Numpy
Estadísticas

CertiDevs

Índice de contenidos

1. Introducción

La **estadística** es una rama de las matemáticas que se ocupa de la recolección, análisis, interpretación, presentación y organización de datos.

La estadística tiene múltiples aplicaciones en una amplia variedad de campos, incluyendo *ciencias naturales*, *ciencias sociales*, *economía*, *medicina*, *industria* y *gobierno*, entre otros.

La estadística sirve para **diversos propósitos**, algunos de los cuales se describen a continuación:

- **Descripción y resumen de datos**: La estadística permite resumir grandes cantidades de datos utilizando medidas descriptivas como la media, mediana, moda, desviación estándar, varianza, coeficiente de correlación, etc. Estas medidas proporcionan información valiosa sobre las tendencias y patrones en los datos.
- **Estimación y predicción**: La estadística ayuda a realizar inferencias sobre una población a partir de una muestra de datos, lo cual es fundamental para la toma de decisiones. Por ejemplo, se pueden estimar parámetros de una población, como la media o la proporción, utilizando datos de una muestra y técnicas de inferencia estadística. Además, se pueden utilizar modelos estadísticos para predecir eventos futuros basados en datos históricos.
- **Análisis de relaciones entre variables**: La estadística permite analizar las relaciones entre dos o más variables, lo cual es útil para entender cómo interactúan distintos factores y cómo afectan los resultados de interés. Por ejemplo, se puede utilizar el coeficiente de correlación para medir la fuerza de la relación entre dos variables, o realizar análisis de regresión para explorar cómo una variable dependiente es afectada por una o más variables independientes.
- **Pruebas de hipótesis**: La estadística permite evaluar la validez de una afirmación o teoría utilizando datos de muestra. Las pruebas de hipótesis permiten comparar las hipótesis nula y alternativa y decidir si se debe rechazar la hipótesis nula a favor de la hipótesis alternativa, basándose en el nivel de significancia y el p-valor.
- **Diseño experimental y control de calidad**: La estadística es fundamental en el diseño de experimentos, ya que permite planificar estudios de manera eficiente y rigurosa, analizar los resultados y sacar conclusiones válidas. En el control de calidad, la estadística ayuda a monitorear y mejorar procesos de producción, reducir la variabilidad y garantizar que los productos cumplan con los estándares de calidad.
- **Toma de decisiones basada en datos**: La estadística proporciona herramientas para tomar decisiones informadas en presencia de incertidumbre. Permite evaluar riesgos, estimar beneficios y costos, y comparar diferentes alternativas basándose en datos y modelos probabilísticos.

La **estadística** se ocupa de la recolección, análisis, interpretación, presentación y organización de **datos**.

La **estadística** se basa en la **teoría de la probabilidad** porque la mayoría de los fenómenos que se estudian en estadística son de naturaleza aleatoria o incierta. La probabilidad proporciona las herramientas matemáticas y los fundamentos teóricos necesarios para analizar y comprender los datos observados.

La **teoría de la probabilidad** es una rama de las matemáticas que estudia los fenómenos aleatorios

y se ocupa de analizar, modelar y predecir los resultados posibles de experimentos o situaciones inciertas. En otras palabras, la **probabilidad** es una medida que **cuantifica la incertidumbre** de los eventos y ayuda a comprender cómo de probable es que ocurra un evento en particular.

2. Estadística en Numpy

NumPy, como una biblioteca de Python ampliamente utilizada para el manejo de datos numéricos, también proporciona funciones para realizar cálculos estadísticos básicos en sus arreglos.

