Ejercicios

Parte 1

Te doy la bienvenida al último paso de este curso de probabilidad para ciencia de datos. Es momento de que pongas a prueba todo lo que has aprendido.

Completa los ejercicios que se proponen en cada una de las notebooks de Colab de diferentes temas cada una. Recuerda compartir tus resultados o dudas en los comentarios para que entre todos podamos apoyarnos.

Tipos de probabilidad Los diferentes tipos de probabilidades que existen con frecuencia generan algo de confusión, estos ejercicios te permitirán fortalecer el entendimiento de estos conceptos:

https://colab.research.google.com/drive/1yQFCKo2GHtbJui0szPjLhAJSjpivBeun?usp=sharing#scrollTo=YTVHbmebucgL

Ejercicios (bloque 1)

Considerando un lanzamiento de un dado y considerando los siguientes eventos aleatorios:

A = {el resultado del lanzamiento de un dado es 6}

B = {el resultado del lanzamiento de un dado es par}

C = {el resultado del lanzamiento de un dado es impar}

Calcula las siguientes probabilidades:

1. $P(A|B) = \frac{1}{3}$
2. $P(A|C) = \frac{0}{3}$
3. $P(B|C) = \frac{0}{3}$

Ejercicios (bloque 2)

Considerando una ruleta de doce números

$\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$

dos jugadores eligen 6 números, cada uno de ellos. Supón que el jugador 1 elige A = $\{1, 2, 3, 4, 5, 6\}$ y calcula las siguientes probabilidades:

1. $P(A|B)$ sabiendo que el jugador 2 elige B = $\{2, 4, 6, 8, 10, 12\}$

   A = {1,2,3,4,5,6} y B = {2,4,6,8,10,12} y $A \cap B$ = {2,4,6}

   $P(A|B) = \frac{3}{6}$

2. $P(A|B)$ sabiendo que el jugador 2 elige B = $\{1, 3, 5, 7, 9, 11\}$

   A = {1,2,3,4,5,6} y B = {1,3,5,7,9,11} y $A \cap B$ = {1,3,5}

   $P(A|B) = \frac{3}{6}$

3. $P(A|B)$ sabiendo que el jugador 2 elige B = $\{5, 6, 7, 8, 9, 10\}$

   A = {1,2,3,4,5,6} y B = {5,6,7,8,9,10} y $A \cap B$ = {5,6}

   $P(A|B) = \frac{2}{6}$

Ejercicios (bloque 3)

Considera un problema donde se lanzan dos monedas, sean m_1 y m_2. Verifica la regla del producto para las siguientes probabilidades (dibuja el espacio muestral y calcula cada probabilidad por separado):

1. $P(m_1 = \text{cara}, m_2 = \text{sello})$
2. $P(m_1 = \text{cara}|m_2 = \text{sello})$
3. $P(m_2 = \text{sello})$

Tenemos el espacio muestral siguiente:

	Moneda 1		
Moneda 2		cara 1	cruz 1
	cara 2	cara 1 cara 2	cruz 1 cara 2
	cruz 2	cara 1 cruz 2	cruz 1 cruz 2

1. $P(m_1 = \text{cara}, m_2 = \text{cruz}) = \frac{1}{4}$
2. $P(m_1 = \text{cara} | m_2 = \text{cruz}) = \frac{1}{2}$
3. $P(m_2 = \text{cruz}) = \frac{2}{4}$

Parte 2

Distribuciones de probabilidad

Luego de que entendemos que hay diferentes tipos de probabilidades, el concepto de distribución de probabilidad nos dice que podemos usar funciones matemáticas para mapear cada ocurrencia posible de una variable aleatoria a un número que es la probabilidad de esa ocurrencia.

En este conjunto de retos exploramos más a fondo el cálculo con distribuciones discretas, especialmente la binomial:

https://colab.research.google.com/drive/10xbi88L6alsPC8-cec_6VuUeh7ttFaEF?usp=sharing

Distribuciones discretas (e.j. binomial)

Recordando que la distribución binomial está dada por:

$$P(k, n; p) = \binom{n}{k} p^k (1-p)^{n-k} = \frac{n!}{k!(n-k)!} p^k (1-p)^{n-k}$$

donde $P(k, n; p)$ representa la probabilidad de obtener k éxitos de n intentos con posibilidad **binaria** (por ejemplo, lanzamientos de moneda).

`Ejemplo` : la probabilidad de obtener 4 caras a partir de 10 lanzamientos consecutivos de moneda, está dada por (tomando $p = 0.5$, por lo tanto $1 - p = 0.5$):

$$P(k = 4, n = 10; p = 0.5) = \binom{10}{4} \left(\frac{1}{2}\right)^{10} = \frac{10!}{4!6!} \left(\frac{1}{2}\right)^{10}$$

Ahora, la probabilidad de obtener k o menos éxitos a partir de n intentos está dada por la distribución acumulada:

$$C(k, n; p) = \sum_{i=0}^{k} P(i, n; p) = \sum_{i=0}^{k} \binom{n}{i} p^i (1-p)^{n-i}$$

Por convención entendemos que:

$$C(k = 3, n = 6; p = 0.5) = P(k \leq 3, n = 6, p = 0.5)$$

`Ejemplo` : la probabilidad de obtener 3 o menos caras a partir de 6 lanzamientos consecutivos está dada por (tomando $p = 0.5$, por lo tanto $1 - p = 0.5$):

$$P(k \leq 3, n = 6; p = 0.5) = \sum_{i=0}^{3} \binom{6}{i} \left(\frac{1}{2}\right)^6$$

$$P(k \leq 3, n = 6; p = 0.5) = \left(\frac{1}{2}\right)^6 \sum_{i=0}^{3} \binom{6}{i}$$

$$P(k \leq 3, n = 6; p = 0.5) = \left(\frac{1}{2}\right)^6 \left\{ \binom{6}{0} + \binom{6}{1} + \binom{6}{2} + \binom{6}{3} \right\}$$

> Nota importante: se puede factorizar el termino debido a que la probabilidad es justa o igual y no existe diferencia entre la probabilidad de fracaso, para el caso que sean probabilidades diferentes de 0.5 se tiene que recurrir al uso de sustitución puntual por cada valor que va adquiriendo K.

Ejercicios (bloque 1)

Calcula a mano las siguientes probabilidades (tomando $p = 0.5$, por lo tanto $1 - p = 0.5$):

1. Probabilidad de obtener 3 caras a partir de 12 lanzamientos de moneda.
2. Probabilidad de obtener 5 o menos caras a partir de 10 lanzamientos de moneda.
3. Probabilidad de obtener menos de 6 caras a partir de 10 lanzamientos de moneda.

Calcula a mano las mismas probabilidades anteriores pero considerando ahora $p = 0.3$.

