😎 Estadística con NumPy

La estadística nos ayuda a entender mejor los datos. Utilizándola podremos inferir, con cierta certeza, que probabilidades de ocurrencia existe sobre algo y nos ayuda a obtener propiedades acerca de los datos.

Dado:

In [5]:
```
import numpy as np
import seaborn as sns

taxis = sns.load_dataset('taxis')
taxis.head(2)
```

Out[5]:

	pickup	dropoff	passengers	distance	fare	tip	tolls	total	color	payment	pickup_zone	dropoff_zone	pickup_borough	dropoff_borough
0	2019-03-23 20:21:09	2019-03-23 20:27:24	1	1.60	7.0	2.15	0.0	12.95	yellow	credit card	Lenox Hill West	UN/Turtle Bay South	Manhattan	Manhattan
1	2019-03-04 16:11:55	2019-03-04 16:19:00	1	0.79	5.0	0.00	0.0	9.30	yellow	cash	Upper West Side South	Upper West Side South	Manhattan	Manhattan

In [4]:
```
np.array(taxis.distance)
```

Out[4]:
```
array([1.6 , 0.79, 1.37, ..., 4.14, 1.12, 3.85])
```

A continuación se explican conceptos estadísticos. Utilizando el conjunto de datos dado, aplica los mismos, crea los gráficos necesarios y obtén conclusiones en cada caso.

Media o promedio

Es útil para obtener el centro de un conjunto de datos. NumPy incluye una función para calcular la media de los arreglos: **np.mean**.

In [2]:
```
arr = np.array([5, 10.4, 4, 3, 6.6])
np.mean(arr)
```

Out[2]:
```
5.8
```

Redondear

¿Puede redondearse un valor float a un número específico de puntos decimales?

Puede usar la función de Python **round()** la cual recibe dos parámetros: número a redondear y número de decimales. NumPy provee también una forma de redondear cada valor en un array a través de la función **np.round()**. Funciona similar al round de python porque recibe ambos parámetros.

In [3]:
```
# redondeando un número con Python
numero = 1.234
print(round(numero, 2))
print(round(numero))
```

```
1.23
1
```

In [4]:
```
# redondeando un array con NumPy
array = np.array([1.11, 2.22, 3.33, 4.44])
print(np.round(array, 1))
```

```
[1.1 2.2 3.3 4.4]
```

Mean y operaciones lógicas

También podemos usar **np.mean()** para **calcular el porcentaje de elementos de un array que tienen ciertas propiedades**.

Recordemos que los operadores lógicos evalúan cada ítem en un array para ver si cumple una condición. Si el ítem cumple, evalúa a **True = 1** o **False = 0**.

Cuando **np.mean()** calcula una sentencia lógica, la media que da como resultado será equivalente al número total de True's dividido por el largo (len) del array.

In [5]:
```
respuestas_encuesta = [5, 10.4, 4, 3, 6.6]
respuestas_array = np.array(respuestas_encuesta)
np.mean(respuestas_array) ## 5.8
```

Out[5]:
```
5.8
```

In [6]:
```
np.mean(respuestas_array > 5)
```

Out[6]:
```
0.4
```

¿Qué hizo la última línea de código?

Evaluó las respuestas que son mayores a 5, y les asigna el valor de 1 (True), y las divide por el largo (len). El resultado nos dice que el 40% de los que respondieron, dieron una calificación de más de 5

Media de los valores

¿Cómo obtenemos la media de los valores, en vez del porcentaje de valores que satisfacen una condición?

In [7]:
```
temperaturas = np.array(list(range(-20, 55, 5)))
temperaturas
```

Out[7]:
```
array([-20, -15, -10,  -5,   0,   5,  10,  15,  20,  25,  30,  35,  40,
        45,  50])
```

Primero seleccionamos los datos específicos basados en una condición, el cual creará un nuevo array para todas las temperaturas mayores a 10: **temperaturas[temperaturas > 10]**

Luego aplicamos la función np.mean() a estos datos, y nos dará la media de los valores que cumplen con ese condicional:

In [8]:
```
temperaturas = np.array(list(range(-20, 55, 5)))
np.mean(temperaturas[temperaturas > 10])
```

Out[8]:
```
32.5
```

32.5 es el valor medio de las temperaturas mayores a 10

Media de Arrays de 2-Dimensiones

En un array de dos dimensiones, np.mean puede calcular las medias así como de los valores interiores.

