Actividad 1: M2003B

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

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 [40]:
```python
# cargar librerías básicas
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm, shapiro
import pandas as pd
```

The Shapiro-Wilk test tests the null hypothesis that the data was drawn from a normal distribution.

0. Introducción

0.1. Simulación del experimento de lanzamientos de dado

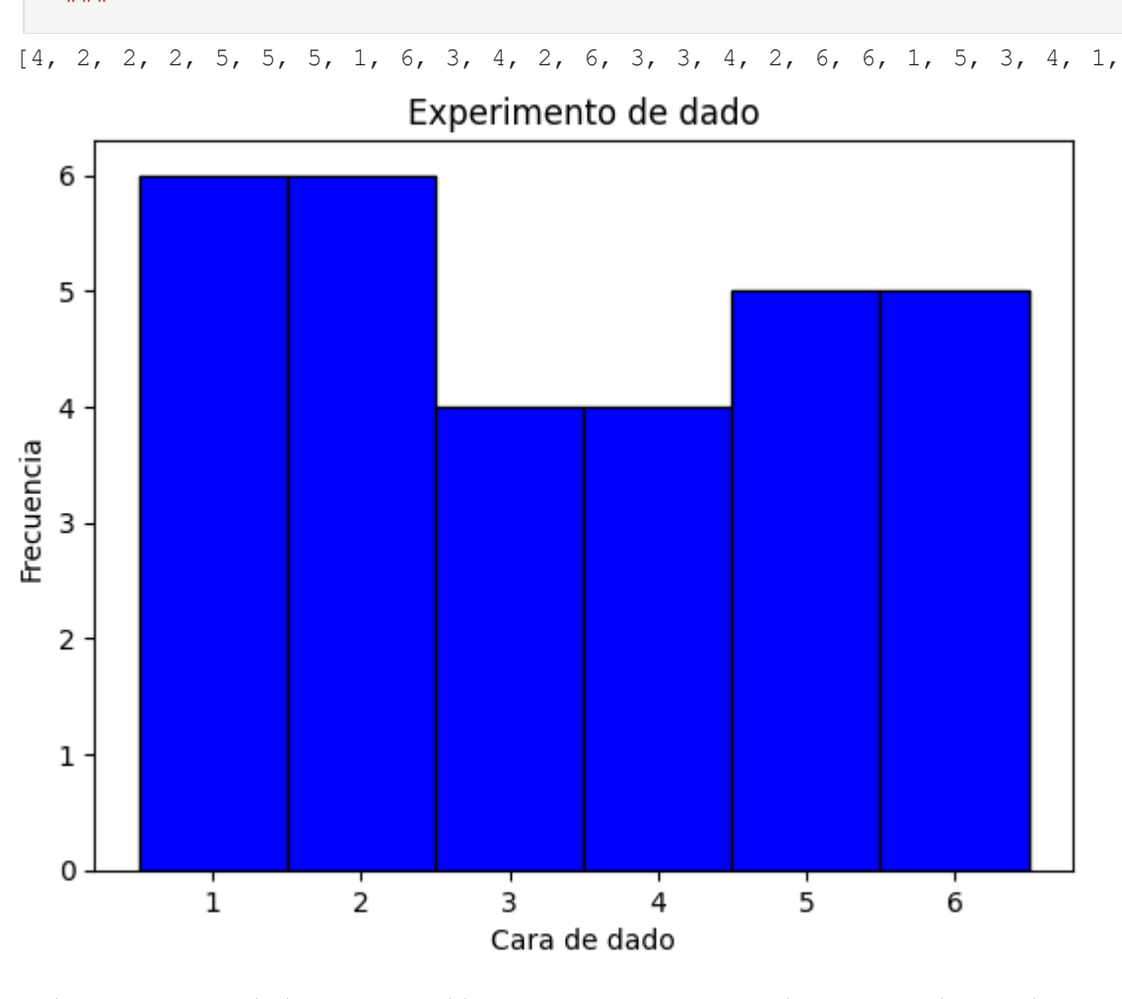
Ejercicio:

- Definir el número de eventos a simular (haré 30 eventos)
- Definir el tipo de dado (dado normal de 6 lados)
- Simular los datos con una distribución adecuada

In [41]:
```python
import random
```

In [42]:
```python
# Inserte su código aquí
#----------------------------
""" Funcion de dado """
# siendo N el numero de elementos de tu lista(numero de tiradas)
def dado(n):
    result=[]
    for i in range(0, n):
      result.append(random.randint(1, 6))
    return result

#crear las 30 tiradas
data= dado(30)#para graficar
print(data)
#como puso bins=bins no queria cambiar el codigo dado so:
bins= [x + 0.5 for x in range(0, 7)]#para centrarlos
#----------------------------
# aqui graficamos
plt.hist(data, bins=bins, color='blue', edgecolor='black')
plt.title("Experimento de dado")
plt.xlabel("Cara de dado")
plt.ylabel("Frecuencia")
plt.show()




"""
Referencias

  https://es.stackoverflow.com/questions/99638/simulador-de-lanzamiento-de-dado-en-python
  https://configuroweb.com/simulador-de-dados-en-python

  """
```

```
[4, 2, 2, 2, 5, 5, 5, 1, 6, 3, 4, 2, 6, 3, 3, 4, 2, 6, 6, 1, 5, 3, 4, 1, 1, 1, 5, 1, 6, 2]
```