Para la resolución de algunos ejercicios nos apoyaremos de

$$C(k, n; p) = \sum_{i=0}^{k} P(i, n; p) = \sum_{i=0}^{k} \binom{n}{i} p^i (1-p)^{n-i}$$

RESPUESTAS

1.- Tomando $p = 0.5$

1. $\frac{55}{1024} = 0.0557$
2. $\frac{319}{512} = 0.6230$
   - $\binom{10}{0} = \frac{1}{1024}$
   - $\binom{10}{1} = \frac{5}{512}$
   - $\binom{10}{2} = \frac{45}{1024}$
   - $\binom{10}{3} = \frac{15}{128}$
   - $\binom{10}{4} = \frac{105}{512}$
   - $\binom{10}{5} = \frac{63}{256}$
3. $\frac{53}{64} = 0.8281$
   - $\binom{10}{0} = \frac{1}{1024}$
   - $\binom{10}{1} = \frac{5}{512}$
   - $\binom{10}{2} = \frac{45}{1024}$
   - $\binom{10}{3} = \frac{15}{128}$
   - $\binom{10}{4} = \frac{105}{512}$
   - $\binom{10}{5} = \frac{63}{256}$
   - $\binom{10}{6} = \frac{105}{512}$

2.- Tomando $p = 0.3$

1. 0.2397
2. 0.95254
   - $\binom{10}{0} = 0.02824$
   - $\binom{10}{1} = 0.1210$
   - $\binom{10}{2} = 0.2335$
   - $\binom{10}{3} = 0.2668$
   - $\binom{10}{4} = 0.2001$
   - $\binom{10}{5} = 0.1029$
3. 0.9893
   - $\binom{10}{0} = 0.02824$
   - $\binom{10}{1} = 0.1210$
   - $\binom{10}{2} = 0.2335$
   - $\binom{10}{3} = 0.2668$

- $\binom{10}{4} = 0.2001$
- $\binom{10}{5} = 0.1029$
- $\binom{10}{6} = 0.0368$

Usando la función `mi_binom()`, definida previamente, verifica el cálculo de todas las probabilidades del punto anterior.

In [ ]:
```python
#Probabilidad de distribución binomial
from math import factorial

def my_binomial(k, n, p):
  return factorial(n)/(factorial(k)*(factorial(n-k)))*pow(p,k)*pow(1-p, n-k)
```

In [ ]:
```python
print(f'Probabilidad de obtener 3 caras a partir de 12 lanzamientos con p=0.5 \nP = {my_binomial(3,12,0.5)}\n')
```

```
Probabilidad de obtener 3 caras a partir de 12 lanzamientos con p=0.5
P = 0.0537109375
```

In [ ]:
```python
#Comprobando ejercicio 1
print(f'Probabilidad de obtener 3 caras a partir de 12 lanzamientos con p=0.5 \nP = {my_binomial(3,12,0.5)}\n')
print(f'Probabilidad de obtener 3 caras a partir de 12 lanzamientos con p=0.3 \nP = {my_binomial(3,12,0.3)}')
```

```
Probabilidad de obtener 3 caras a partir de 12 lanzamientos con p=0.5
P = 0.0537109375

Probabilidad de obtener 3 caras a partir de 12 lanzamientos con p=0.3
P = 0.23970042557999985
```

Como se puede observar el calculo fue correcto. Lo haré con el siguiente código, para verificar si se puede expresar en forma racional.
Pero solo tomaré en cuenta cuando K=3.

In [ ]:
```python
#Obteniendo el código para calcular la distribución acumulada
from fractions import Fraction
from math import factorial

def mi_bino(k,n,p):
    proba_kn=[]
    for ka in range(k+1):
        dis_ka=(factorial(n)/(factorial(ka)*factorial(n-ka)))*(p**ka)*((1-p)**(n-ka))
        print(f'Probabilidad para obtener {ka} caras = {Fraction(dis_ka)} = {dis_ka} ')
        proba_kn.append(dis_ka)

    proba_acu=sum(proba_kn)
    print(f'\nProbabilidad para obtener {k} o menos caras  = {Fraction(proba_acu)} = {proba_acu}\n')
    return proba_acu,proba_kn
```

In [ ]:
```python
# Valores específicos:
k_val = 2
n_val = 3
p_val = 0.5

proba_acu, proba_kn = mi_bino(k_val, n_val, p_val)
print(f'Probabilidad por Kn: {proba_kn}')
print(f'Probabilidad total: {proba_acu}')
```

```
Probabilidad para obtener 0 caras = 1/8 = 0.125
Probabilidad para obtener 1 caras = 3/8 = 0.375
Probabilidad para obtener 2 caras = 3/8 = 0.375
Probabilidad para obtener 2 o menos caras  = 7/8 = 0.875

Probabilidad por Kn: [0.125, 0.375, 0.375]
Probabilidad total: 0.875
```

In [ ]:
```python
# Resolviendo ejercicio 1:
k_val = 3
n_val = 12

#Resolviendo para p=0.5
p_val = 0.5
proba_acu, proba_kn = mi_bino(k_val, n_val, p_val)
print(f"La probabilidad de que se obtenga un valor de k = {k_val} en {n_val} intentos con p = {p_val}\nP = {proba_kn[3]} = {Fracti

#Resolviendo para p=0.3
p_val = 0.3
proba_acu, proba_kn = mi_bino(k_val, n_val, p_val)
print(f"La probabilidad de que se obtenga un valor de k = {k_val} en {n_val} intentos con p = {p_val}\nP = {proba_kn[3]} = {Fracti
```

```
Probabilidad para obtener 0 caras = 1/4096 = 0.000244140625
Probabilidad para obtener 1 caras = 3/1024 = 0.0029296875
Probabilidad para obtener 2 caras = 33/2048 = 0.01611328125
Probabilidad para obtener 3 caras = 55/1024 = 0.0537109375

Probabilidad para obtener 3 o menos caras  = 299/4096 = 0.072998046875

La probabilidad de que se obtenga un valor de k = 3 en 12 intentos con p = 0.5
P = 0.0537109375 = 55/1024

Probabilidad para obtener 0 caras = 7978958832736201/576460752303423488 = 0.01384128720099999
Probabilidad para obtener 1 caras = 2564665339093779/36028797018963968 = 0.07118376274799995
Probabilidad para obtener 2 caras = 6045282585006765/36028797018963968 = 0.1677902979059999
Probabilidad para obtener 3 caras = 8636117978581093/36028797018963968 = 0.23970042557999985

Probabilidad para obtener 3 o menos caras  = 8872375414863825/18014398509481984 = 0.4925157734349997

La probabilidad de que se obtenga un valor de k = 3 en 12 intentos con p = 0.3
P = 0.23970042557999985 = 8636117978581093/36028797018963968
```

In [ ]:
```python
#Resolviendo ejercicio 2
k_val = 5
n_val = 10

#Resolviendo para p=0.5
p_val = 0.5
proba_acu, proba_kn = mi_bino(k_val, n_val, p_val)

#Resolviendo para p=0.3
p_val = 0.3
proba_acu, proba_kn = mi_bino(k_val, n_val, p_val)
```

```
Probabilidad para obtener 0 caras = 1/1024 = 0.0009765625
Probabilidad para obtener 1 caras = 5/512 = 0.009765625
Probabilidad para obtener 2 caras = 45/1024 = 0.0439453125
Probabilidad para obtener 3 caras = 15/128 = 0.1171875
Probabilidad para obtener 4 caras = 105/512 = 0.205078125
Probabilidad para obtener 5 caras = 63/256 = 0.24609375

Probabilidad para obtener 5 o menos caras  = 319/512 = 0.623046875

Probabilidad para obtener 0 caras = 8141794727281839/288230376151711744 = 0.028247524899999984
Probabilidad para obtener 1 caras = 272604734172383/2251799813685248 = 0.12106082099999993
Probabilidad para obtener 2 caras = 2102950806472669/9007199254740992 = 0.23347444049999988
Probabilidad para obtener 3 caras = 4806744700508957/18014398509481984 = 0.2668279319999998
Probabilidad para obtener 4 caras = 7210117050763437/36028797018963968 = 0.2001209489999999
Probabilidad para obtener 5 caras = 7416120395070963/72057594037927936 = 0.10291934519999994

Probabilidad para obtener 5 o menos caras  = 8580717490718967/9007199254740992 = 0.9526510125999995
```

