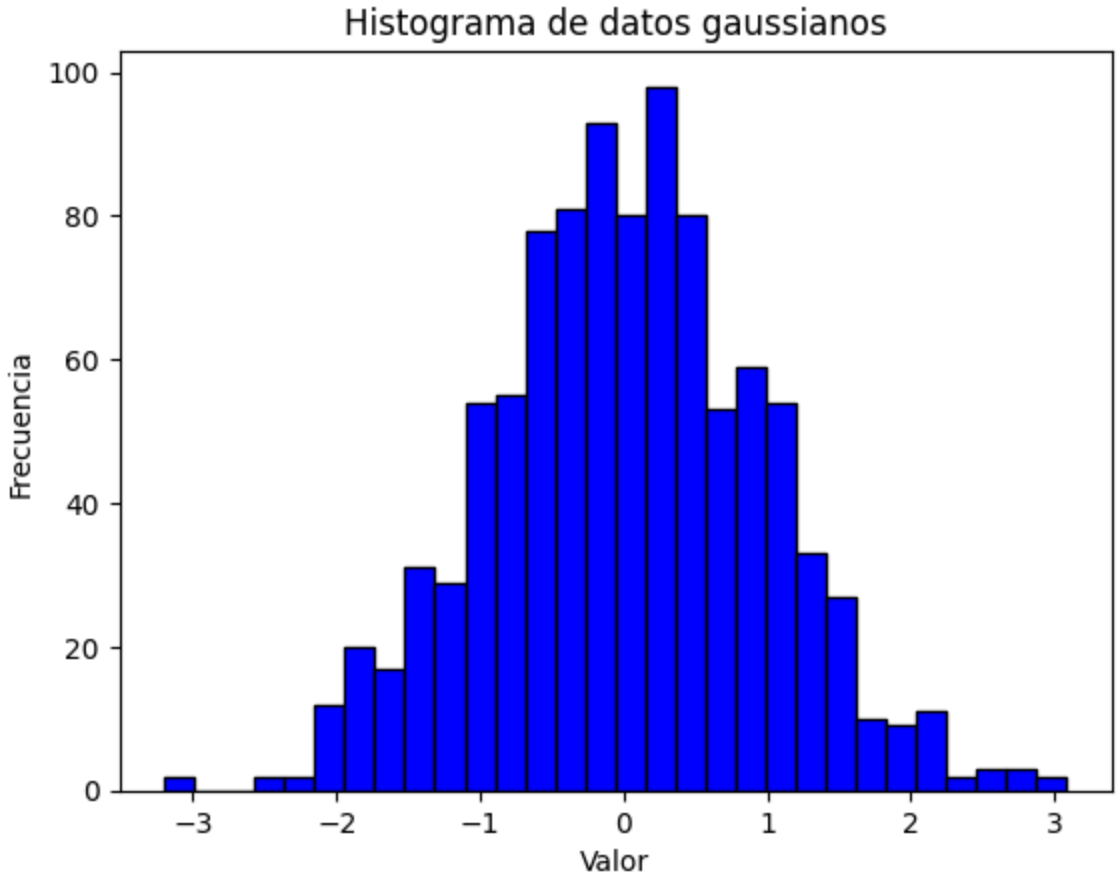
```
# guardamos los datos en un archivo CSV
nombre_archivo = "datos_gaussianos.csv"
df.to_csv(nombre_archivo, index=False)
print(f"datos guardados en {nombre_archivo}")

# ahora cargamos los datos
datos_cargados = pd.read_csv("datos_gaussianos.csv")
```

```
datos guardados en datos_gaussianos.csv
```

Ahora grafiquemos los datos

Compruebe que los datos están de acuerdo con lo que esperabamos. Para lo anterior, calculemos la media y la desviación estándar usando funciones de numpy y a mano. Compare sus respuestas.