Out[42]:
```
'\nReferencias\n\n  https://es.stackoverflow.com/questions/99638/simulador-de-lanzamiento-de-dado-en-python\n  https://configuroweb.com/simulador-de-dados-en-python\n  \n  '
```

0.2. Distribuciones Gaussianas (normales)

Ejercicio:

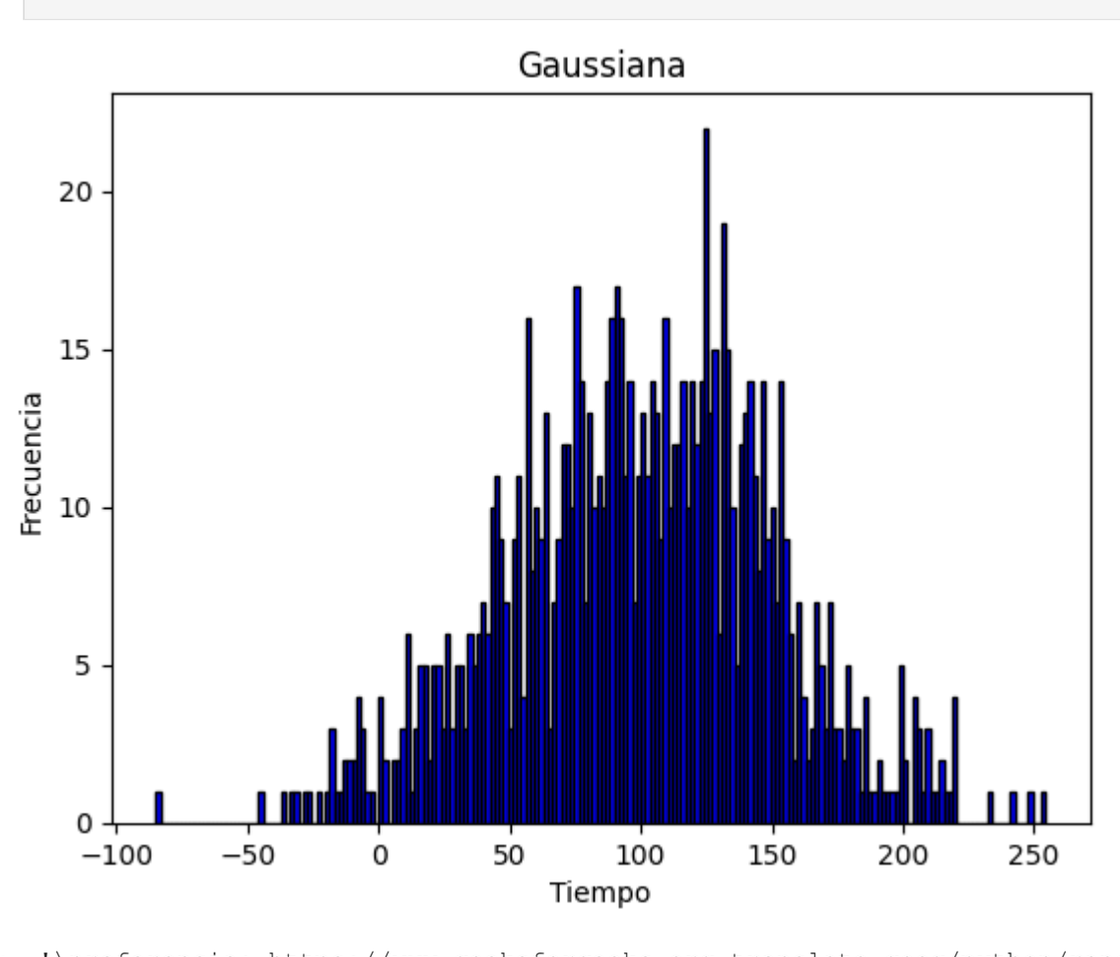
- Escriba código para generar una distribución Gaussiana. Explique los parámetros de dicha función de probabilidad.
- Repita esta actividad para la distribución de Poisson y Bernoilli

Una distribución gaussiana, también conocida como distribución normal, es una distribución de probabilidad continua caracterizada por su curva en forma de campana. Se define mediante dos parámetros:

μ (mu): La media o valor esperado de la distribución σ (sigma): La desviación típica, que mide la dispersión de la distribución

Referencias: https://www.datacamp.com/es/tutorial/gaussian-distribution

In [43]:
```python
# Inserte su código aquí
#----gaussiana----

n = 1000 #muestras
mu = 100 #media
sigma = 50 #desviacion


nums = [] # store the random numbers in a list
for i in range(n):
    temp = random.gauss(mu, sigma)
    nums.append(temp)

data = nums
bins = 200


# aqui graficamos
plt.hist(data, bins=bins, color='blue', edgecolor='black')
plt.title("Gaussiana")
plt.xlabel("Tiempo")
plt.ylabel("Frecuencia")
plt.show()


"""
referencia: https://www-geeksforgeeks-org.translate.goog/python/random-gauss-function-in-python/?_x_tr_sl=en&_x_tr_tl=es&_x_tr_hl=es&_x_tr_pto=tc
"""
```



Out[43]:
```
'\nreferencia: https://www-geeksforgeeks-org.translate.goog/python/random-gauss-function-in-python/?_x_tr_sl=en&_x_tr_tl=es&_x_tr_hl=es&_x_tr_pto=tc\n'
```

La distribución de Poisson es una distribución de probabilidad discreta que expresa la probabilidad de que se produzca un número determinado de sucesos en un intervalo fijo de tiempo o espacio. Supone que estos sucesos ocurren con una frecuencia media conocida e independientemente del tiempo transcurrido desde el último suceso.