Ejemplo: en un juego de baloncesto a un jugador le cometen 4 faltas y le conceden 2 tiros libres por cada falta. En el array 'encestar', cada array interior es un intento y cada item es un tiro libre. 1 representa una cesta exitosa, 0 representa un fallo.

In [9]:
```
encestar = np.array([[1, 0],
                     [0, 0],
                     [1, 0],
                     [1, 0]])
np.mean(encestar)
```

Out[9]:
```
0.375
```

0.375 significa es que de todos los tiros libres que tuvo oportunidad de encestar, logró el 37.5% de ellos.

Con numpy también podemos calcular la media de los arrays internos, especificando si lo queremos hacer sobre la fila (axis 1) o sobre la columna (axis 0)

In [10]:
```
np.mean(encestar, axis=1)
```

Out[10]:
```
array([0.5, 0. , 0.5, 0.5])
```

Este resultado significa es que en la primera de las cuatro faltas personales que le cobraron al jugador, encesto el 50%. En la segunda falta encesto el 0% y en las demás faltas encesto también el 50%

In [11]:
```
np.mean(encestar, axis=0)
```

Out[11]:
```
array([0.75, 0.  ])
```

Este resultado significa que en los primeros intentos tuvo una efectividad del 75% mientras que en los segundos intentos siempre falló los tiros libres, ya que no encesto ninguna.

Valores Atípicos (Outliers)

La media es un valor útil para entender rápidamente diferentes partes de los datos, sin embargo, está altamente influenciada por valores específicos del conjunto.

¿Qué pasa si uno de esos valores es significativamente diferente del resto?

Los valores que no se ajustan dentro de la mayoría del set de datos, son conocidos como **outliers** (valores atípicos)

Es importante identificar valores atípicos porque si pasan desapercibidos, pueden sesgar nuestros datos y conducir a errores en nuestro análisis (como la determinación de la media).

También pueden ser útiles para señalar errores en nuestra recopilación de datos. Cuando somos capaces de identificar los outliers, podemos determinar si es debido a un error en la recolección de datos o si representan una desviación significativa pero real de la media.

Ejemplo:

In [12]:
```
estaturas_estudiantes = [1.6, 1.55, 1.7, 1.68, 1.8, 3.9, 1.67, 1.70, 1.72]
estaturas_estudiantes
```

Out[12]:
```
[1.6, 1.55, 1.7, 1.68, 1.8, 3.9, 1.67, 1.7, 1.72]
```

¿Se observa algún error en los datos?

El valor de 3.9 es probable que sea un error.

¿ Y la siguiente lista?

In [13]:
```
estaturas_estudiantes = [1.6, 1.3, 1.7, 1.68, 1.8, 1.8, 1.67, 1.70, 1.72]
estaturas_estudiantes
```

Out[13]: [1.6, 1.3, 1.7, 1.68, 1.8, 1.8, 1.67, 1.7, 1.72]

Se observa un valor de 1.3. Puede que no sea un error, puede ser una persona de estatura baja, por tanto sigue siendo un valor válido dentro de la lista.

¿Y en la próxima lista?

In [14]:
```
estaturas_estudiantes = [1.6, 1.3, 1.7, 2.1, 1.8, 1.8, 1.67, 1.70, 1.72]
estaturas_estudiantes
```

Out[14]: [1.6, 1.3, 1.7, 2.1, 1.8, 1.8, 1.67, 1.7, 1.72]

Se observa un valor de 2.1 lo cual es posible aunque puede considerarse un valor atípico en comparación con los demás valores de la lista.

¿Como determinar outliers?

Algunas veces puede hacerse solo mirando el dataset e indicando los valores que parecen mayores o menores que otros valores. Pero es un método manual, propenso a errores y difícil de realizar.

Una forma de identificar rápidamente outliers es ordenando los datos de mayor a menor o viceversa. Una vez que los datos están ordenados, puede observarse el inicio y el final del dataset para detectar si hay valores más allá del rango esperado. Para ello podemos usar la función de NumPy, **np.sort**.