In [19]:
```
# Datos
datos_cargados = pd.read_csv(nombre_archivo)['data']

# Graficar histograma de los datos cargados
plt.hist(datos_cargados, bins=30, color='blue', edgecolor='black')
plt.title("Histograma de datos gaussianos")
plt.xlabel("Valor")
plt.ylabel("Frecuencia")
plt.show()

# Calculo NumPy
media_np = np.mean(datos_cargados)
desv_np = np.std(datos_cargados)

# Resultados
print(f"Media con NumPy: {media_np}")
print(f"Desviación estándar con NumPy: {desv_np}")
```




```
Media con NumPy: 0.010027629397233322
Desviación estándar con NumPy: 0.9448157149550453
```

Los resultados que dieron son un valor cercano al valor que se esperaba, media = 0 y desviación estandar = 1

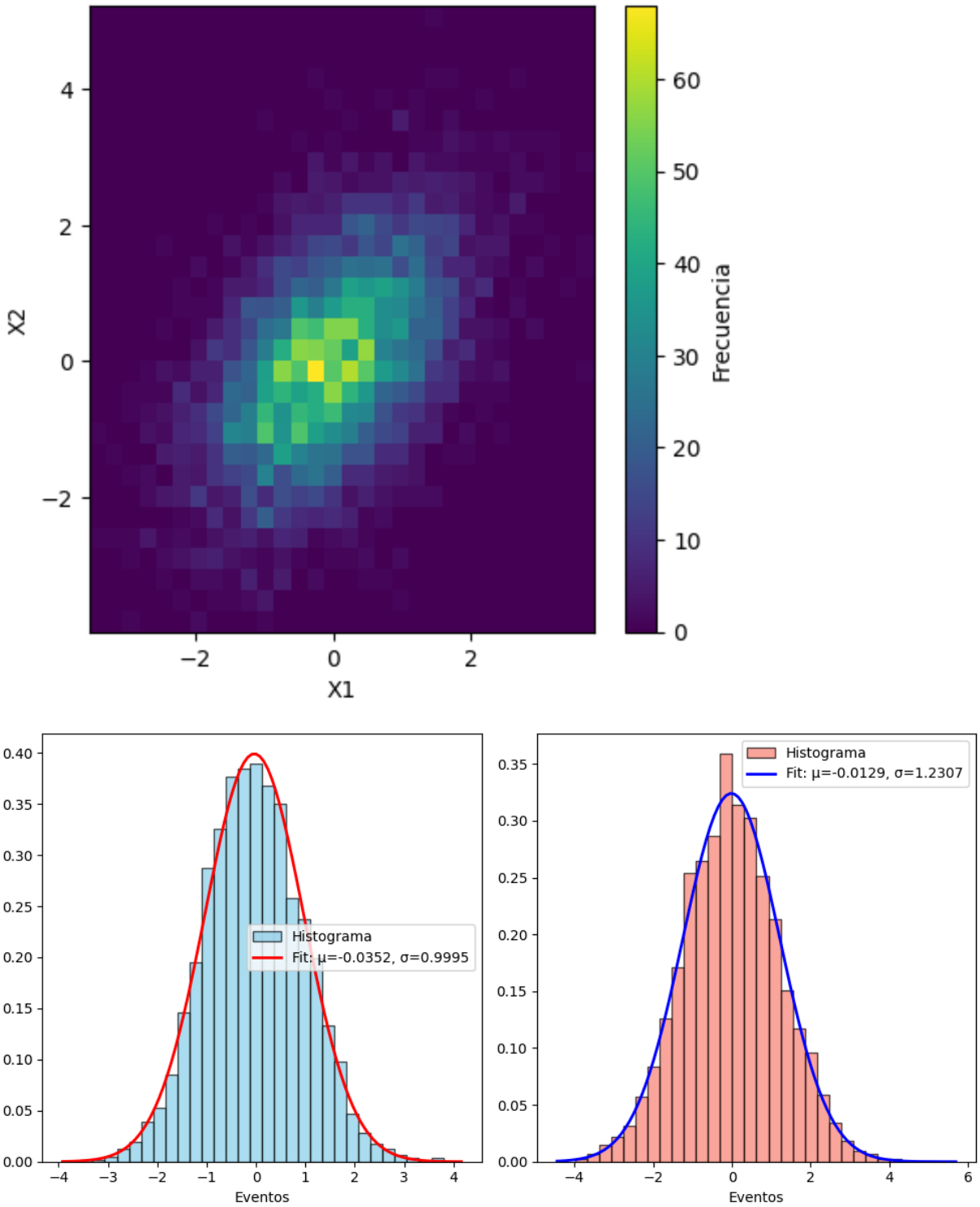
LATEX

0.4 Probabilidad (densidad) multivariada

- Genere datos en dos dimensiones distribuidos siguiendo una probabilidad Gaussiana
- Grafique los datos usando un histograma en dos dimensiones
- Guarde los datos en un archivo csv
- Cargue los datos y grafique cada una de las variables en un histograma sencillo
- Realizar un ajuste para encontrar la PDF analítica y graficar