In [ ]:
```python
#Resolviendo ejercicio 3
k_val = 6
n_val = 10

#Resolviendo para p=0.5
p_val = 0.5
proba_acu, proba_kn = mi_bino(k_val, n_val, p_val)

#Resolviendo para p=0.3
p_val = 0.3
proba_acu, proba_kn = mi_bino(k_val, n_val, p_val)
```

```
Probabilidad para obtener 0 caras = 1/1024 = 0.0009765625
Probabilidad para obtener 1 caras = 5/512 = 0.009765625
Probabilidad para obtener 2 caras = 45/1024 = 0.0439453125
Probabilidad para obtener 3 caras = 15/128 = 0.1171875
Probabilidad para obtener 4 caras = 105/512 = 0.205078125
Probabilidad para obtener 5 caras = 63/256 = 0.24609375
Probabilidad para obtener 6 caras = 105/512 = 0.205078125

Probabilidad para obtener 6 o menos caras  = 53/64 = 0.828125

Probabilidad para obtener 0 caras = 8141794727281839/288230376151711744 = 0.028247524899999984
Probabilidad para obtener 1 caras = 272604734172383/2251799813685248 = 0.12106082099999993
Probabilidad para obtener 2 caras = 2102950806472669/9007199254740992 = 0.23347444049999988
Probabilidad para obtener 3 caras = 4806744700508957/18014398509481984 = 0.2668279319999998
Probabilidad para obtener 4 caras = 7210117050763437/36028797018963968 = 0.2001209489999999
Probabilidad para obtener 5 caras = 7416120395070963/72057594037927936 = 0.10291934519999994
Probabilidad para obtener 6 caras = 2648614426811059/72057594037927936 = 0.03675690899999999

Probabilidad para obtener 6 o menos caras  = 8911794294070349/9007199254740992 = 0.9894079215999995
```

Parte 3

Mientras que en este otro conjunto de retos exploramos más a detalle la distribución gaussiana como representante de las distribuciones continuas:

https://colab.research.google.com/drive/1OVQ0fwQ1q1lPFjmjPHyuCkD58SZfm4o9?usp=sharing

Distribuciones continuas (e.j. gaussiana)

Recordemos que la distribución de probabilidad normal o gaussiana está dada por:

$$P(X) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left[-\frac{1}{2}\left(\frac{X-\mu}{\sigma}\right)^2\right]$$

donde:

- μ: media de la distribución
- σ: desviación estandar de la distribución

`Ejemplo` : considerando una variable aleatoria que sigue una distribución normal con media $\mu = 4$ y desviación estándar $\sigma = 0.3$, la probabilidad de que dicha variable tome el valor de 0.2 está dada por:

$$P(0.2) = \frac{1}{0.3\sqrt{2\pi}} \exp\left[-\frac{1}{2}\left(\frac{0.2-4}{0.3}\right)^2\right]$$

Lo cual en Python se traduce en:

```
from scipy.stats import norm

norm(mu, sigma).pdf(X)
```

Así también, la distribucion de probabilidad acumulada correspondiente está dada por:

$$C(X) = \int_{x<=X} P(x)dx = \int_{-\infty}^{x} P(X)dX$$

teniendo en cuenta que $Dom(X) = (-\infty, \infty)$.

`Ejemplo` : considerando una variable aleatoria que sigue una distribucion normal con media $\mu = 4$ y desviación estándar $\sigma = 0.3$, la probabilidad de que dicha variable tome el valor de 0.2 o menos está dada por:

$$C(0.2) = \int_{x<=0.2} P(x)dx = \int_{-\infty}^{0.2} \left\{ \frac{1}{0.3\sqrt{2\pi}} \exp\left[-\frac{1}{2}\left(\frac{X-4}{0.3}\right)^2\right] \right\} dX$$

La cual se calcula en Python como:

```
from scipy.stats import norm

norm(mu, sigma).cdf(X)
```

Es importante recordar que la función de probabilidad acumulada de la distribución gaussiana no se puede calcular de forma exacta, ya que la integral anterior no tiene una expresión cerrada conocida. Es decir, los métodos de integración conocidos no funcionan para resolver esta integral.

Ejercicios (bloque 1)

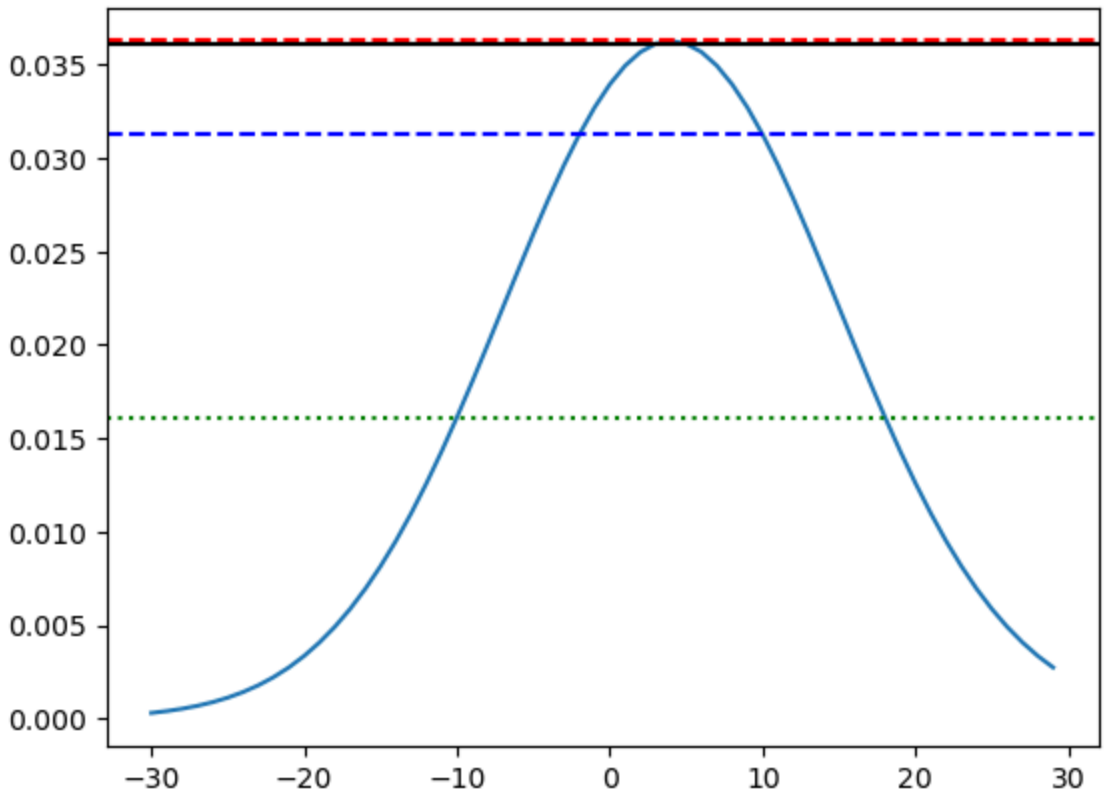
Considerando una variable aleatoria que sigue una distribución normal con media $\mu = 4$ y desviación estándar $\sigma = 0.1$, calcula las siguientes probabilidades (usando Python):