In [15]:
```
estaturas_estudiantes = [1.6, 1.3, 1.7, 2.1, 1.8, 1.8, 1.67, 1.70, 1.72]
np.sort(estaturas_estudiantes)
```

Out[15]: array([1.3 , 1.6 , 1.67, 1.7 , 1.7 , 1.72, 1.8 , 1.8 , 2.1])

También pueden aplicarse otros cálculos estadísticos para determinar un rango de valores, en el caso que algún valor salga del rango, es un outlier. Esto se puede hacer con cuantiles. Algo así:

[primer_cuantil - 1.5 * IQR, tercer_cuantil + 1.5 * IQR]

donde IQR es **"Interquartile Range (Rango Intercuantil)"**.

Mediana

Otra métrica importante que puede sere utilizada para el análisis de datos es la mediana **np.median**. La mediana es el valor del medio de un set de datos que ha sido ordenado en términos de magnitud (del más pequeño al más grande). Si el largo del set de datos es un número impar, la mediana sería el valor central. Hay que recordar ordenar previamente. En el siguiente ejemplo, la mediana sería 3

In [16]:
```
numeros = [1, 1, 2, 3, 4, 5, 5]
numeros = np.array(numeros)
np.median(numeros)
```

Out[16]: 3.0

Si el largo del set de datos es un número par, la mediana sería el valor a medio camino entre los dos valores centrales. Así que en el siguiente ejemplo, la mediana sería 3.5

In [17]:
```
numeros = [1, 1, 2, 3, 4, 5, 5, 6]
numeros = np.array(numeros)
np.median(numeros)
```

Out[17]: 3.5

Los valores centrales aquí son 3 y 4, por tanto decimos que 3.5 está a medio camino entre ambos valores.

¿Pero qué pasa si tenemos un conjunto de datos muy grande y con muchos valores?

Sería muy tedioso ordenar y contar todos los valores. NumPy realiza ambos pasos sin importar si es un array de largo par o impar.

In [18]:
```
numeros = [50, 38, 291, 59, 14]
numeros = np.array(numeros)
np.median(numeros)
```

Out[18]: 50.0

Diferencias entre la Media y la Mediana

Estos dos valores pueden ser muy similares según el set de datos, pero son diferentes y representan conceptos diferentes.

- La **mediana** retorna el valor en una posición central de un dataset ordenado. Si el largo del dataset es par, será el promedio de los dos valores del medio. Es conocido también como el "50th percentile".
- La **media** es el promedio de todos los valores del set de datos.

Los outliers pueden influir mucho en la media, pero no en la mediana.

In [19]:
```
valores = [1, 1, 1, 2, 900]
valores = np.array(valores)
np.mean(valores)
```

Out[19]: 181.0

In [20]:
```
np.median(valores)
```

Out[20]: 1.0

Observar en este ejemplo cómo la media se ve fuertemente impactada por valores atípicos, mientras que la mediana representa mejor la mayoría de los datos.

Representación de Mayoría

¿Cómo podemos decir si un valor representa mejor la mayoría de los datos?

Podemos hacer lo siguiente:

- Primero remover los outliers del set de datos.
- La media ahora puede ser calculada con estos nuevos valores para una representación más acertada de los datos.
- Luego podemos verificar si cualquiera de los dos valores (media y mediana) está más cerca de esa media y podría decirse que ésta es la que representa mejor la mayoría de datos.

In [21]:
```
valores = [1, 1, 1, 2, 900]
valores = np.array(valores)
np.mean(valores)
```

Out[21]: 181.0

In [22]:
```
np.median(valores)
```

Out[22]: 1.0

In [23]:
```
valores_sin_outliers = [1, 1, 1, 2]
np.mean(valores_sin_outliers)
```

Out[23]: 1.25

¿Cuál de las estadísticas (media y mediana) calculadas sobre el array valores, está más cercana al valor dado por la media calculada sobre el array valores_sin_outliers?

Rta: La mediana está más cercana, por tanto representa mejor la mayoría de los datos para este caso en particular.