In [20]:
```
# generar los datos
n = 5000
mean = [0, 0]
cov = [[1.0, 0.6], [0.6, 1.5]]
data = np.random.multivariate_normal(mean, cov, size=n)

# aqui graficamos
plt.figure(figsize=(5, 5))
```

```
plt.hist2d(data[:, 0], data[:, 1], bins=30, cmap='viridis')
plt.colorbar(label='Frecuencia')
plt.xlabel('X1')
plt.ylabel('X2')
plt.show()

# guardar datos
df = pd.DataFrame(data, columns=['X1', 'X2'])
df.to_csv('gaussian_2d_data.csv', index=False)

# cargar datos
df_cargado = pd.read_csv('gaussian_2d_data.csv')

plt.figure(figsize=(10, 5))

# primera variable de histograma y fit
plt.subplot(1, 2, 1)
x1 = df_cargado['X1'].values
# hacer el fit
mu, std = norm.fit(x1)
plt.hist(x1, bins=30, density=True, color='skyblue', edgecolor='black', alph
         label='Histograma')

# graficar el ajuste
xmin, xmax = plt.xlim()
x_fit = np.linspace(xmin, xmax, 100)
y_fit = norm.pdf(x_fit, mu, std)
plt.plot(x_fit, y_fit, 'r-', linewidth=2, label=f'Fit: μ={mu:.4f}, σ={std:.4
plt.xlabel('Eventos')
plt.legend()

# Segunda variable graficada
plt.subplot(1, 2, 2)
x2 = df_cargado['X2'].values
mu, std = norm.fit(x2)
plt.hist(x2, bins=30, density=True, color='salmon', edgecolor='black', alpha
         label='Histograma')

# graficar el ajuste
ymin, ymax = plt.xlim()
y_fit = np.linspace(ymin, ymax, 100)
y_pdf = norm.pdf(y_fit, mu, std)
plt.plot(y_fit, y_pdf, 'b-', linewidth=2, label=f'Fit: μ={mu:.4f}, σ={std:.4
plt.xlabel('Eventos')
plt.legend()

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



El código simula y visualiza una distribución gaussiana multivariada, mostrando su comportamiento en un plano bidimensional. El primer gráfico, un histograma 2D, revela una correlación positiva entre las variables X1 y X2, evidenciada por la forma elíptica de la densidad de datos. Posteriormente, el análisis individual de cada variable confirma que ambas siguen una distribución normal, lo cual se demuestra al superponer una curva de ajuste a sus respectivos histogramas unidimensionales.

1. Teorema del límite central

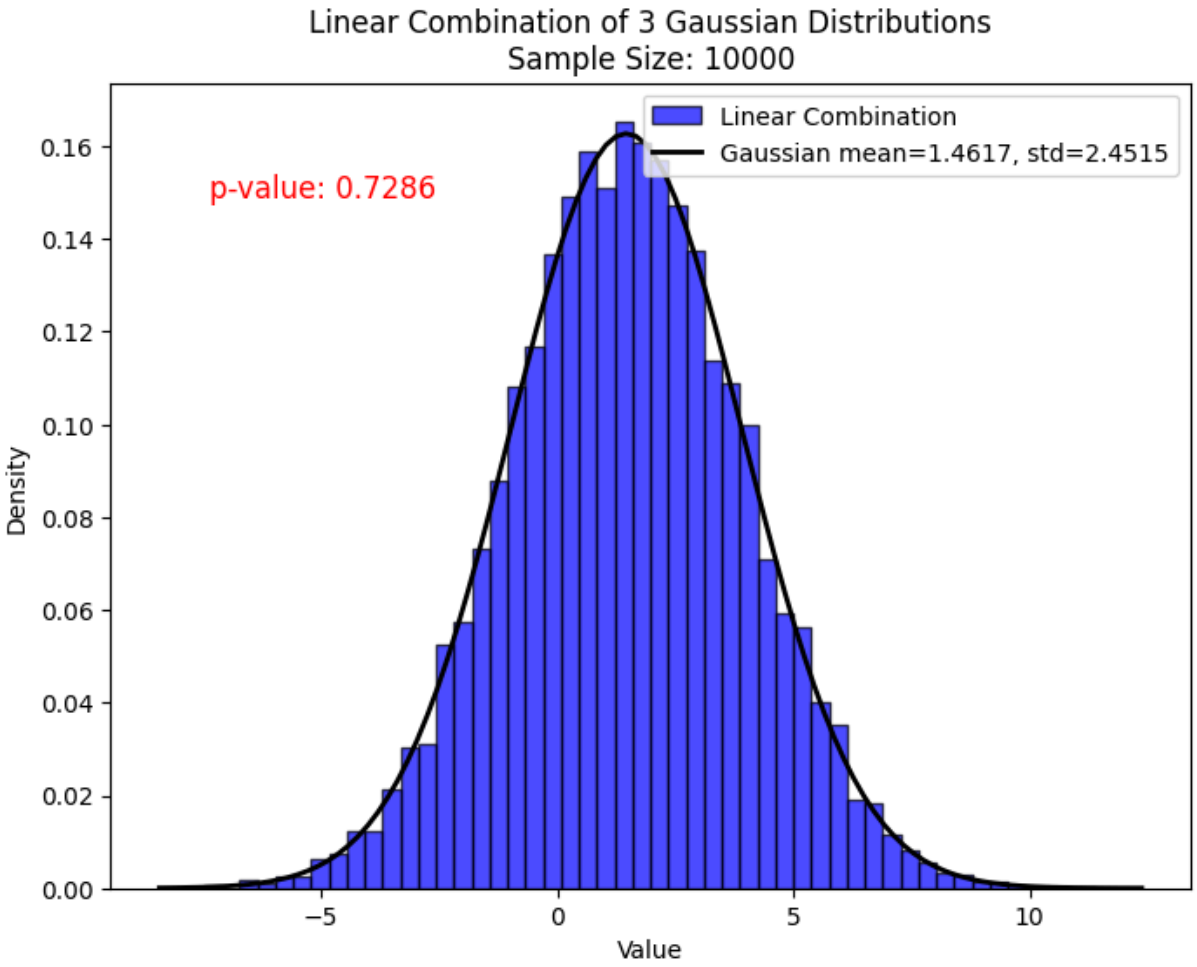
1.1 Distribuciones Gaussianas

Ejercicio: Muestre numéricamente que la combinación lineal de distribuciones Gaussianas es una distribución Gaussiana. Asegúrese de que esta nueva distribución en efecto sea Gaussiana con una prueba de Shapiro-Wilk, es decir encuentre un p-valor a aceptar que es Gaussiana.

In [21]:
```
def sum_gaussians_tlc(num_samples, n_gaussians):
    """