- **Medidas de tendencia central**
  - Media: el promedio de los valores en un conjunto de datos.
  - Mediana: el valor central en un conjunto de datos ordenado.
  - Moda: el valor que ocurre con mayor frecuencia en un conjunto de datos.
- **Medidas de dispersión**
  - Rango: la diferencia entre el valor máximo y el mínimo en un conjunto de datos.
  - Varianza: la medida de la dispersión de los valores en un conjunto de datos en relación con la media.
  - Desviación estándar: la raíz cuadrada de la varianza, que mide la dispersión en las mismas unidades que los datos.
  - Rango intercuartílico (IQR): la diferencia entre el primer cuartil (Q1) y el tercer cuartil (Q3), que mide la dispersión en el 50% central de un conjunto de datos.
- **Cuartiles y percentiles**
  - Cuartiles: valores que dividen un conjunto de datos ordenado en cuatro partes iguales.
  - Percentiles: valores que dividen un conjunto de datos ordenado en 100 partes iguales.
- **Correlación y regresión**
  - Coeficiente de correlación: una medida de la relación lineal entre dos variables.
  - Regresión lineal: un modelo que describe la relación lineal entre una variable dependiente y una o más variables independientes.
- **Probabilidad y distribuciones**
  - Probabilidad: una medida de la posibilidad de que ocurra un evento.
  - Distribuciones de probabilidad: funciones que describen la probabilidad de ocurrencia de diferentes resultados en un experimento.
  - Distribución normal: una distribución de probabilidad simétrica en forma de campana que describe muchas variables en la naturaleza y en los procesos sociales.
  - Distribución binomial: una distribución de probabilidad discreta que describe el número de éxitos en una secuencia de ensayos de Bernoulli independientes.
  - Distribución de Poisson: una distribución de probabilidad discreta que describe el número de eventos que ocurren en un intervalo fijo de tiempo o espacio.
  - Distribución exponencial: una distribución de probabilidad continua que describe el tiempo

entre eventos en un proceso de Poisson.

- Distribución de t-Student: una distribución de probabilidad continua similar a la normal pero con colas más pesadas, utilizada principalmente en inferencia estadística cuando el tamaño de la muestra es pequeño.

- **Pruebas de hipótesis y estadística inferencial**
  - Hipótesis nula y alternativa: supuestos iniciales en una prueba de hipótesis que se comparan para determinar si hay evidencia suficiente para rechazar la hipótesis nula.
  - Valor p: la probabilidad de obtener un resultado al menos tan extremo como el observado, asumiendo que la hipótesis nula es cierta.
  - Nivel de significancia: un umbral predeterminado (generalmente 0.05) utilizado para decidir si se rechaza o no la hipótesis nula.
- **Intervalos de confianza**
  - Intervalo de confianza: un rango de valores dentro del cual se estima que se encuentra un parámetro desconocido de la población, con un nivel de confianza específico (por ejemplo, 95%).

Numpy ofrece funciones para calcular **estadísticas** en arrays, como la *media*, *mediana*, *desviación estándar* y *correlación*.

```
t = np.array([2, 4, 6, 8, 10])

# Media
u = np.mean(t)
print("Media:", u)

# Mediana
v = np.median(t)
print("Mediana:", v)

# Desviación estándar
w = np.std(t)
print("Desviación estándar:", w)

# Correlación
x = np.array([1, 2, 3, 4, 5])
y = np.array([2, 4, 6, 8, 10])
z = np.corrcoef(x, y)
print("Correlación:\n", z)
```

2.1. Media

La **media**, también conocida como **promedio**, es una medida de tendencia central que representa el valor típico o central de un conjunto de datos.

Se calcula sumando todos los valores en el conjunto de datos y dividiendo la suma por el número

total de valores.

La media es útil para resumir y comparar conjuntos de datos y puede ser aplicada en contextos como el análisis de rendimiento, las calificaciones de exámenes, la economía y la investigación científica.

A continuación, se muestran ejemplos de cómo calcular la media utilizando NumPy:

```
import numpy as np

# Ejemplo 1: Calcular la media de un arreglo 1D (una dimensión)
data1 = np.array([5, 8, 10, 15, 20])
mean1 = np.mean(data1)
print("Media del arreglo data1:", mean1)

# Ejemplo 2: Calcular la media de un arreglo 2D (dos dimensiones)
data2 = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
mean2 = np.mean(data2)
print("Media del arreglo data2:", mean2)

# Ejemplo 3: Calcular la media a lo largo de un eje específico en un arreglo 2D
row_mean = np.mean(data2, axis=1)  # Media a lo largo de las filas (eje 1)
print("Media a lo largo de las filas:", row_mean)

column_mean = np.mean(data2, axis=0)  # Media a lo largo de las columnas (eje 0)
print("Media a lo largo de las columnas:", column_mean)
```

En estos ejemplos, utilizamos la función `np.mean()` para calcular la media de arreglos de diferentes dimensiones. También vimos cómo calcular la media a lo largo de un eje específico en un arreglo 2D, lo cual es útil cuando se trabaja con arreglos multidimensionales y se desea obtener información estadística en una dirección específica.