Percentiles (Cuantiles)

Como se vió anteriormente, la mediana está en el medio del set de datos. Es el número en el cual el 50% de los ítems están por debajo y 50% de los ítems están por arriba.

¿Pero qué pasa si queremos encontrar un punto en el cual el 40% de los ítems están por debajo y el 60% de los ítems están por arriba?

Este tipo de punto es llamado **percentile**. El Nth percentile está definido como el punto N% de ítems que están por debajo de él.

Así que el punto donde el 40% de los ítems está por debajo, es llamado el 40th percentile.

Los percentiles son medidas útiles porque pueden decirnos donde está situado un valor en particular dentro de un conjunto de datos muy grande.

Ejemplo:

In [24]:
```
d = [1, 2, 3, 4, 4, 4, 6, 6, 7, 8, 8]
d
```

Out[24]: [1, 2, 3, 4, 4, 4, 6, 6, 7, 8, 8]

- Hay 11 números en el set de datos.
- El 40th percentile tendrá el 40% de los 10 números restantes por debajo (40% de 10 es 4) y 60% de los números encima de él (60% de 10 es 6).
- Así que el 40th percentile es 4 para este ejemplo.

En NumPy podemos calcular los percentiles usando la función **np.percentile**, que toma dos argumentos: el array y el percentile a calcular.

In [25]:
```
d = [1, 2, 3, 4, 4, 4, 6, 6, 7, 8, 8]
d2 = np.array(d)
np.percentile(d2, 40)
```

Out[25]: 4.0

¿Qué nos dicen los percentiles?

- Nos dice cómo se encuentra un valor, de un set de datos, en relación con el resto de datos.
- Un ejemplo muy conocido es para estandarizar puntajes (**scores**).
- Los percentiles pueden decirnos que tan bien le fue a alguien en relación con otros puntajes.
- En Data Analysis sirve para determinar el rango interquartile (**3rd -1st percentile**) para saber que tan disperso está el conjunto de datos.

Algunos percentiles tienen nombres específicos:

- El **25th percentile** es llamado **first quartile** (primer cuantil)

- El **50th percentile** es llamado **the median** (la mediana)
- El **75th percentile** es llamado **third quartile** (tercer cuantil)
- El **valor mínimo**, el first quartile, la mediana, el third quartile y el **valor máximo** de un set de datos son llamados el **"five-number summary"**. Este set de números es siempre bueno calcularlos cuando obtenemos un set de datos por primera vez.

La diferencia entre el first quartile y third quartile es un valor llamado interquartile range (Rango Intercuartil). Decimos que el 50% del set de datos estará siempre dentro del rango intercuartil.

El rango intercuartil nos da una idea de la dispersión de nuestros datos. Cuanto menor sea el valor del rango intercuartil, menor será la desviación en nuestro dataset. Cuanto mayor sea el valor, mayor será la desviación.

In [26]:
```
d = [1, 2, 3, 4, 4, 4, 6, 6, 7, 8, 8]
d2 = np.array(d)
primer_cuantil = np.percentile(d2, 25)
tercer_cuantil = np.percentile(d2, 75)

print(primer_cuantil) # 3.5
print(tercer_cuantil) # 6.5
print("Rango Intercuantil: ", tercer_cuantil - primer_cuantil)
```

```
3.5
6.5
Rango Intercuantil:  3.0
```

Five-number summary

¿Por qué es útil el five-number summary?

- Consiste en estos valores del dataset: mínimo, 1st quartile, median, 3rd quartile y máximo.
- Podemos calcular el IQR (Interquartile Range) que nos dice que tan dispersa estan los datos.
- Es útil también para determinar los outliers con los valores máximo y mínimo.
- Es un resumen no afectado por los outliers.
- Podemos comparar las 5 métricas de resumen con otros datasets para ver similitudes o diferencias.

Varianza

La varianza es una medida de dispersión que se utiliza para representar la variabilidad de un conjunto de datos respecto de la media aritmética de los mismos, esto quiere decir que la varianza mide qué tanto varía un grupo de datos (una población o muestra) respecto a su promedio. En numpy existe la función **np.var()**.