    """
    # generar distribuciones gausianas independientes
    means = np.linspace(-1.0, 1.0, n_gaussians)
    std_devs = np.random.uniform(0.5, 1.5, size=n_gaussians)
    coefficients = np.random.normal(0, 1, size=n_gaussians)

    # creando una lista vacia para guardar las muestras
    gaussians = []

    # ciclo de cada distribucion gaussiana
    for i in range(n_gaussians):
        samples = np.random.normal(loc=means[i], scale=std_devs[i], size=num
        gaussians.append(samples)

    # combinacion lineal de los gausianos
    linear_combination = np.zeros(num_samples)
    for i in range(n_gaussians):
        linear_combination += coefficients[i] * gaussians[i]

    #fiteamos una distribucion gausiana a la combinacion linear
    mu, std = norm.fit(linear_combination)

    # graficamos el histograma de la combinacion linear
    plt.figure(figsize=(8, 6))
    count, bins, _ = plt.hist(linear_combination, bins=50, density=True, alp

    # graficamos el fit gaussiano
    xmin, xmax = plt.xlim()  # los limites de x para graficar el fit
    x = np.linspace(xmin, xmax, 100)
    p = norm.pdf(x, mu, std)   # el fit de la distribucion gausiana

    # escalamos el fit a la altura maxima del histograma para alinearlo
    plt.plot(x, p, 'k', linewidth=2, label=f'Gaussian mean={mu:.4f}, std={st

    # shapiro wilk para ver la normalidad