2.2. Mediana

La **mediana** es otra medida de tendencia central que representa el valor que se encuentra exactamente en el medio de un conjunto de datos cuando estos están ordenados.

A diferencia de la media, la mediana es menos sensible a valores extremos o atípicos en los datos.

Por lo tanto, en ciertos casos, la mediana puede proporcionar una mejor representación del valor central de un conjunto de datos.

A continuación, se presentan ejemplos de cómo calcular la mediana utilizando NumPy:

2.2.1. Ejemplo 1: Edades en un grupo de personas

Suponga que desea conocer la edad central en un grupo de personas para entender qué edad es más representativa del grupo. En este caso, la mediana puede ser útil.

```
import numpy as np

ages = np.array([25, 30, 22, 45, 18, 34, 28, 55, 32, 41])
median_ages = np.median(ages)
print("Mediana de edades:", median_ages)
```

2.2.2. Ejemplo 2: Tiempos de respuesta en una encuesta

Imagine que realiza una encuesta en línea y desea analizar los tiempos de respuesta de los participantes para identificar la duración típica que toma completar la encuesta.

Los tiempos de respuesta pueden verse afectados por valores extremos, como usuarios que dejan la encuesta abierta durante horas antes de completarla. En este caso, la mediana es una medida más apropiada.

```
import numpy as np

response_times = np.array([5, 8, 10, 5, 7, 120, 12, 5, 9, 6, 4])
median_response_times = np.median(response_times)
print("Mediana de tiempos de respuesta:", median_response_times)
```

2.2.3. Ejemplo 3: Precios de casas en un vecindario

Suponga que está analizando los precios de casas en un vecindario y desea obtener una comprensión general del valor central de las propiedades. Los precios de las casas pueden variar mucho, por lo que la mediana puede proporcionar un valor más representativo.

```
import numpy as np

house_prices = np.array([250000, 300000, 400000, 150000, 350000, 450000, 1200000,
280000, 320000])
median_house_prices = np.median(house_prices)
print("Mediana de precios de casas:", median_house_prices)
```

En estos ejemplos, utilizamos la función `np.median()` para calcular la mediana de diferentes conjuntos de datos.

La mediana puede ser una medida más representativa en situaciones donde los datos contienen valores extremos o atípicos que podrían distorsionar la media.

2.3. Moda

La **moda** es otra medida de tendencia central que representa el valor o los valores que ocurren con mayor frecuencia en un conjunto de datos.

A diferencia de la media y la mediana, la moda puede ser aplicable tanto a datos numéricos como a

datos categóricos. La moda puede ser útil en situaciones donde se desea identificar el valor más común o popular en un conjunto de datos.

NumPy no tiene una función incorporada para calcular la moda directamente, pero podemos utilizar la biblioteca SciPy, que es compatible con NumPy y proporciona una función para calcular la moda.

A continuación, se presentan ejemplos de cómo calcular la moda utilizando SciPy:

2.3.1. Ejemplo 1: Calificaciones más frecuentes en un examen

Suponga que desea conocer la calificación más común obtenida por los estudiantes en un examen para entender el nivel de dificultad.

```
import numpy as np
from scipy import stats

grades = np.array([75, 80, 85, 90, 80, 75, 85, 90, 75, 80, 90, 85, 80, 80])
mode_grades = stats.mode(grades)
print("Moda de las calificaciones:", mode_grades.mode[0])
```