Ejemplo: tenemos un conjunto de calificaciones de 5 diferentes alumnos y queremos conocer qué tanto varían las calificaciones con respecto a el promedio:

calificaciones = 9, 10, 8, 9, 7

Obtenemos el promedio de las calificaciones:

9 + 10 + 8 + 9 + 7 / 5 = 8.6

Calculamos la varianza con todas las calificaciones:

(9–8.6)² + (10–8.6)² + (8–8.6)² + (9–8.6)²+ (7–8.6)² = 0.16 + 1.96 + 0.36 + 0.16 + 2.56 / 5 = 1.04

Varianza = 1.04

In [27]:
```
calificaciones =[9, 10, 8, 9, 7]
varianza = np.var(calificaciones, ddof = 1)
print("La varianza es de {0:.2f}".format(varianza))
```

```
La varianza es de 1.30
```

*El parámetro **ddof** "Delta Degrees of Freedom" define el número de grados de libertad que ayuda a ajustar el sesgo con muestras de entrada limitadas en relación con la varianza de la población. El valor por defecto en numpy 0*

Más información (https://numpy.org/doc/stable/reference/generated/numpy.var.html#numpy.var)

Desviación Estándar

Mientras que la media y la mediana nos habla sobre el centro de nuestros datos, ellas no reflejan el rango de datos. Aquí es donde llega la desviación standard.

Similar a los rangos intercuartiles, la desviación estándar nos dice que tan dispersa está nuestra data.

- A **mayor desviación estándar**, más dispersa está nuestra data del centro.
- A **menor desviación estándar**, más datos están agrupados alrededor de la media.

Con NumPy podemos usar la función llamada **np.std**.

In [28]:
```
numeros = np.array([65, 36, 52, 91, 63, 79])
np.std(numeros)
```

Out[28]: 17.716909687891082

Desviación Estándar Vs Rango Intercuartil

Desviación Estándar

- Toma en cuenta todos los valores del dataset, incluyendo outliers.
- Depende de la media, porque este valor es usado para determinar qué tanto están los datos desviados de la media del dataset.
- Es importante cuando necesitamos la **varianza** de un set de datos, como en regresiones lineales.
- Es más confiable cuando los datos están normalizados y no sesgados (**skewed**).

Rango Intercuartil

- Nos dice que tan dispersos están los datos. A mayor valor, más dispersa están los datos y viceversa.
- No toma en cuenta todos los datos del conjunto de datos, sino principalmente las posiciones cuando el set está ordenado.
- No se afecta tanto por los outliers, o por datos no normalizados o sesgados.
- Utilizar ambos cuando analizamos datos es mejor que usar uno solo y tenemos mejores **insights**.

Distribución Estadística

La distribución estadística, también conocida como distribución de probabilidad, muestra la probabilidad o número de veces que cierto resultado ocurra dado un número de intentos.

Esto puede ayudarnos a observar y determinar la probabilidad de que un resultado ocurra. Teniendo en cuenta que siempre hay incertidumbre, esto puede ayudarnos a tomar decisiones más certeras basadas en probabilidades. La forma de la distribución puede ser categorizada.

Histogramas

Cuando vemos por primera vez un set de datos, vamos a querer entender rápidamente ciertas cosas acerca de él, como:

- Algunos valores ocurren más seguido que otros?
- ¿Cuál es el rango del set de datos? (el máximo y mínimo)
- ¿Hay muchos valores atípicos? (outliers)

Podemos visualizar esto con un gráfico llamado histograma.

Gráficos de barras vs histogramas

Aunque parecen similares, los histogramas y los gráficos de barras son diferentes y sirven para propósitos específicos.

Histogramas:

- Son usados para graficar distribuciones o frecuencias para datos cuantitativos.
- Visualmente todas las barras se están tocando, y no hay espacio entre ellas.
- Las barras representan los valores del dataset que caen dentro de cada rango de valores.

Gráficos de Barras:

- Son usados para agrupar datos basados en categorías.
- Visualmente están esparcidas las barras.
- Cada barra representa cuántos de los datos caen dentro de ciertas categorías.

Creando un histograma

Supongamos que tenemos un set de datos muy grande y los datos están en un rango entre 0 y 50. Aquí podríamos querer saber cuántos datos caen o están entre 0 y 5, 6 y 10, 11 y 15, etc.