    stat, p_value = shapiro(linear_combination[:5000])

    # pvalue
    plt.text(xmin + (xmax - xmin) * 0.05, max(count) * 0.9, f'p-value: {p_va

    # titulo y leyenda
    plt.title(f'Linear Combination of {n_gaussians} Gaussian Distributions\r
```

```
    plt.xlabel('Value')
    plt.ylabel('Density')
    plt.legend()
    plt.show()

    if p_value > 0.05:
        print(f"p-value = {p_value:.4f} > 0.05 Gaussiana")
    else:
        print(f"p-value = {p_value:.4f} ≤ 0.05 No Gaussiana")
```

In [22]:
```
sum_gaussians_tlc(10000, 3)
```




```
p-value = 0.7286 > 0.05 Gaussiana
```

El código ilustra el Teorema del Límite Central (TLC). Se simula una combinación de distribuciones gaussianas, y el histograma resultante demuestra que la suma converge a una distribución normal. Este resultado se valida con una prueba de normalidad, donde un p-valor superior a 0.05 confirma que la distribución resultante es gaussiana.

1.2 Distribuciones uniformes

Ejercicio: Repita el ejercicio anterior con distribuciones uniformes finitas. Es decir, el teorema establece que la distribución de la combinación lineal de un gran número de

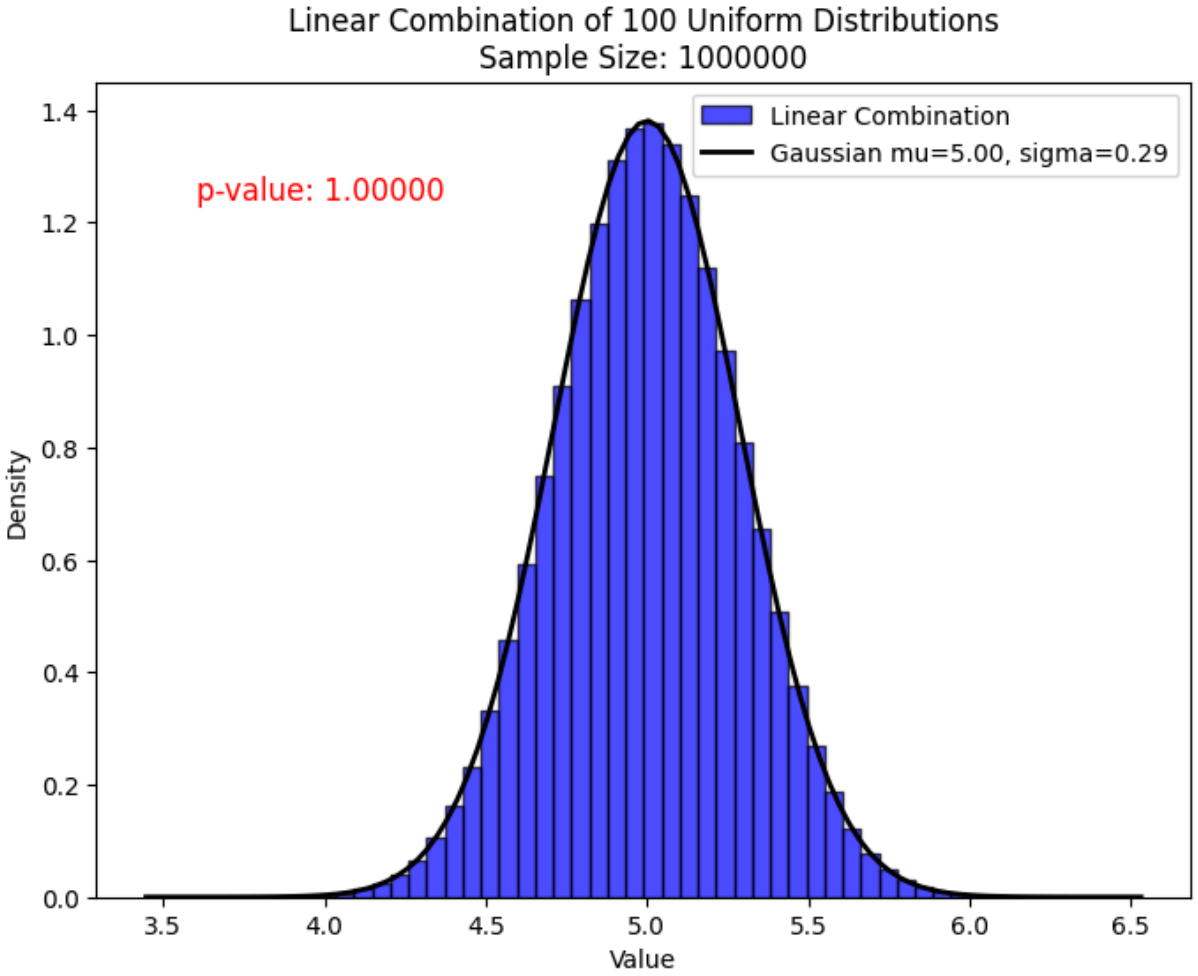
variables aleatorias independientes e idénticamente distribuidas se aproximará a una distribución normal. Sume cuando menos 50 distribuciones.

In [23]:
```
def sum_uniforms_tlc(num_samples=1000000, n_uniforms=100):
    """