2.3.2. Ejemplo 2: Color de camiseta más vendido en una tienda

Imagine que administra una tienda de ropa y desea conocer el color de camiseta más vendido para comprender las preferencias de los clientes.

```
import numpy as np
from scipy import stats

shirt_colors = np.array(['blue', 'red', 'blue', 'green', 'blue', 'red', 'blue',
'green', 'blue', 'red', 'blue'])
mode_shirt_colors = stats.mode(shirt_colors)
print("Moda de los colores de camiseta:", mode_shirt_colors.mode[0])
```

2.3.3. Ejemplo 3: Tamaño de zapato más común en un grupo de personas

Suponga que está organizando un evento y desea conocer el tamaño de zapato más común entre los asistentes para facilitar la logística relacionada con el calzado.

```
import numpy as np
from scipy import stats

shoe_sizes = np.array([8, 9, 10, 8, 9, 8, 7, 9, 10, 8, 8, 9, 11, 6, 7])
mode_shoe_sizes = stats.mode(shoe_sizes)
print("Moda de los tamaños de zapato:", mode_shoe_sizes.mode[0])
```

En estos ejemplos, utilizamos la función `stats.mode()` de **SciPy** para calcular la moda de diferentes conjuntos de datos. La moda puede ser útil para identificar el valor más común en un conjunto de datos y proporcionar información sobre las tendencias y preferencias.

2.4. Desviación estándar

La **desviación estándar** es una medida de dispersión que indica cuánto varían los valores de un conjunto de datos con respecto a la media.

Una desviación estándar alta indica que los valores están más dispersos, mientras que una desviación estándar baja indica que los valores están más agrupados alrededor de la media.

La desviación estándar es útil para evaluar la consistencia, la variabilidad y la incertidumbre en un conjunto de datos.

A continuación, se presentan ejemplos de cómo calcular la desviación estándar utilizando NumPy:

2.4.1. Ejemplo 1: Variabilidad en las calificaciones de los estudiantes

Suponga que desea evaluar la consistencia en el rendimiento de los estudiantes en un examen. La desviación estándar de las calificaciones puede proporcionar información sobre la variabilidad en las calificaciones.

```
import numpy as np

grades = np.array([85, 90, 75, 80, 95, 70, 78, 88, 92, 76])
std_grades = np.std(grades)
print("Desviación estándar de las calificaciones:", std_grades)
```

2.4.2. Ejemplo 2: Variabilidad en los precios de las casas

Imagine que está analizando la variabilidad en los precios de las casas en un vecindario. La desviación estándar de los precios de las casas puede proporcionar información sobre la variabilidad en los precios.

```
import numpy as np

house_prices = np.array([250000, 300000, 400000, 150000, 350000, 450000, 1200000,
280000, 320000])
std_house_prices = np.std(house_prices)
print("Desviación estándar de los precios de las casas:", std_house_prices)
```

2.4.3. Ejemplo 3: Variabilidad en los tiempos de respuesta en una encuesta

Suponga que desea analizar la variabilidad en los tiempos de respuesta de los participantes en una encuesta. La desviación estándar de los tiempos de respuesta puede proporcionar información sobre la consistencia en los tiempos de respuesta.

```
import numpy as np

response_times = np.array([5, 8, 10, 5, 7, 120, 12, 5, 9, 6, 4])
std_response_times = np.std(response_times)
print("Desviación estándar de los tiempos de respuesta:", std_response_times)
```

En estos ejemplos, utilizamos la función `np.std()` para calcular la desviación estándar de diferentes conjuntos de datos. La desviación estándar puede ser útil para evaluar la variabilidad y la consistencia en un conjunto de datos y proporcionar información sobre la incertidumbre asociada con los valores del conjunto de datos.

2.5. Máximo y mínimo

El **máximo** y el **mínimo** son medidas que indican el valor más alto y el valor más bajo en un conjunto de datos, respectivamente.

Estas medidas pueden ser útiles para identificar rangos, extremos y tendencias en los datos.