1. $P(X = 4)$
2. $P(X = -10)$
3. $P(X = 10)$
4. $P(X <= 4)$
5. $P(X >= 4)$

In [ ]:
```
from scipy.stats import norm
from matplotlib import pyplot
promedio = 4
desviacion = 11
print(f'Calculo de Densidad de probabilidad con Promedio = {promedio} y desviación estándar = {desviacion} \n')

# 1 X=4
X=4
print(f'F(X = {X}) = {norm(promedio, desviacion).pdf(X)}\n')

# 1 X=-10
X=-10
print(f'F(X = {X}) = {norm(promedio, desviacion).pdf(X)}\n')

# 1 X=10
X=10
print(f'F(X = {X}) = {norm(promedio, desviacion).pdf(X)}\n')

# 1 X<=4
X=4
print(f'F(X <= {X}) = {norm(promedio, desviacion).cdf(X)}\n')
```

```
# 1 X>=
X=4
print(f'F(X >= {X}) = {1-norm(promedio, desviacion).cdf(X)}\n')

#Creando un objeto basado en la distribución normal
distribucion = norm(promedio,desviacion)

#Obtener valores numéricos sobre los cuales evaluar la función
#teórica de la Densidad de Probabilidad
X = [value for value in range(-30,30)]
#Calculando las probabilidades de cada uno de esos datos
#de manera recursiva y usando el objeto distribucion
probabilidades = [distribucion.pdf(x) for x in X]

#Graficando la función de densidad de probabilidad
pyplot.plot(X,probabilidades)
pyplot.axhline(y=distribucion.pdf(4), color='r',linestyle='--')
pyplot.axhline(y=distribucion.pdf(-10),color='g',linestyle=':')
pyplot.axhline(y=distribucion.pdf(10),color='b',linestyle='--')
pyplot.axhline(y=distribucion.pdf(3),color='k')
pyplot.show()
```

```
Calculo de Densidad de probabilidad con Promedio = 4 y desviación estándar = 11

F(X = 4) = 0.03626748003649388

F(X = -10) = 0.0161352144695116

F(X = 10) = 0.03125443050835863

F(X <= 4) = 0.5

F(X >= 4) = 0.5
```



De la binomial a la gaussiana

En la clase 8 vimos como generar secuencias aleatorias de experimentos binomiales (aquí), donde cada experimento era lanzar un cierto número de monedas.

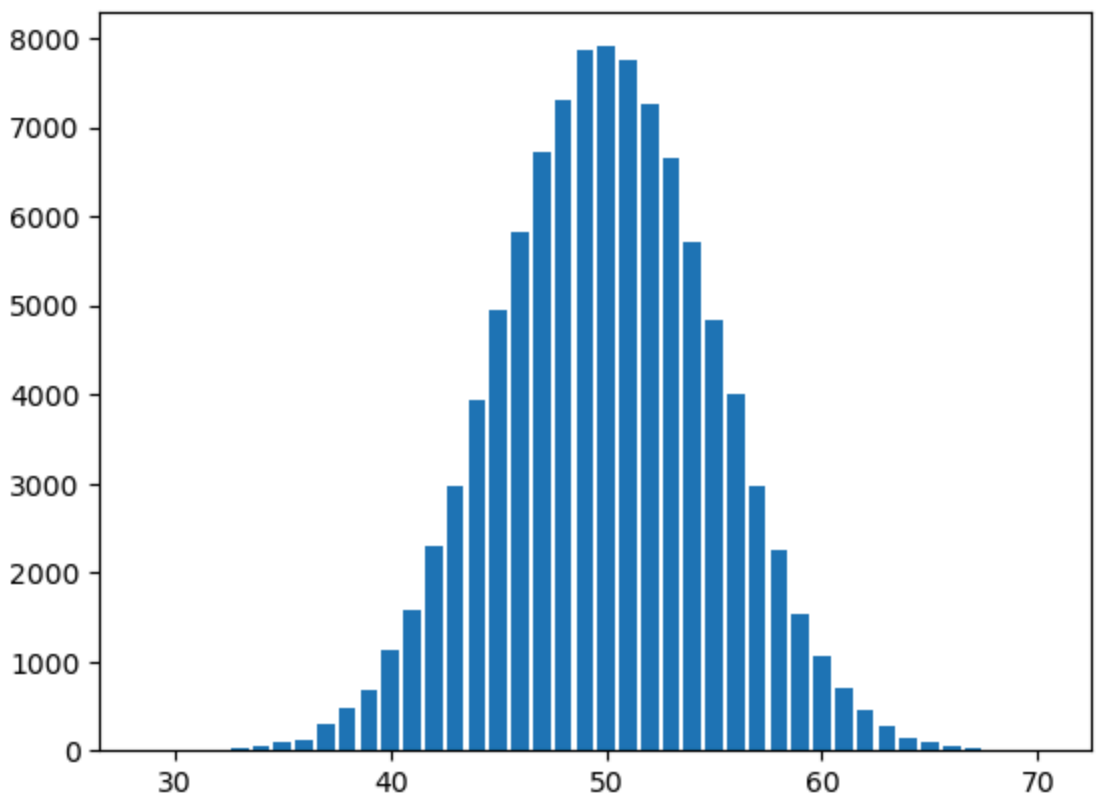
¿Qué sucede si el número consecutivo de monedas que lanzamos en cada experimento (trial) es muy largo?

La función `generate_binomial_trials()` nos muestra lo que sucede si graficamos los resultados de muchos experimentos de lanzar 100 monedas en cada intento, con pyplot:

In [ ]:
```
import numpy as np
from numpy.random import binomial
import matplotlib.pyplot as plt

def generate_binomial_trials(trials=1000, coin_toss=100):
  '''
  el resultado de esta funcion es generar un conjuntos
  de experimentos binomiales (trials) y de cada uno obtener
  las cantidades de exitos en cada secuencia (e.j. lanzar monedas).

  * trial: es una secuencia de <coin_toss> lanzamientos de moneda

  * coin_toss: es el numero de monedas lanzadas en cada trial
  '''
  arr = []
  for _ in range(trials):
    arr.append(binomial(coin_toss, 0.5))
  values, dist = np.unique(arr, return_counts=True)

  return values, dist, arr
```

```
values, dist,arr = generate_binomial_trials(100000)
plt.bar(values, dist)
```