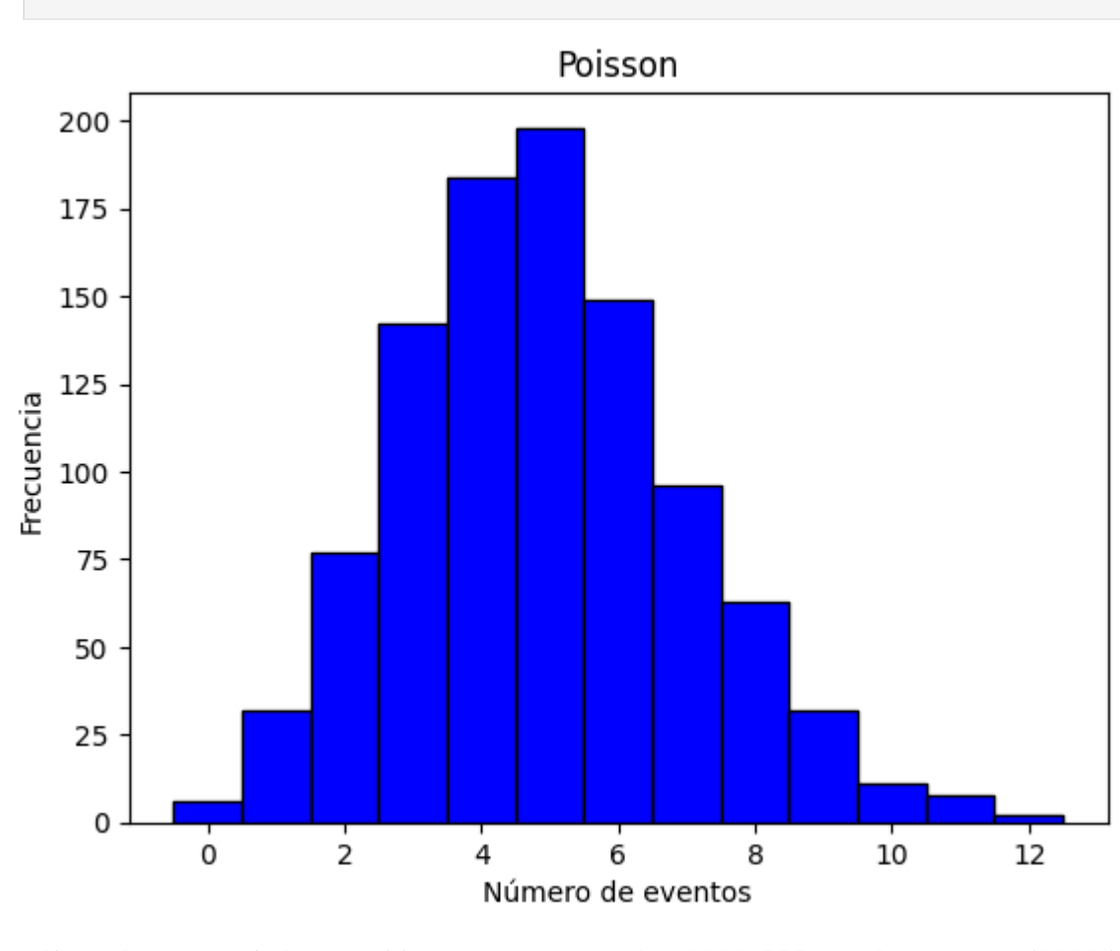
El modelo supone:

Los acontecimientos se producen de forma independiente. La tasa media de aparición (λ) permanece constante a lo largo del intervalo. Dos acontecimientos no pueden ocurrir exactamente en el mismo instante.

Su media (valor esperado) es igual a su varianza. Ambos están representados por el parámetro λ (lambda), que denota el número medio de sucesos en el intervalo.

Referencia: https://www.datacamp.com/es/tutorial/poisson-distribution

In [44]:
```python
# Inserte su código aquí

from scipy.stats import poisson

#----poisson----


n = 1000 # muestras
lam = 5  #  lambda (tasa promedio de ocurrencia)

nums = []
for i in range(n):
    temp = poisson.rvs(mu=lam)
    nums.append(temp)


data = nums

k_values = np.arange(0, max(data) + 1) #eventos
bins = np.arange(-0.5, k_values[-1] + 1.5, 1)


# aqui graficamos
plt.hist(data, bins=bins, color='blue', edgecolor='black')
plt.title("Poisson")
plt.xlabel("Número de eventos")
plt.ylabel("Frecuencia")
plt.show()

"""
Referencia:
https://programacionpython80889555.wordpress.com/2025/03/05/distribucion-de-poisson-teoria-e-implementacion-en-python
"""
```