A este agrupamiento lo llamamos **bins** (o ubicaciones en español).

Todos los bins en un histograma serán del mismo tamaño. El ancho (width) de cada bin es la distancia entre max y min de cada bin. En nuestro ej, seria 5

Aquí podría surgir una pregunta, y es: **¿hay un ancho ideal de bins para los histogramas?**

No existe un ancho que sirva para todos los problemas. Es totalmente independiente a la cantidad de datos y al rango que existe entre ellos.

Lo ideal es que tenga una significativa representación de los datos. Elegir valores muy bajos o muy altos podría dificultar la tarea de analizar y entender los datos.

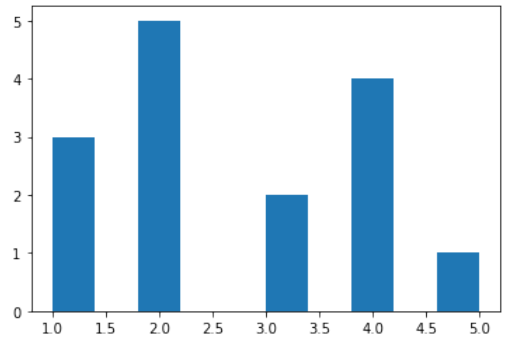
Ejemplo:

In [29]:

```
## importamos las librerías
from matplotlib import pyplot as plt
import numpy as np

## dibujamos el histograma
data = np.array([1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 4, 4, 4, 4, 5])
plt.hist(data)

plt.show()
# plt.savefig("mi-histograma.png")
```



El histograma automáticamente se crea con 10 bins.

Si usted quiere un valor diferente, debe usar la palabra reservada bins en plt.hist.

Ejemplo: **plt.hist(data, bins=5)**

Si se desea un rango diferente, puede pasarse un máximo y mínimo usando la palabra reservada **range**.

Ejemplo: **plt.hist(data, range=(20, 51))**

Selección de bins

¿Podría afectar la apariencia de una distribución la elección de bins?

Si, porque puede cambiar la interpretación de los datos, aunque no afectará el set de datos en sí.

Por ejemplo si ponemos un bin muy bajo, bins=2 las barras probablemente no darán mucha información acerca de la distribución de los datos. Por lo tanto es importante elegir un buen tamaño de bins.

Tipos de distribución

Los histogramas y sus set de datos pueden ser clasificados basados en la forma de los valores graficados. Una manera de clasificar un dataset es contando el número de diferentes picos (peaks) presentes en el gráfico. Los picos representan concentración de datos:

- **Unimodal**: Un solo pico en el centro.
- **Bimodal**: Dos picos. Pasa cuando los datos contienen dos poblaciones diferentes.
- **Multimodal**: Tiene múltiples picos.
- **Uniforme**: No tiene picos, es plano.

La mayoría de los set de datos con los que trabajamos serán unimodales (un solo pico). Podemos subclasificar las distribuciones unimodales a su vez describiendo dónde está la mayoría de datos en relación al pico:

- **Unimodal - Simétrica**: mayoría de números en el centro.
- **Unimodal - Sesgado a la derecha**: la mayoria de datos estan en la izquierda.
- **Unimodal - Sesgado a la izquierda**: la mayoria de datos estan a la derecha.

El tipo de distribución afecta la posición de la media y la mediana. En distribuciones muy sesgadas la media (mean) se vuelve menos útil.

La distribución normal

El tipo de distribución más común en estadística es la distribución normal, la cual es una distribución unimodal y simétrica.

Muchas cosas pueden conformar una distribución normal, por ejemplo: la altura de un gran número de personas, la presión sanguínea de un grupo de personas sanas, etc.

La distribución normal está definida por la media y la desviación standard.

La media nos dice el "medio" de la distribución y la desviación estándar nos dice el "ancho" de la distribución.

A mayor desviación estándar nos lleva a una distribución más ancha, y a menor desviación standard, más delgada la distribución.