Out[ ]: <BarContainer object of 42 artists>



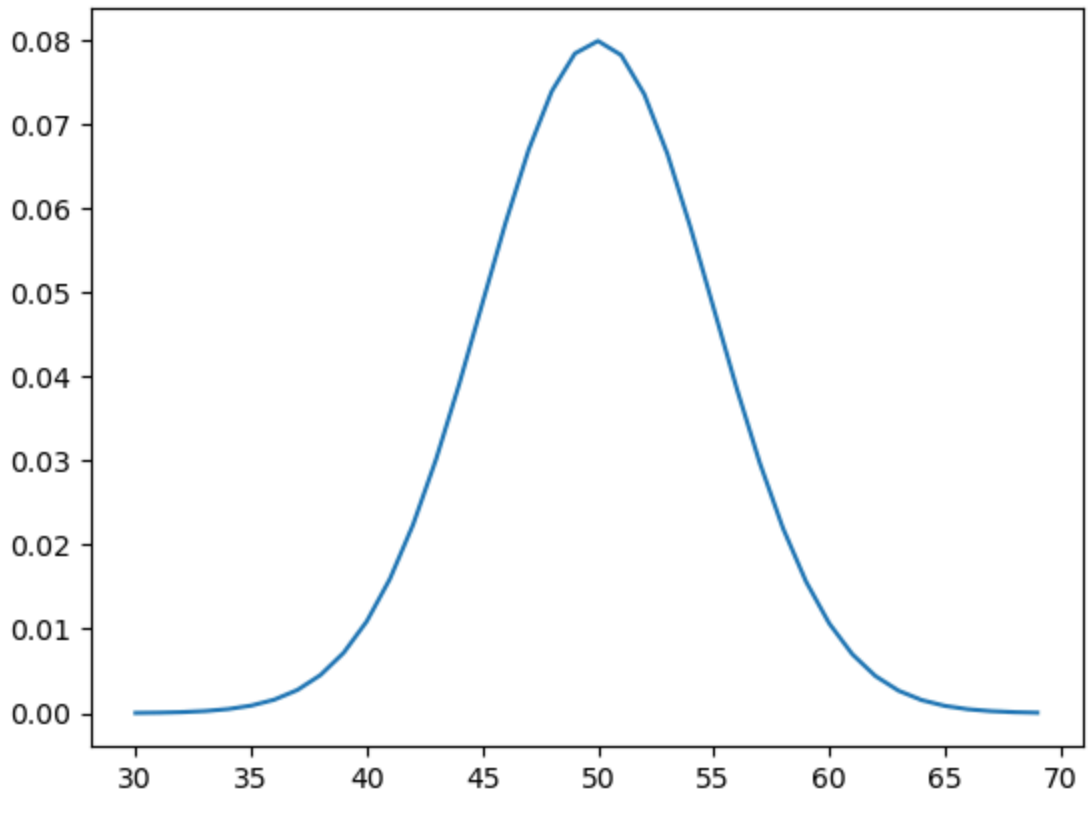
¿Se parece a algo conocido? Tal vez una ditribución normal se ajuste, para verificarlo haremos el siguiente ejercicio:

Ejercicios (bloque 2)

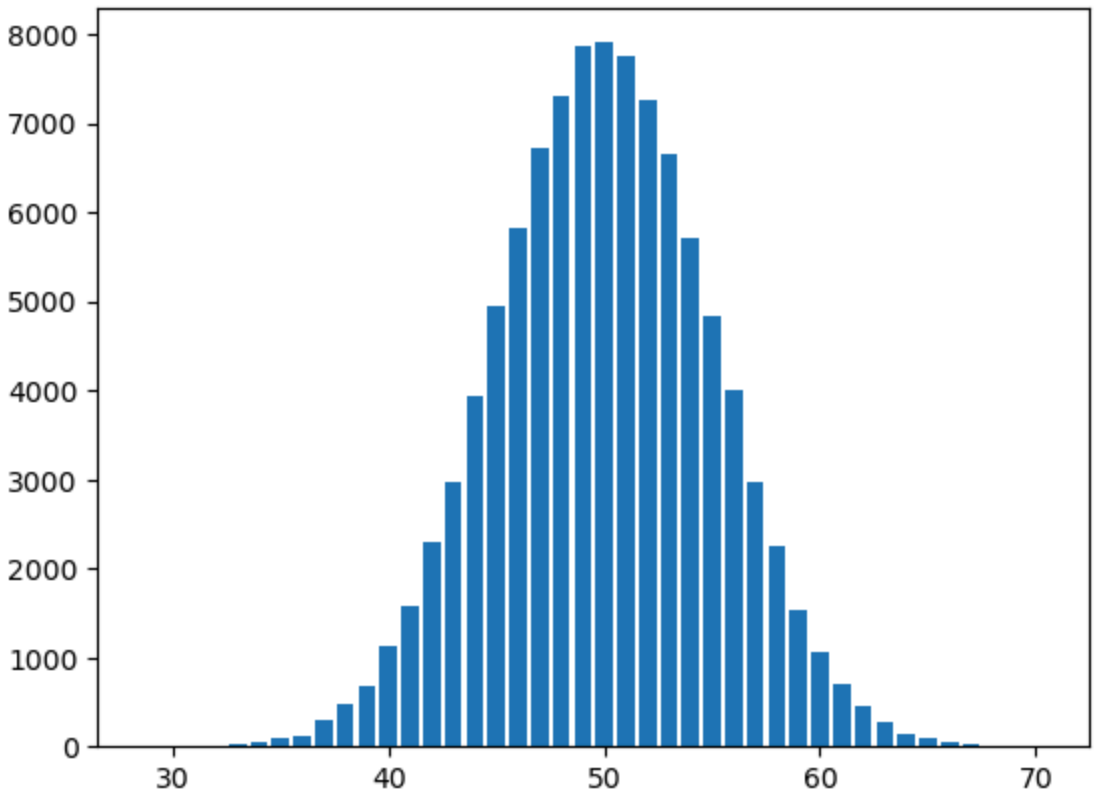
1. Con los resultados anteriores guardados en `values, dist` ajusta, usando el método de estimación paramétrica, una distribución gaussiana donde la media y desviación estándar correspondan a lo calculado a partir de los datos, de la misma manera que se hizo en el notebook de la clase 9 (aquí).

In [ ]:
```
#OBTENIENDO DATOS IMPORTANTES PARA LA DISTRIBUCIÓN TEÓRICA
#Obtener datos de l ejercicio anterior
sample = np.array(arr)
#Calcular el promedio de los datos
promedio = sample.mean()
#Calcular la desviación estándar de los datos
desviacion = sample.std()
#Creando un objeto basado en la distribución normal
distribucion = norm(promedio,desviacion)
#Obtener valores numéricos sobre los cuales evaluar la función
#teórica de la Densidad de Probabilidad
x = [value for value in range(30,70)]
#Calculando las probabilidades de cada uno de esos datos
#de manera recursiva y usando el objeto distribucion
probabilidades = [distribucion.pdf(value) for value in x]
#Graficando función teórica
pyplot.plot(x,probabilidades)
```

Out[ ]: [<matplotlib.lines.Line2D at 0x7fe90e64aab0>]