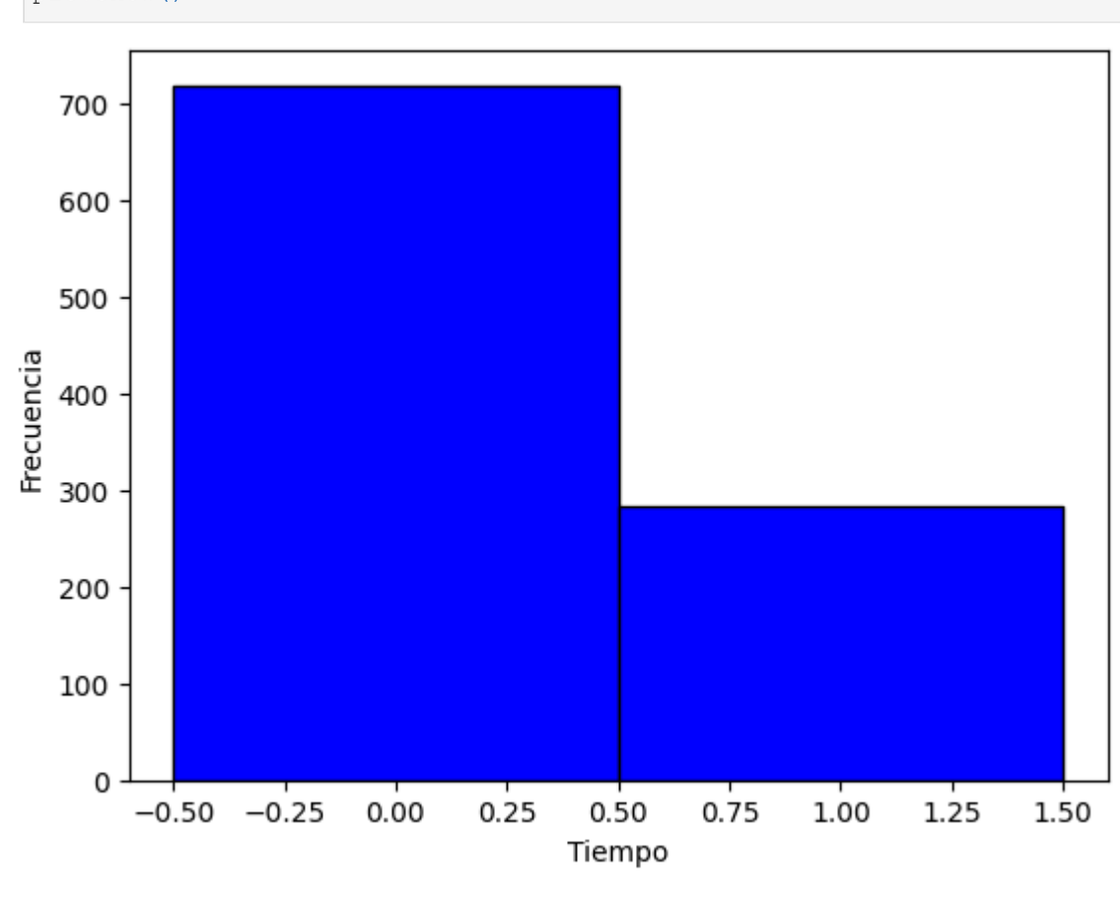


Out[44]:
```
'\nReferencia:\nhttps://programacionpython80889555.wordpress.com/2025/03/05/distribucion-de-poisson-teoria-e-implementacion-en-python\n'
```

Bernoilli A Bernoulli distribution is a discrete probability distribution that models a random variable with only two possible outcomes. These outcomes are typically labeled as "success" and "failure," or else they are represented numerically as 1 and 0.

referncia: https://www.datacamp.com/es/tutorial/bernoulli-distribution

In [45]:
```python
from scipy.stats import bernoulli

n = 1000 # muestras
p = 0.3

nums = []
for i in range(n):
    x = bernoulli.rvs(p=p) # 0 o 1
    nums.append(x)

data = nums
bins = [-0.5, 0.5, 1.5]


# aqui graficamos
plt.hist(data, bins=bins, color='blue', edgecolor='black')
plt.title("")
plt.xlabel("Tiempo")
plt.ylabel("Frecuencia")
plt.show()
```



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 [46]:
```python
# aqui simulamos los datos, por ejemplo gaussianos

n = 1000  #muestras
mu = 0.0 # media
sigma = 1.0 # desviacion estandar
data = np.random.normal(loc=mu, scale=sigma, size=n)

# 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 [47]:
```python
### Escriba su código aquí

x = datos_cargados['data'].to_numpy()

#--------con funciones------------#
media = np.mean(x)
desv = np.std(x)