A continuación, se presentan ejemplos de cómo encontrar el máximo y el mínimo utilizando NumPy:

2.5.1. Ejemplo 1: Máximo y mínimo de las temperaturas en una semana

Supongamos que desea conocer la temperatura más alta y más baja registradas durante una semana. Los datos de temperatura diaria (en grados Celsius) son: 15, 18, 22, 24, 20, 17 y 19.

```
import numpy as np

temperatures = np.array([15, 18, 22, 24, 20, 17, 19])
max_temp = np.max(temperatures)
min_temp = np.min(temperatures)

print("Temperatura máxima:", max_temp)
print("Temperatura mínima:", min_temp)
```

2.5.2. Ejemplo 2: Máximo y mínimo de los precios de productos en una tienda

Imagine que está analizando los precios de los productos en una tienda y desea conocer el producto más caro y el más barato. Los precios de los productos (en dólares) son: 10, 25, 40, 5, 15, 60 y 30.

```
import numpy as np

product_prices = np.array([10, 25, 40, 5, 15, 60, 30])
max_price = np.max(product_prices)
min_price = np.min(product_prices)
```

```
print("Precio máximo:", max_price)
print("Precio mínimo:", min_price)
```

2.5.3. Ejemplo 3: Máximo y mínimo de la edad de los empleados en una empresa

Supongamos que desea conocer la edad del empleado más joven y más mayor en una empresa. Las edades de los empleados son: 22, 35, 28, 45, 31, 56 y 39.

```
import numpy as np

employee_ages = np.array([22, 35, 28, 45, 31, 56, 39])
max_age = np.max(employee_ages)
min_age = np.min(employee_ages)

print("Edad máxima:", max_age)
print("Edad mínima:", min_age)
```

En estos ejemplos, utilizamos las funciones np.max() y np.min() para encontrar el máximo y el mínimo en diferentes conjuntos de datos. Estas medidas pueden ser útiles para identificar rangos, extremos y tendencias en los datos, y proporcionar información sobre la distribución y las características de los datos.

2.6. Cuartiles

Los **cuartiles** son medidas que dividen un conjunto de datos ordenado en **cuatro partes iguales**.

Hay **tres cuartiles**:

- El **primer cuartil (Q1)**, que representa el valor que separa el **25% inferior** de los datos
- El **segundo cuartil (Q2)**, que es la mediana y separa el **50% inferior** de los datos
- El **tercer cuartil (Q3)**, que representa el valor que separa el **75% inferior** de los datos.

Los **cuartiles** pueden ser útiles para entender la **distribución** y la **dispersión** de los **datos**, así como para identificar valores atípicos y tendencias.

A continuación, se presentan ejemplos de cómo calcular los cuartiles utilizando NumPy:

2.6.1. Ejemplo 1: Cuartiles de las calificaciones de los estudiantes

Suponga que desea analizar la distribución de las calificaciones de los estudiantes en un examen. Las calificaciones son: 55, 65, 70, 75, 80, 85, 90, 95, 100.

```
import numpy as np
```

```
grades = np.array([55, 65, 70, 75, 80, 85, 90, 95, 100])
Q1 = np.percentile(grades, 25)
Q2 = np.percentile(grades, 50)
Q3 = np.percentile(grades, 75)

print("Primer cuartil (Q1):", Q1)
print("Segundo cuartil (Q2 - Mediana):", Q2)
print("Tercer cuartil (Q3):", Q3)
```

2.6.2. Ejemplo 2: Cuartiles de los precios de las casas

Imagine que está analizando la distribución de los precios de las casas en un vecindario. Los precios de las casas son: 150000, 200000, 250000, 300000, 350000, 400000, 450000, 500000, 550000.

```
import numpy as np

house_prices = np.array([150000, 200000, 250000, 300000, 350000, 400000, 450000,
500000, 550000])
Q1 = np.percentile(house_prices, 25)
Q2 = np.percentile(house_prices, 50)
Q3 = np.percentile(house_prices, 75)

print("Primer cuartil (Q1):", Q1)
print("Segundo cuartil (Q2 - Mediana):", Q2)
print("Tercer cuartil (Q3):", Q3)
```