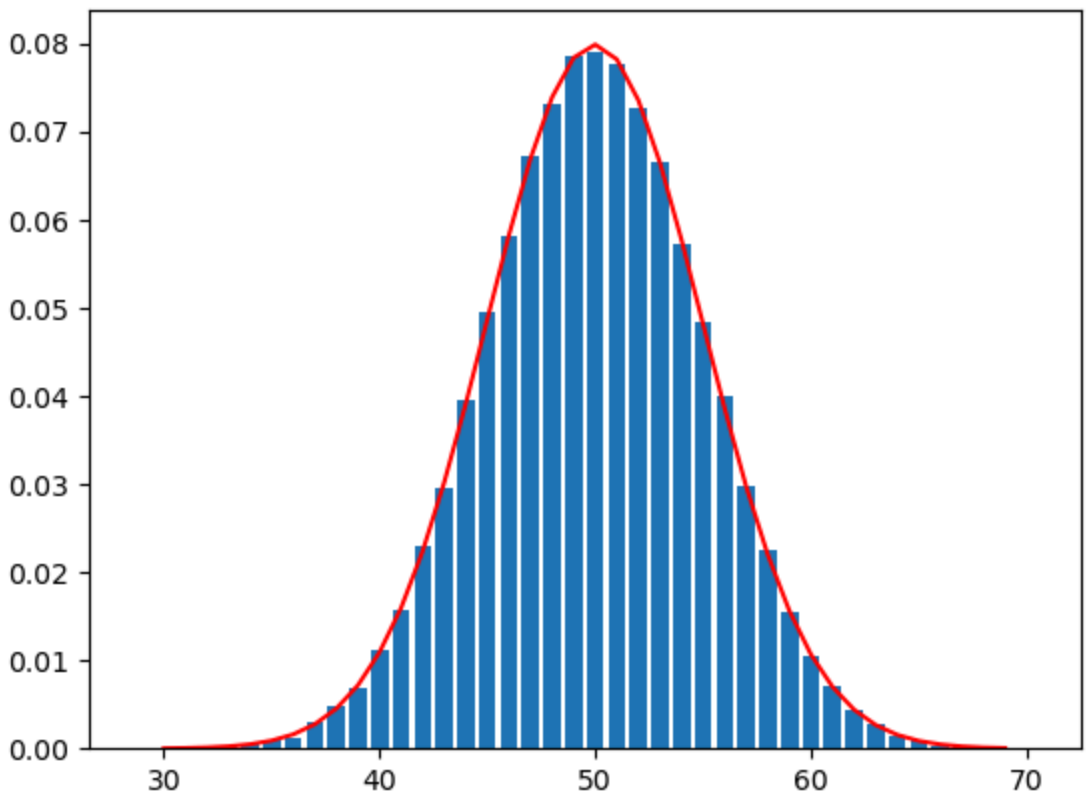
In [ ]:
```
#Graficando los datos del ejercicios de las monedas
plt.bar(values, dist)
#Graficando la función de densidad de probabilidad
pyplot.show()
```



Gratificándolos juntos

In [ ]:
```
#Graficando función teórica
pyplot.plot(x,probabilidades,color='r')
#Graficando los datos del ejercicios de las monedas
#pyplot.hist(arr,bins=30,density=True)
plt.bar(values, dist/len(arr))
```

Out[ ]: <BarContainer object of 42 artists>



Si todo salió bien, habrás notado que en efecto una distribución normal se ajusta perfectamente a los datos. Esto se conoce como el **teorema del límite central**, el cual establece que en muchas situaciones conocidas, cuando variables aleatorias independientes se combinan, su total tiende a seguir una distribución normal cuando el número de variables que se combinan es muy grande $n \to \infty$.

Parte 4

Estimación de densidades de probabilidad Uno de los métodos más importantes para estimar densidades de probabilidad es el MLE (Maximum Likelihood Estimation), del cual podrás profundizar en este reto:

https://colab.research.google.com/drive/19F8F0ID9ErtiHUNbWsOIJ7f22u7jjS0d?usp=sharing

Estimación de máxima verosimilitud

Como ya vimos, esta técnica, cuyas siglas en inglés son MLE (maximum likelihood estimation), nos permite encontrar la distribución de probabilidad que mejor estima un cierto conjunto de datos. Para ello consideramos dos pasos:

1. Escogemos una distribución $P(X;\theta)$, con un conjunto de parámetros θ, dado un conjunto de datos X.
2. Seleccionamos los valores de los parámetros $\theta = \hat{\theta}$ que mejor ajustan los datos siguiendo la premisa que serán aquellos tales que:

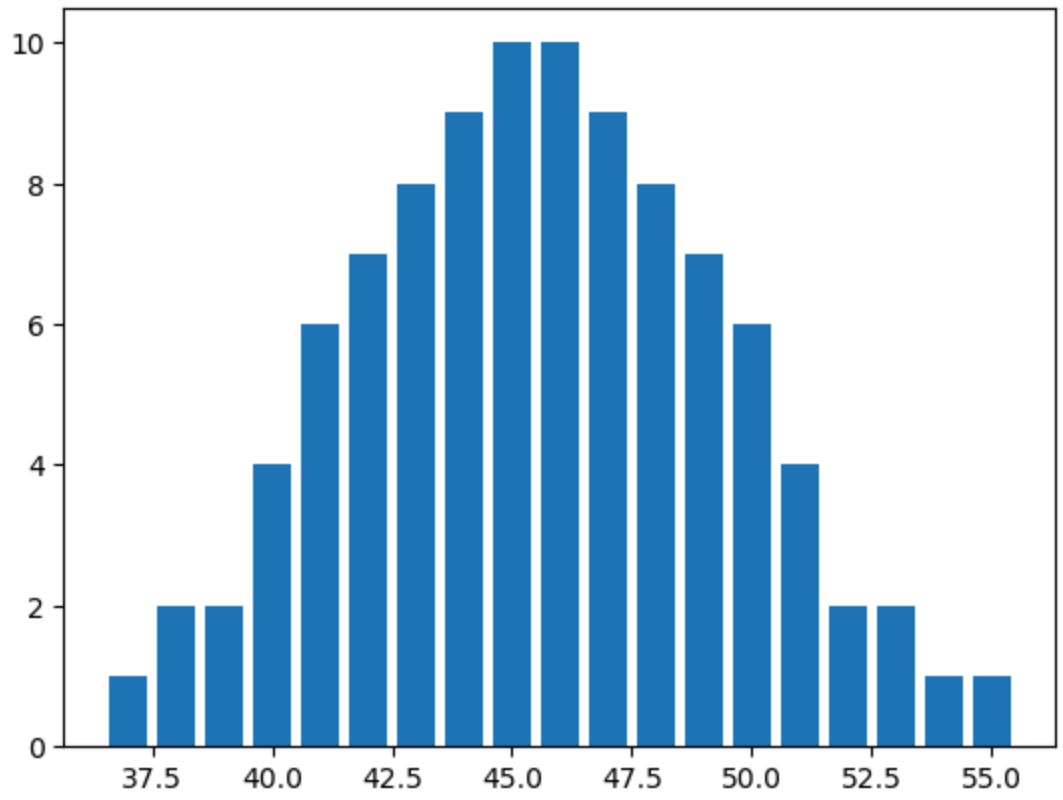
$$P(X,\theta) = L(X,\hat{\theta}) = \max_{\theta} \{L(X,\theta)\}$$

Así pues, consideremos un conjunto de datos x_i como el siguiente, el cual trabajamos en una clase pasada:

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# recuerda que este archivo lo puedes bajar de: https://seattlecentral.edu/qelp/sets/057/057.html
df = pd.read_excel('dataset/s057.xls')
arr = df['Normally Distributed Housefly Wing Lengths'].values[4:]
values, dist = np.unique(arr, return_counts=True)
print(values)
plt.bar(values, dist)
```