print(media, desv)
```

```
0.007988819099535095 1.0440275601731104
```

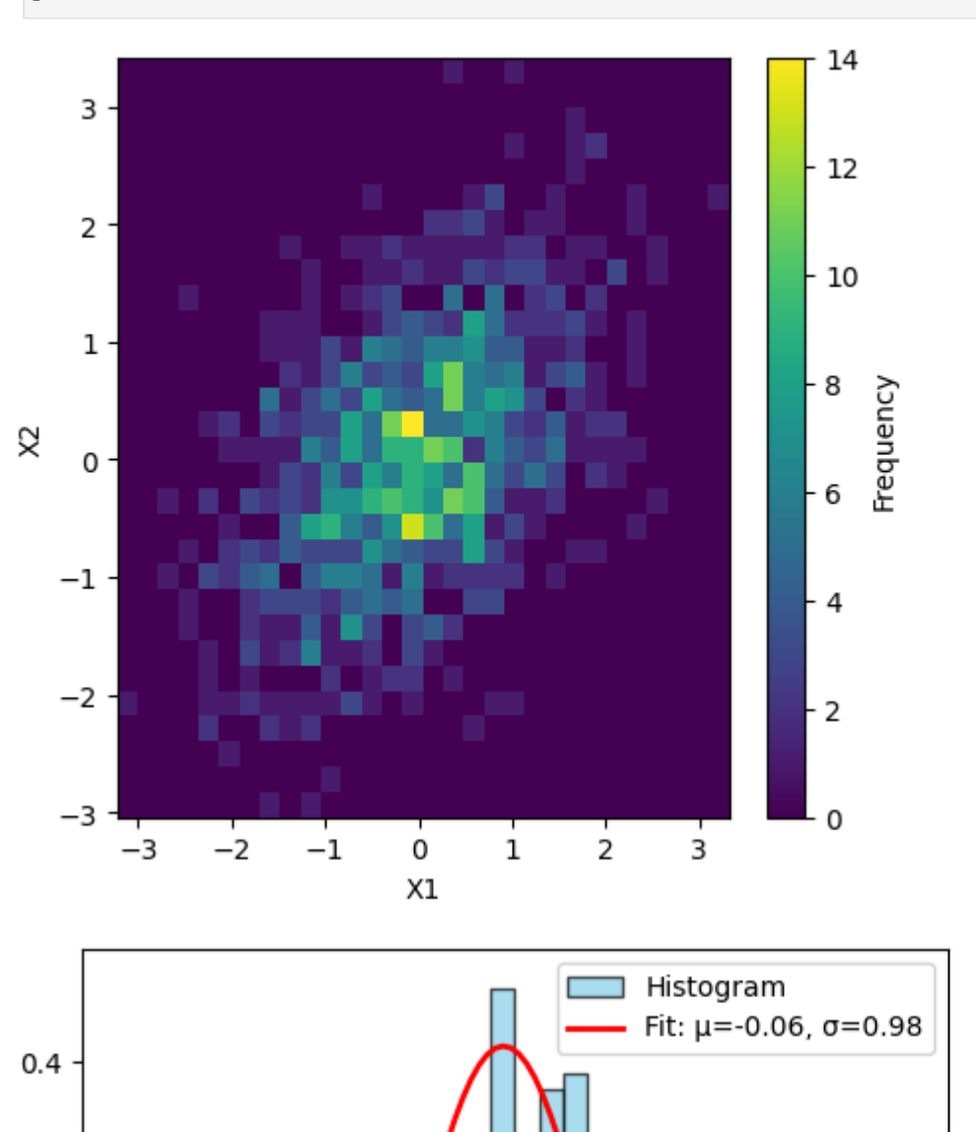
In [48]:
```python
#------a mano------
import math

x = datos_cargados['data'].tolist()
n = len(x)
m = sum(x) / n
var_mue = sum((xi - m) ** 2 for xi in x) / (n - 1)
desv = math.sqrt(var_mue)

print(m,desv)
```

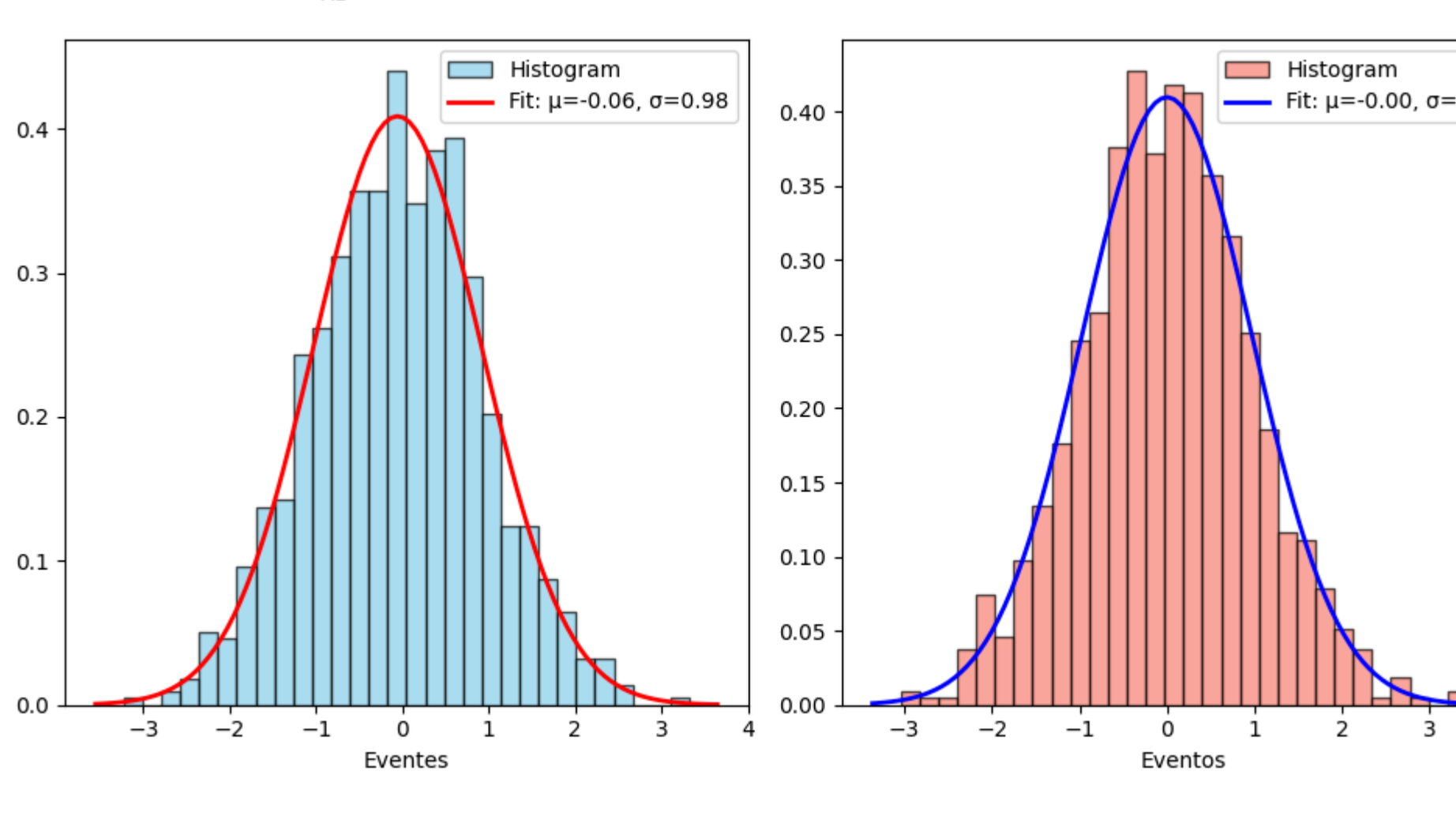
```
0.007988819099535109 1.044549965790076
```

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 [49]:
```python
# generar los datos
np.random.seed(42)
n = 1000
mean = [0.0, 0.0]
cov = [[1.0, 0.5],
       [0.5, 1.0]]