Podemos usar una función de NumPy para generar un set de números que tengan la forma de una distribución normal, **np.random.normal**, que recibe los siguientes argumentos:

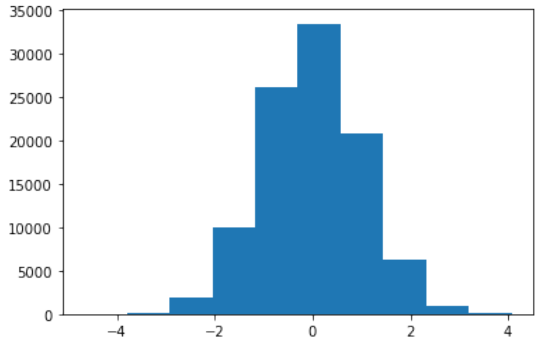
- **loc**: la media para la distribución normal.
- **scale**: la desviación standard de la distribución.
- **size**: el número de números aleatorios a generar.

In [30]:
```
from matplotlib import pyplot as plt
import numpy as np

a = np.random.normal(0, 1, size=100000)
a
```

Out[30]:
```
array([ 2.0158294 , -1.40947019,  0.20760192, ..., -1.67443401,
        1.29757983, -0.85418083])
```

In [31]:
```
plt.hist(a)
plt.show()
# plt.savefig("distribucion-normal.png")
```



Distribución normal vs desviación estándar

Para entender mejor la relación entre a la desviación estándar y la distribución normal debemos entender que en una distribución normal, sabemos ahora, que la media y la desviación standard determinan ciertas características de la forma de los datos, **¿Pero cómo exactamente?**.

Supongamos que tenemos una distribución normal con una media de 50 y una desviación standard de 10. Cuando decimos "dentro de una desviación standard de la media", esto es lo que estamos diciendo matemáticamente:

```
rango_bajo = mean - std
           = 50 - 10
           = 40
```

```
rango_alto = mean + std
           = 50 + 10
           = 60
```

Esto significa que podemos esperar que cerca del 68% de nuestro dataset esté entre 40 y 60 de esta distribución. No importa cual media o desviación estándar elijamos, el 68% de nuestros ejemplos caerán en +/-1 desviación estándar de la media.

Estas son algunas reglas de la distribución normal:

```
68% de nuestros datos caerán entre +/- 1 Desviaciones estándar de la media:
    una_desviacion = [mean - std, mean + std]
```

```
95% de nuestros datos caerán entre +/- 2 Desviaciones estándar de la media:
    una_desviacion = [mean - (2 * std), mean + (2 * std)]
```

```
99.7% de nuestros datos caerán entre +/- 3 Desviaciones estándar de la media:
    una_desviacion = [mean - (3 * std), mean + (3 * std)]
```

La distribución binomial

La distribución binomial nos dice que tan probable es que un número de "aciertos" se den, dada una probabilidad de éxito y un número de intentos. Es importante porque nos permite conocer qué tan probable es cierto resultado, incluso cuando no es esperado.

Ejemplo:

- Los jugadores de basket normalmente encestan el 30% de los tiros libres.
- Un jugador tiene el chance de hacer 10 tiros libres en un juego.
- ¿Cuántos tiros libres se espera que enceste? Rta:/ 0.3 * 10 = 3
- Pero el jugador encesta 4 tiros libres, o sea un 40%
- ¿Esto es un resultado sorprendente? ¿Significa que él es mejor jugador de lo que se pensaba?

Las respuestas correctas son:

- No es tan sorprendente que un jugador haga 4 de 10
- Sin embargo es poco probable que haga 0 de 10 o que haga 10 de 10
- Incluso tiene pocas chances de hacer 7, 8 o 9 de 10

Comprobamos con código:

Para obtener las diferentes probabilidades de que un jugador haga 1, 2, 3 o más cestas por cada 10 intentos, puede usarse **np.random.binomial**, la cual puede determinar la probabilidad de diferentes resultados.

La función retorna el número de "éxitos" por cada "experimento". Y toma estos 3 argumentos:

- **N**: el número de intentos o ejemplos.
- **P**: la probabilidad de éxito.
- **size**: número de experimentos.