    """
    # generar distribuciones uniformes independientes
    lower_bounds = np.zeros(n_uniforms)
    upper_bounds = np.ones(n_uniforms)
    coefficients = np.ones(n_uniforms) / np.sqrt(n_uniforms)

    uniforms = []

    # ciclo de cada distribucion uniforme
    for i in range(n_uniforms):
        # generamos muestras al azar para la distribucion
        current_samples = np.random.uniform(low=lower_bounds[i], high=upper_
        # agregamos estas muestras a la lista
        uniforms.append(current_samples)

    # combinacion linear de las distribuciones uniformes
    linear_combination = np.zeros(num_samples)
    for i in range(n_uniforms):
        linear_combination += coefficients[i] * uniforms[i]

    # fiteamos una distribucion gausiana a la combinacion linear
    mu, std = norm.fit(linear_combination)

    # graficamos el histograma de la combinacion linear
    plt.figure(figsize=(8, 6))
    count, bins, _ = plt.hist(linear_combination, bins=50, density=True, alp

    # graficamos el fit gausiano
    xmin, xmax = plt.xlim()  # Get the limits of the x-axis for plotting the
    x = np.linspace(xmin, xmax, 100)
    p = norm.pdf(x, mu, std)

    # escalamos el fit a la altura maxima del histograma para alinearlo
    plt.plot(x, p, 'k', linewidth=2, label=f'Gaussian mu={mu:.2f}, sigma={st

    # shapiro wil para ver la normalidad
    stat, p_value = shapiro(linear_combination)

    # pvalue
    plt.text(xmin + (xmax - xmin) * 0.05, max(count) * 0.9, f'p-value: {p_va

    # titulo y leyenda
    plt.title(f'Linear Combination of {n_uniforms} Uniform Distributions\nSa
    plt.xlabel('Value')
    plt.ylabel('Density')
    plt.legend()

    # graficamos
```

```
    plt.show()

    if p_value > 0.05:
        print(f"p-value = {p_value:.5f} > 0.05")
    else:
        print(f"p-value = {p_value:.5f} ≤ 0.05")
```

In [24]:
```
sum_uniforms_tlc(num_samples=1000000, n_uniforms=100)
```

```
/opt/anaconda3/envs/mv-tec/lib/python3.12/site-packages/scipy/stats/_axis_nan_policy.py:573: UserWarning: scipy.stats.shapiro: For N > 5000, computed p-value may not be accurate. Current N is 1000000.
  res = hypotest_fun_out(*samples, **kwds)
```




```
p-value = 1.00000 > 0.05
```

El código ilustra el Teorema del Límite Central (TLC). Se simula una combinación de distribuciones uniformes, y el histograma resultante demuestra que la suma de estas converge a una distribución normal. Este resultado se valida con una prueba de normalidad, donde un p-valor superior a 0.05 confirma que la distribución resultante es gaussiana, estableciendo una validación empírica del teorema.