# [Sugerencia: use np.random.multivariate_normal()]
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='Frequency')
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
# inserte código aqui
df_cargado = pd.read_csv('gaussian_2d_data.csv')


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

# First variable histogram and fit
plt.subplot(1, 2, 1)
x1 = df_cargado['X1'].values
# hacer el fit [Sugerencia: usar norm.fit() ]
mu, std = norm.fit(x1)
plt.hist(x1, bins=30, density=True, color='skyblue', edgecolor='black', alpha=0.7,
         label='Histogram')

# 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:.2f}, σ={std:.2f}')
plt.xlabel('Eventes')
plt.legend()

# mismos comentarios que para x1
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=0.7,
         label='Histogram')

# 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:.2f}, σ={std:.2f}')
plt.xlabel('Eventos')
plt.legend()

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





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.

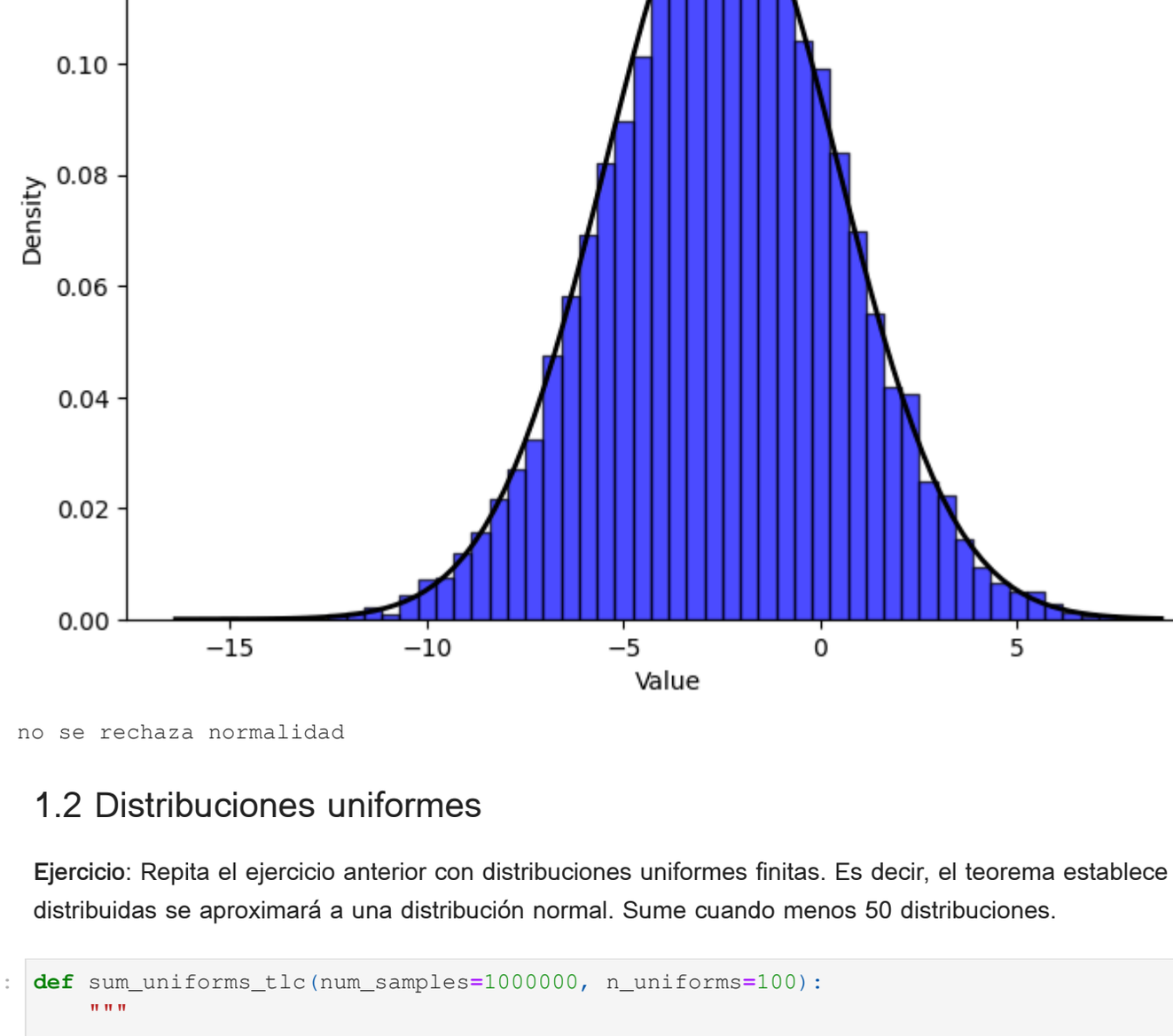
La prueba de Shapiro-Wilk es una prueba de hipótesis que evalúa si un conjunto de datos se distribuye normalmente. Evalúa los datos de una muestra con la hipótesis nula de que el conjunto de datos se distribuye normalmente. Un valor p alto indica que el conjunto de datos se distribuye normalmente; un valor p bajo, que no se distribuye normalmente.