2.6.3. Ejemplo 3: Cuartiles del tiempo de espera en una cola

Supongamos que desea analizar la distribución del tiempo de espera en una cola en una tienda. Los tiempos de espera en minutos son: 5, 10, 15, 20, 25, 30, 35, 40, 45.

```
import numpy as np

waiting_times = np.array([5, 10, 15, 20, 25, 30, 35, 40, 45])
Q1 = np.percentile(waiting_times, 25)
Q2 = np.percentile(waiting_times, 50)
Q3 = np.percentile(waiting_times, 75)

print("Primer cuartil (Q1):", Q1)
print("Segundo cuartil (Q2 Mediana):", Q2)
print("Tercer cuartil (Q3):", Q3)
```

En estos ejemplos, utilizamos la función `np.percentile()` para calcular los cuartiles en diferentes conjuntos de datos. Los cuartiles pueden proporcionar información sobre la distribución y la dispersión de los datos, lo que nos ayuda a identificar tendencias y valores atípicos. También nos permiten comparar diferentes conjuntos de datos y comprender cómo se distribuyen los valores en diferentes rangos.

2.7. Percentiles

Los **percentiles** son medidas que dividen un conjunto de datos ordenado **en 100 partes iguales**, indicando el valor por debajo del cual se encuentra un porcentaje específico de los datos.

Los **percentiles** son similares a los **cuartiles**, pero son más flexibles porque pueden dividir los datos en cualquier porcentaje. En otras palabras, los **cuartiles son un caso especial de los percentiles**:

- el primer cuartil es el percentil 25
- el segundo cuartil (la mediana) es el percentil 50
- el tercer cuartil es el percentil 75

Los percentiles son útiles para describir la distribución y la dispersión de los datos, así como para identificar valores atípicos y tendencias.

A continuación, se presentan ejemplos de cómo calcular los percentiles utilizando NumPy:

2.7.1. Ejemplo 1: Percentiles de las calificaciones de los estudiantes

Suponga que desea analizar la distribución de las calificaciones de los estudiantes en un examen y encontrar los percentiles 10, 50 y 90. Las calificaciones son: 55, 65, 70, 75, 80, 85, 90, 95, 100.

```
import numpy as np

grades = np.array([55, 65, 70, 75, 80, 85, 90, 95, 100])
P10 = np.percentile(grades, 10)
P50 = np.percentile(grades, 50)
P90 = np.percentile(grades, 90)

print("Percentil 10:", P10)
print("Percentil 50 (Mediana):", P50)
print("Percentil 90:", P90)
```

2.7.2. Ejemplo 2: Percentiles de los precios de las casas

Imagine que está analizando la distribución de los precios de las casas en un vecindario y desea encontrar los percentiles 5, 50 y 95. Los precios de las casas son: 150000, 200000, 250000, 300000, 350000, 400000, 450000, 500000, 550000.

```
import numpy as np

house_prices = np.array([150000, 200000, 250000, 300000, 350000, 400000, 450000,
500000, 550000])
P5 = np.percentile(house_prices, 5)
P50 = np.percentile(house_prices, 50)
P95 = np.percentile(house_prices, 95)
```

```
print("Percentil 5:", P5)
print("Percentil 50 (Mediana):", P50)
print("Percentil 95:", P95)
```

2.7.3. Ejemplo 3: Percentiles del tiempo de espera en una cola

Supongamos que desea analizar la distribución del tiempo de espera en una cola en una tienda y desea encontrar los percentiles 20, 50 y 80. Los tiempos de espera en minutos son: 5, 10, 15, 20, 25, 30, 35, 40, 45.

```
import numpy as np

waiting_times = np.array([5, 10, 15, 20, 25, 30, 35, 40, 45])
P20 = np.percentile(waiting_times, 20)
P50 = np.percentile(waiting_times, 50)
P80 = np.percentile(waiting_times, 80)

print("Percentil 20:", P20)
print("Percentil 50 (Mediana):", P50)
print("Percentil 80:", P80)
```

En estos ejemplos, utilizamos la función `np.percentile()` para calcular los percentiles en diferentes conjuntos de datos. Los percentiles, al igual que los cuartiles, proporcionan información sobre la **distribución** y la **dispersión** de los datos, lo que nos ayuda a identificar tendencias y valores atípicos. La principal diferencia entre los **percentiles** y los **cuartiles** es que los percentiles son más flexibles y permiten dividir los datos en cualquier porcentaje, lo que nos brinda una descripción más detallada de la distribución de los datos.