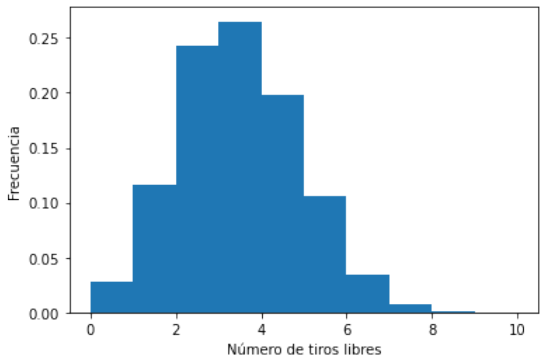
Continuando con el ejemplo de baloncesto, generamos 10.000 "experimentos" con 10 "intentos" cada uno, y una probabilidad de éxito del 30%:

In [32]:
```
plt.close() # esta línea la usamos para "limpiar" el gráfico anterior, de lo contrario pueden generarse dos histogramas
from matplotlib import pyplot as plt
import numpy as np

a = np.random.binomial(10, 0.3, size=10000)
print(a)

plt.hist(a, range=(0, 10), bins=10, density=True)
plt.xlabel("Número de tiros libres")
plt.ylabel("Frecuencia")
plt.show()
# plt.savefig("distribucion-binomial.png")
```

```
[4 2 3 ... 3 2 2]
```



Analizando las preguntas anteriores:

- Los jugadores de basket normalmente encestan el 30% de los tiros libres.
- Un jugador tiene el chance de hacer 10 tiros libres en un juego.
- ¿Cuántos tiros libres se espera que enceste? R:/ 0.3 * 10 = 3.
- Pero el jugador encesta 4 tiros libres, o sea un 40%.
- ¿Esto es un resultado sorprendente? ¿Significa que él es mejor jugador de lo que se pensaba?

Las respuestas correctas siguiendo el histograma que acabamos de crear son:

- No es tan sorprendente que un jugador haga 4 de 10.
- Sin embargo es poco probable que haga 0 de 10 o que haga 10 de 10.
- Incluso tiene pocas chances de hacer 7, 8 o 9 de 10

Otras conclusiones:

El jugador tenía 30% de chances de hacer el tiro libre. Pero en 10 intentos hace 4 tiros libres, aunque esperábamos que hiciera 3.

¿Qué probabilidades había de que hiciera esos 4 tiros?

- Podemos calcular el porcentaje de esa probabilidad usando la función np.mean.
- Tomando la media de una sentencia lógica nos dará un porcentaje de valores que satisfacen esa sentencia lógica.

Observemos los resultados

In [33]:
```
a = np.random.binomial(10, 0.3, size=10000)
print("La probabilidad de que haga 4 tiros libres es de {0:.2f}%".format((np.mean(a == 4)) * 100))
```

```
La probabilidad de que haga 4 tiros libres es de 19.18%
```

In [34]:
```
print("La probabilidad de que haga 8 de 10 tiros libres es de solo {0:.2f}%".format((np.mean(a == 8)) * 100))
```

```
La probabilidad de que haga 8 de 10 tiros libres es de solo 0.11%
```

In [35]:
```
print("La probabilidad de que haga 1 de 10 tiros libres es de {0:.2f}%".format((np.mean(a == 1)) * 100))
print("Lo que indica que es más probable que un jugador enceste pocos tiros libres a que enceste mucho tiros libres.")
```

```
La probabilidad de que haga 1 de 10 tiros libres es de 12.30%
Lo que indica que es más probable que un jugador enceste pocos tiros libres a que enceste mucho tiros libres.
```

In [36]:
```
print("La probabilidad de que haga más de 5 tiros libres es de solo el {0:.2f}%".format((np.mean(a > 5)) * 100))
```

```
La probabilidad de que haga más de 5 tiros libres es de solo el 5.08%
```

In [37]:

```
print("La probabilidad de que haga menos de 5 tiros libres es del {0:.2f}%".format((np.mean(a <= 5)) * 100))
print("Lo que indica que es altamente probable que ni siquiera enceste la mitad de los 10 tiros libres.")
```

```
La probabilidad de que haga menos de 5 tiros libres es del 94.92%
Lo que indica que es altamente probable que ni siquiera enceste la mitad de los 10 tiros libres.
```