```
[37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55]
```

Out[ ]: <BarContainer object of 19 artists>



Si buscamos la distribución que mejor ajuste los datos, lo más razonable es pensar en una gaussiana o normal, y como ya se mostró en una notebook anterior, la distribucion gaussiana ajusta muy bien estos datos. En esta notebook veremos otra cara de la moneda sobre este mismo ejercicio.

RETO

Usando MLE vas a demostrar que si asumimos que los datos vienen descritos por una función gaussiana:

$$P(X;\theta) = P(X;\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left[-\frac{1}{2}\left(\frac{X-\mu}{\sigma}\right)^2\right]$$

Entonces los parámetros que mejor ajustan los datos x_i de la celda anterior están dados por:

$$\mu = \frac{1}{n}\sum_i^n x_i$$

$$\sigma^2 = \frac{1}{n}\sum_i^n (x_i-\mu)^2$$

donde *n* es la cantidad de datos.

Paso a paso (a mano)

1) Escribe la verosimilitud (likelihood)

Considera que en este caso asumimos que cada punto sigue una distribución normal

$$L(X;\theta) = \prod_i^n P(x_i;\theta)$$

y por lo tanto la verosimilitud está dada por:

$$L(X;\mu,\sigma) = L(\mu,\sigma) = \prod_i^n \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{\frac{-1}{2\sigma^2}(x_i-\mu)^2\right\}$$

2) Calcula las ecuaciones del valor máximo

Usando MLE se deben calcular los parámetros que conduzcan al máximo de probabilidad:

$$\max L(\mu,\sigma) \rightarrow \max \log L(\mu,\sigma)$$

donde consideramos el logaritmo de la verosimilitud. Demuestra que esto equivale a:

$$\max L(\mu, \sigma) = \min \left(n \log \sigma + \frac{1}{2\sigma^2} \sum_i (x_i - \mu)^2 \right)$$

Esto equivale a encontrar el minimo de la función:

$$f(\mu, \sigma) = n \log \sigma + \frac{1}{2\sigma^2} \sum_i (x_i - \mu)^2$$

lo cual se hace derivando parcialmente la función respecto a ambas variables e igualando a cero.

$$\frac{\partial L}{\partial \mu} = 0$$

$$\frac{\partial L}{\partial \sigma} = 0$$

3) Resuelve el sistema de ecuaciones resultante

Al final habrás obtenido un par de ecuaciones cuyas incógnitas son los parámetros:

$$\frac{-2}{\sigma^2} \sum_i (x_i - \mu) = 0$$

$$\frac{n}{\sigma} - \frac{1}{\sigma^3} \sum_i (x_i - \mu)^2 = 0$$

Resuélvelas y habrás llegado a las fórmulas indicadas previamente para los parámetros óptimos.

¿No son estas justamente las definiciones típicas de la media y la desviación estándar para un conjunto de datos?

¡Sí, así es! Esto indica que justamente los parámetros son óptimos cuando la desviación estándar y media de los datos coinciden con aquellas de la distribucion gaussiana.

Comprobación numérica

Vamos ahora a hacer una comprobación numérica de que esos parámetros efectivamente ajustan de manera óptima los datos. Construye funciones en Python que te permitan calcular directamente los parámetros óptimos según las ecuaciones encontradas:

$$\mu = \frac{1}{n} \sum_i^n x_i$$

$$\sigma^2 = \frac{1}{n} \sum_i^n (x_i - \mu)^2$$

In [ ]:
```python
## parámetros optimos a partir del cálculo con MLE
## completa con tu código aquí:

def optimal_mu(arr=arr):
  pass

def optimal_sigma(arr=arr):
  pass
```

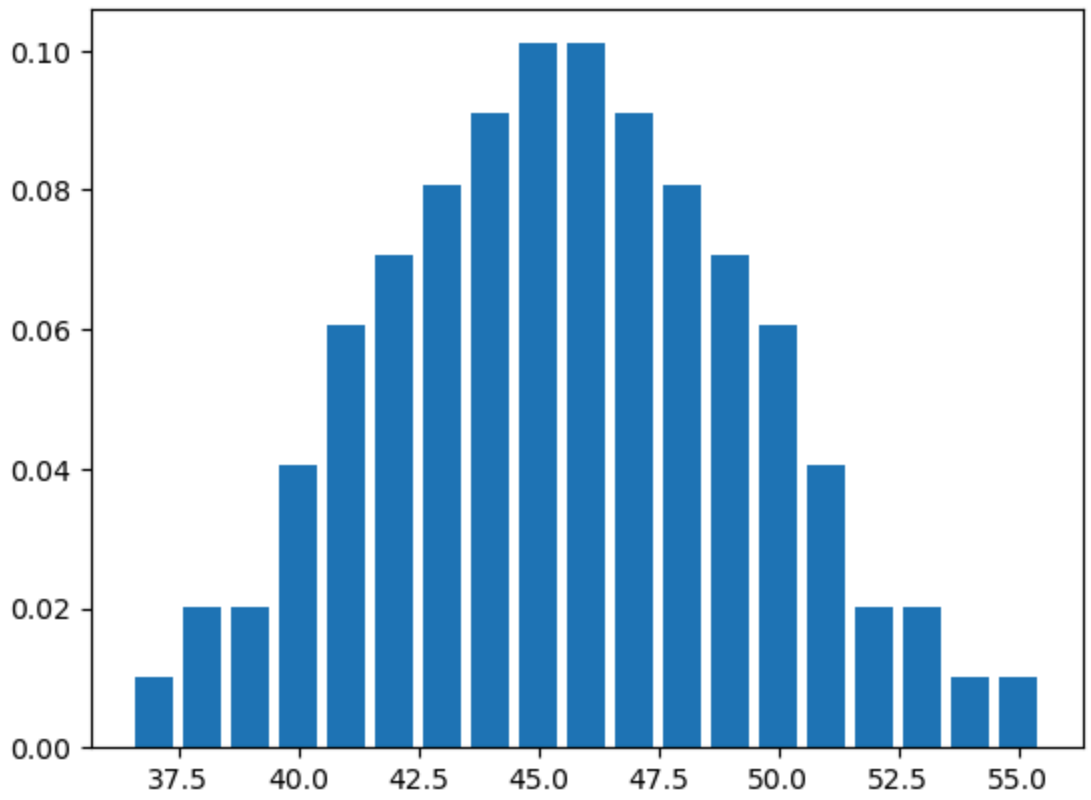
In [ ]:
```python
print(optimal_mu(), arr.mean())
print(optimal_sigma(), arr.std())
```

```
None 45.5959595959596
None 3.8003699764580126
```

De manera que luego puedas encontrar que ajustan muy bien los datos como sucedió en la notebook de la clase 9.

In [ ]:
```python
from scipy.stats import norm

values, dist = np.unique(arr, return_counts=True)
plt.bar(values, dist/len(arr))

dist = norm(optimal_mu(), optimal_sigma())
x = np.arange(30, 60, 0.1)
y = [dist.pdf(value) for value in x]
plt.plot(x,y)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[21], line 8
      6 dist = norm(optimal_mu(), optimal_sigma())
      7 x = np.arange(30, 60, 0.1)
----> 8 y = [dist.pdf(value) for value in x]
      9 plt.plot(x,y)

File ~/miniforge3/lib/python3.12/site-packages/scipy/stats/_distn_infrastructure.py:555, in rv_continuous_frozen.pdf(self, x)
    554 def pdf(self, x):
--> 555     return self.dist.pdf(x, *self.args, **self.kwds)

File ~/miniforge3/lib/python3.12/site-packages/scipy/stats/_distn_infrastructure.py:1988, in rv_continuous.pdf(self, x, *args, **kw
ds)
   1986 args = tuple(map(asarray, args))
   1987 dtyp = np.promote_types(x.dtype, np.float64)
-> 1988 x = np.asarray((x - loc)/scale, dtype=dtyp)
   1989 cond0 = self._argcheck(*args) & (scale > 0)
   1990 cond1 = self._support_mask(x, *args) & (scale > 0)