2.8. Rango intercuartílico

El **rango intercuartílico (IQR)** es una **medida de dispersión** que indica la diferencia entre el **primer cuartil (Q1)** y el **tercer cuartil (Q3)** en un conjunto de datos.

El **IQR** es útil porque proporciona una medida de la dispersión de los datos en el 50% central, lo que nos ayuda a identificar la variabilidad de los datos en ese rango y a detectar **valores atípicos**.

A diferencia de la **desviación estándar** y la **varianza**, el **IQR** no se ve afectado por **valores extremos**, lo que lo convierte en una medida de dispersión **más robusta** en ciertos casos.

A continuación, se presentan ejemplos de cómo calcular el **rango intercuartílico** utilizando NumPy:

2.8.1. Ejemplo 1: Rango intercuartílico de las calificaciones de los estudiantes

Suponga que desea analizar la distribución de las calificaciones de los estudiantes en un examen y

calcular el IQR. Las calificaciones son: 55, 65, 70, 75, 80, 85, 90, 95, 100.

```
import numpy as np

grades = np.array([55, 65, 70, 75, 80, 85, 90, 95, 100])
Q1 = np.percentile(grades, 25)
Q3 = np.percentile(grades, 75)
IQR = Q3 - Q1

print("Rango intercuartílico (IQR):", IQR)
```

2.8.2. Ejemplo 2: Rango intercuartílico de los precios de las casas

Imagine que está analizando la distribución de los precios de las casas en un vecindario y desea calcular el IQR. Los precios de las casas son: 150000, 200000, 250000, 300000, 350000, 400000, 450000, 500000, 550000.

```
import numpy as np

house_prices = np.array([150000, 200000, 250000, 300000, 350000, 400000, 450000,
500000, 550000])
Q1 = np.percentile(house_prices, 25)
Q3 = np.percentile(house_prices, 75)
IQR = Q3 - Q1

print("Rango intercuartílico (IQR):", IQR)
```

2.8.3. Ejemplo 3: Rango intercuartílico del tiempo de espera en una cola

Supongamos que desea analizar la distribución del tiempo de espera en una cola en una tienda y desea calcular el IQR. Los tiempos de espera en minutos son: 5, 10, 15, 20, 25, 30, 35, 40, 45.

```
import numpy as np

waiting_times = np.array([5, 10, 15, 20, 25, 30, 35, 40, 45])
Q1 = np.percentile(waiting_times, 25)
Q3 = np.percentile(waiting_times, 75)
IQR = Q3 - Q1

print("Rango intercuartílico (IQR):", IQR)
```

En estos ejemplos, utilizamos la función **np.percentile()** para calcular el **primer** y **tercer cuartil** y, a continuación, calculamos el **rango intercuartílico** restando **Q1** de **Q3**.

El **IQR** nos proporciona información sobre la **dispersión** de los datos en el 50% central, lo que nos permite comprender la **variabilidad de los datos** en ese rango.

Además, el **IQR** es una **medida de dispersión robusta** que no se ve afectada por valores extremos, lo que lo hace útil en situaciones donde los valores atípicos pueden distorsionar otras medidas de dispersión, como la desviación estándar y la varianza.

2.9. Correlación

La **correlación** es una medida que indica la fuerza y dirección de la relación lineal entre dos variables.

Los **coeficientes de correlación** varían entre `-1` y `1`, donde:

- `-1` indica una correlación negativa perfecta
- `1` indica una correlación positiva perfecta
- `0` indica que no hay correlación lineal

La **correlación** es útil para entender si **dos variables están relacionadas** y cómo están relacionadas, lo que puede ayudar en la toma de decisiones, el análisis de datos y la predicción.