referencia:https://builtin-com.translate.goog/data-science/shapiro-wilk-test?_x_tr_sl=en&_x_tr_tl=es&_x_tr_hl=es&_x_tr_pto=tc

se uso el mismo para el codigo

In [54]:
```python
def sum_gaussians_tlc(num_samples=10000, n_gaussians=3):
    """


    """
    # Generate independent Gaussian distributions
    means = np.random.uniform(-2.0, 2.0, size=n_gaussians)
    std_devs = np.random.uniform(0.5, 2.0, size=n_gaussians)
    coefficients = np.random.uniform(-2.0, 2.0, size=n_gaussians)

    # Initialize an empty list to store the samples
    gaussians = []

    # Loop through each Gaussian distribution
    for i in range(n_gaussians):
        # Generate samples for the current Gaussian distribution
        samples = np.random.normal(loc=means[i], scale=std_devs[i], size=num_samples)
        # Add the generated samples to the list
        gaussians.append(samples)

    # Linear combination of these Gaussians
    linear_combination = np.zeros(num_samples)
    for i in range(n_gaussians):
        linear_combination = linear_combination + coefficients[i] * gaussians[i]

    # Fit a Gaussian (normal) distribution to the linear combination
    mu, std = norm.fit(linear_combination)

    # Plot the histogram of the linear combination
    plt.figure(figsize=(8, 6))
    count, bins, _ = plt.hist(linear_combination, bins=50, density=True, alpha=0.7, color='blue', edgecolor='black', label='Linear Combination')

    # Plot the Gaussian fit
    xmin, xmax = plt.xlim()  # Get the limits of the x-axis for plotting the fit over the same range
    x = np.linspace(xmin, xmax, 100)
    p = norm.pdf(x, mu, std)  # Gaussian distribution fit

    # Scale the fit by the max height of the histogram to make it align
    plt.plot(x, p, 'k', linewidth=2, label=f'Gaussian mean={mu:.2f}, std={std:.2f}')

    # Perform the Shapiro-Wilk test for normality
    stat, p_value = shapiro(linear_combination)

    # Display the p-value on the plot
    plt.text(xmin + (xmax - xmin) * 0.05, max(count) * 0.9, f'p-value: {p_value:.5f}', fontsize=12, color='red')

    # Add labels, title, and legend
    plt.title(f'Linear Combination of {n_gaussians} Gaussian Distributions\nSample Size: {num_samples}')
    plt.xlabel('Value')
    plt.ylabel('Density')
    plt.legend()

    # Show the plot
    plt.show()

    # Print the p-value
    if p_value > 0.05:
        print(f"no se rechaza normalidad")
    else:
        print(f"se rechazo")
```

In [55]:
```python
# Ejecute su código
sum_gaussians_tlc(num_samples=10000, n_gaussians=3)
```