TypeError: unsupported operand type(s) for -: 'float' and 'NoneType'
```



Como te habrás dado cuenta los valores óptimos de los parámetros coinciden perfectamente con las definiciones de media y desviación estándar, así como sus contrapartes en Numpy `arr.mean()` y `arr.std()`.

Parte 5

Teorema de Bayes y machine learning Finalmente, encontrarás en la siguiente notebook un ejemplo donde se profundiza el desarrollo de un clasificador de Naive Bayes. Aquí verás por qué es importante el uso de distribuciones iniciales para ajustar las verosimilitudes con el fin de poder aplicar tus modelos a datos que no estaban en el dataset original de entrenamiento:

https://colab.research.google.com/drive/1XbQ6-5Ax8Pksik2MR2kWB87ay85RMX2b?usp=sharing

Ahora que tienes claros estos conceptos, a través del desarrollo de estos retos, espero que hayas disfrutado del curso y que con esto tengas toda la motivación para continuar con lo que sigue en temas de matemáticas, machine learning y ciencias de datos en general. Nos vemos en la próxima.

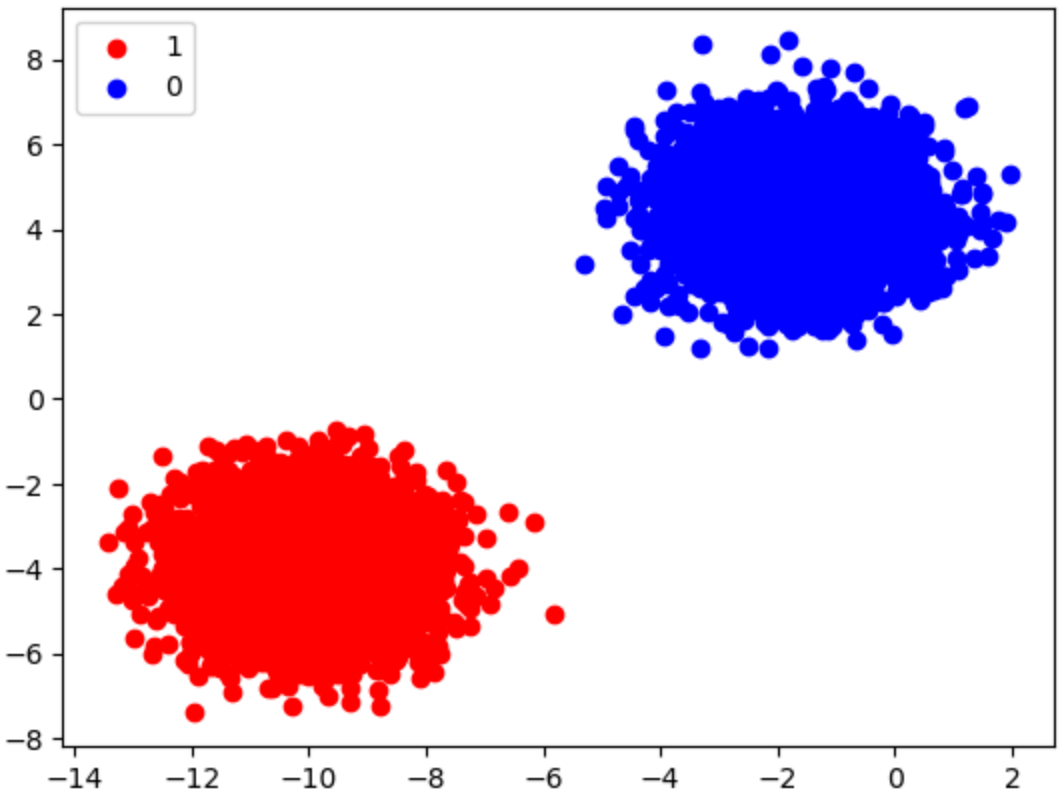
¡Nunca pares de aprender! 💚

Ejemplo con clasificador de Naive Bayes

Consideremos un conjunto de datos artificial sobre el cual podamos probar un clasificador de Naive Bayes:

In [ ]:
```
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import norm

X, y = make_blobs(n_samples=10000, centers=2, n_features=2, random_state=1)

# esta función ajusta una gausiana
# a un conjunto 'data'
def fit_distribution(data):
  mu = data.mean()
  sigma = data.std()
  dist = norm(mu, sigma)
  return dist

plt.scatter(X[y==1][:,0], X[y==1][:,1], label = '1', color='red')
plt.scatter(X[y==0][:,0], X[y==0][:,1], label = '0', color = 'blue')
plt.legend()
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x7fe90d21f7a0>
```



Consideramos un modelo de clasificacion de Naive Bayes:

$$P(c|x) = P(x|c)P(c)$$

donde $P(c)$ es la probabilidad prior dada una clase c y $P(x|c)$ es la verosimilitud de x dada la una clase c, con Naive Bayes esto resulta en:

$$P(c|x) = P(c)\prod_i P(x_i|c)$$

Lo cual para nuestro caso (`n_features=2`) se traduce en:

$$P(c|x) = \underbrace{P(c)}_{\text{prior}}\underbrace{P(x_0|c)P(x_1|c)}_{\text{likelihood}}$$

In [ ]:
```python
# calculamos priors
def prior(c):
  return len(X[y==c])/len(X)

# tenemos cuatro posibles distribuciones a ajustar (verosimilitud)
def distX0(c):
  if c==0:
    return fit_distribution(X[y==0][:,0])
  elif c==1:
    return fit_distribution(X[y==1][:,0])

def distX1(c):
  if c==0:
    return fit_distribution(X[y==0][:,1])
  elif c==1:
    return fit_distribution(X[y==1][:,1])

# verosimilitud
def likelihood(X, c):
  return distX0(c).pdf(X[0])*distX1(c).pdf(X[1])

# posterior
def probability(c, X):
  return prior(c)*likelihood(X,c)

predictions = [np.argmax([probability(0, vector), probability(1, vector)]) for vector in X]
```

Al final la distribución posterior nos da la probabilidad de que un dato `X` corresponda a una clase `c`. Luego de esto evaluamos el ajuste del modelo de clasificación al dataset artificial con una matriz de confusión:

In [ ]:
```python
from sklearn.metrics import confusion_matrix
confusion_matrix(y, predictions)
```

Out[ ]:
```
array([[5000,    0],
       [   0, 5000]])
```

Donde vemos que la distribución ajusta perfectamente los datos, de lo cual podemos también estimar la clase para otros puntos que no estaban inicialmente en el dataset:

In [ ]:
```python
def class_distribution(x, y):
  return np.argmax([probability(0, [x,y]), probability(1, [x,y])])

class_distribution(-6, 0)
```

Out[ ]:
```
np.int64(1)
```

In [ ]:
```python
class_distribution(-4, 0)
```

Out[ ]:
```
np.int64(0)
```

In [ ]:
```
plt.scatter(X[y==1][:,0], X[y==1][:,1], label = '1', color='red', marker = '*')
plt.scatter(X[y==0][:,0], X[y==0][:,1], label = '0', color = 'blue', marker='*')
plt.scatter(-6, 0, color = 'red', marker='s', s=53)
plt.scatter(-4, 0, color = 'blue', marker='s', s=53)
plt.legend()
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x7fe90cd816a0>
```


En este plot anterior se evidencia cómo el clasificador basado en una distribución posterior puede clasificar puntos que no estaban en el conjunto de datos inicial (puntos con forma de cuadrado), permitiendo de esta manera extrapolar las funciones de clasificación mas allá de los datos iniciales.