A continuación, se presentan ejemplos de cómo calcular la correlación utilizando NumPy:

2.9.1. Ejemplo 1: Correlación entre la edad y el salario

Suponga que desea analizar la relación entre la edad de las personas y sus salarios. Aquí hay dos listas de edades y salarios de un grupo de personas:

```
import numpy as np

ages = np.array([25, 30, 35, 40, 45, 50, 55])
salaries = np.array([50000, 60000, 70000, 80000, 90000, 100000, 110000])

correlation = np.corrcoef(ages, salaries)[0, 1]
print("Correlación entre la edad y el salario:", correlation)
```

2.9.2. Ejemplo 2: Correlación entre la cantidad de ejercicio y el índice de masa corporal (IMC)

Suponga que desea analizar la relación entre la cantidad de ejercicio que hacen las personas (en horas por semana) y sus índices de masa corporal (IMC):

```
import numpy as np

exercise_hours = np.array([1, 2, 3, 4, 5, 6, 7])
bmi = np.array([28, 26, 24, 22, 20, 18, 16])

correlation = np.corrcoef(exercise_hours, bmi)[0, 1]
print("Correlación entre la cantidad de ejercicio y el IMC:", correlation)
```

2.9.3. Ejemplo 3: Correlación entre las ventas de helados y la temperatura

Suponga que desea analizar la relación entre las ventas de helados (en unidades) y la temperatura (en grados Celsius):

```
import numpy as np

ice_cream_sales = np.array([100, 150, 200, 250, 300, 350, 400])
temperatures = np.array([10, 15, 20, 25, 30, 35, 40])

correlation = np.corrcoef(ice_cream_sales, temperatures)[0, 1]
print("Correlación entre las ventas de helados y la temperatura:", correlation)
```

En estos ejemplos, utilizamos la función `np.corrcoef()` para calcular el coeficiente de correlación entre dos conjuntos de datos.

La correlación nos ayuda a entender si dos variables están relacionadas y cómo están relacionadas, lo que es útil para tomar decisiones informadas y realizar análisis en diferentes contextos de la vida real.

3. Ejemplo estadísticas

Supongamos que tienes los siguientes datos de **ventas mensuales en dólares** para una tienda:

```
ventas_mensuales = np.array([1200, 1500, 1100, 1300, 1400, 1600, 1350, 1550, 1100,
1300, 1400, 1650])
```

Calcular la **venta total** del año:

```
venta_total_anual = np.sum(ventas_mensuales)
print("Venta total anual:", venta_total_anual)
```

Calcular el **promedio** o **media** de ventas mensuales:

```
promedio_ventas_mensuales = np.mean(ventas_mensuales)
print("Promedio de ventas mensuales:", promedio_ventas_mensuales)
```

Encontrar el mes con las **ventas más altas** y más **bajas**:

```
mes_ventas_max = np.argmax(ventas_mensuales) + 1
mes_ventas_min = np.argmin(ventas_mensuales) + 1
print("Mes con las ventas más altas:", mes_ventas_max)
print("Mes con las ventas más bajas:", mes_ventas_min)
```

Calcular la **varianza** y la **desviación estándar** de las ventas mensuales:

```
varianza_ventas = np.var(ventas_mensuales)
desviacion_estandar_ventas = np.std(ventas_mensuales)
print("Varianza de las ventas mensuales:", varianza_ventas)
print("Desviación estándar de las ventas mensuales:", desviacion_estandar_ventas)
```

Calcular los **percentiles** 25, 50 (mediana) y 75 de las ventas mensuales:

```
percentil_25 = np.percentile(ventas_mensuales, 25)
percentil_50 = np.percentile(ventas_mensuales, 50)
percentil_75 = np.percentile(ventas_mensuales, 75)
print("Percentil 25:", percentil_25)
print("Percentil 50 (mediana):", percentil_50)
print("Percentil 75:", percentil_75)
```

Supongamos que también tienes datos de costo mensual:

```
costos_mensuales = np.array([900, 1000, 800, 950, 1100, 1300, 1000, 1200, 800, 1000,
1200, 1300])
```

Calcular la **correlación** entre las **ventas** mensuales y los **costos** mensuales:

```
correlacion_ventas_costos = np.corrcoef(ventas_mensuales, costos_mensuales)[0, 1]
print("Correlación entre ventas y costos mensuales:", correlacion_ventas_costos)
```