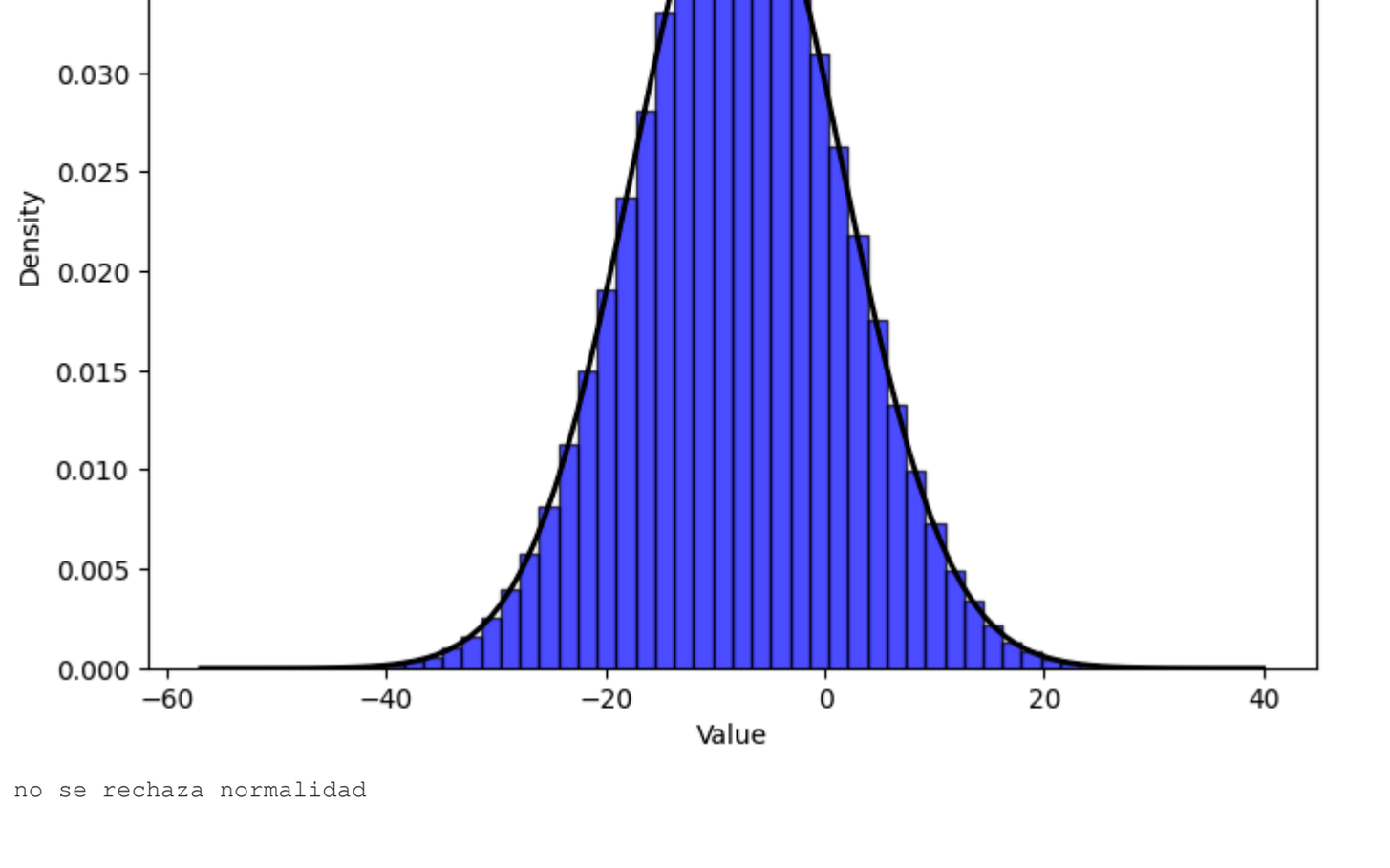

```
no se rechaza normalidad
```

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 [56]:
```python
def sum_uniforms_tlc(num_samples=1000000, n_uniforms=100):
    """


    """
    # Generate independent uniform distributions
    lower_bounds =np.random.uniform(-3.0, 0.0, size=n_uniforms)
    upper_bounds =np.random.uniform(0.0, 3.0, size=n_uniforms)
    coefficients =np.random.uniform(-2.0, 2.0, size=n_uniforms)

    uniforms = []

    # Loop through each uniform distribution
    for i in range(n_uniforms):
        # Generate random samples for the current uniform distribution
        current_samples = np.random.uniform(low=lower_bounds[i], high=upper_bounds[i], size=num_samples)
        # Append the generated samples to the list
        uniforms.append(current_samples)

    # Linear combination of these uniforms
    linear_combination = np.zeros(num_samples)
    for i in range(n_uniforms):
        linear_combination = linear_combination + coefficients[i] * uniforms[i]

    # Fit a Gaussian (normal) distribution to the linear combination
    mu, std = norm.fit(linear_combination)

    # Plot the histogram of the linear combination
    plt.figure(figsize=(8, 6))
    count, bins, _ = plt.hist(linear_combination, bins=50, density=True, alpha=0.7, color='blue', edgecolor='black', label='Linear Combination')

    # Plot the Gaussian fit
    xmin, xmax = plt.xlim()  # Get the limits of the x-axis for plotting the fit over the same range
    x = np.linspace(xmin, xmax, 100)
    p = norm.pdf(x, mu, std)

    # Scale the fit by the max height of the histogram to make it align
    plt.plot(x, p, 'k', linewidth=2, label=f'Gaussian mu={mu:.2f}, sigma={std:.2f}')

    # Perform the Shapiro-Wilk test for normality
    stat, p_value = shapiro(linear_combination)

    # Display the p-value on the plot
    plt.text(xmin + (xmax - xmin) * 0.05, max(count) * 0.9, f'p-value: {p_value:.5f}', fontsize=12, color='red')

    # Add labels, title, and legend
    plt.title(f'Linear Combination of {n_uniforms} Uniform Distributions\nSample Size: {num_samples}')
    plt.xlabel('Value')
    plt.ylabel('Density')
    plt.legend()

    # Show the plot
    plt.show()

    # Print the p-value
    if p_value > 0.05:
        print(f"no se rechaza normalidad")
    else:
        print(f"se rechaza normalidad")
```

In [58]:
```python
sum_uniforms_tlc(num_samples=1000000, n_uniforms=100)
```




```
no se rechaza normalidad
```