

Comandos principales de NumPy y Pandas

Estudiante

Juan Diego Ortiz Baquero

Universidad de Cundinamarca Extensión Chía
Programa de Ingeniería de Sistemas y Computación
Machine Learning

Docente

Edison Gustavo Canon Varela

NumPy – Creación y Manipulación de Arreglos

Introducción

NumPy es la biblioteca principal para trabajar con arrays en Python. Usar arrays (ndarrays) permite procesar datos de forma eficiente y realizar cálculos matemáticos rápidamente, algo fundamental en ciencia de datos y análisis numérico.

A continuación explico cómo crear arrays, cambiarlos de forma (reshape), concatenarlos y realizar operaciones básicas, con ejemplos probados en Google Colab. En cada sección agrego una breve idea de para qué sirve cada operación.

1. Creación de arrays

¿Para qué sirve?

Los arrays son la base para procesar datos en NumPy. Puedes crearlos a partir de listas, tuplas o usando funciones de la propia librería para generar datos automáticamente.

Desde listas o tuplas

Convierte listas y tuplas de Python en arrays NumPy. Muy útil para comenzar a trabajar con tus propios datos.

```
import numpy as np

# Array de una dimensión (1D)
a1 = np.array([1, 2, 3, 4])
print(a1)
# Resultado: [1 2 3 4]

# Array de dos dimensiones (2D)
a2 = np.array([[1, 2], [3, 4]])
print(a2)
# Resultado:
# [[1 2]
#  [3 4]]
```

Con funciones de NumPy

NumPy ofrece varias funciones para crear arrays rápidamente, ideales para pruebas, inicialización y generación de datos sintéticos.

```
# arange: secuencia de números
arr = np.arange(0, 6)
print(arr)
# Resultado: [0 1 2 3 4 5]
```

```
# zeros y ones
zeros = np.zeros((2, 3))
ones = np.ones((2, 3))
print(zeros)
print(ones)
# Resultado:
# [[0. 0. 0.]
#  [0. 0. 0.]]
# [[1. 1. 1.]
#  [1. 1. 1.]]
```

2. Cambiar la forma de un array (reshape)

¿Para qué sirve?

Muchas veces necesitas reorganizar tus datos para adaptarlos a modelos, gráficos o análisis. Con **reshape** puedes cambiar la estructura del array sin perder la información.

```
arr = np.arange(6)
reshaped = arr.reshape((2, 3))
print(reshaped)
# Resultado:
# [[0 1 2]
#  [3 4 5]]
```

3. Concatenar arrays

¿Para qué sirve?

Unir arrays te permite agrupar información, combinar resultados o construir estructuras más grandes a partir de datos pequeños.

Unir arrays puede hacerse horizontal o verticalmente.

```
a = np.array([[1, 2], [3, 4]])
b = np.array([[5, 6]])

# Vertical (añadir filas)
v_concat = np.vstack((a, b))
print(v_concat)
# Resultado:
# [[1 2]
#  [3 4]
#  [5 6]]

# Horizontal (añadir columnas)
c = np.array([[7], [8], [9]])
h_concat = np.hstack((v_concat, c))

print(h_concat)
# Resultado:
# [[1 2 7]
#  [3 4 8]
#  [5 6 9]]
```

Para arrays 1D, se puede usar **concatenate**:

```
x = np.array([1, 2, 3])
y = np.array([4, 5, 6])
xy = np.concatenate([x, y])
print(xy)
# Resultado: [1 2 3 4 5 6]
```

4. Operaciones básicas con arrays

¿Para qué sirve?

Las operaciones matemáticas sobre arrays son esenciales para análisis, procesamiento de señales, estadística y cualquier cálculo sobre datos. NumPy permite operar sobre todos los elementos sin necesidad de escribir bucles.

Las operaciones matemáticas se aplican a todos los elementos. No es necesario recorrer el array con for.

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

# Suma y resta
print(arr + 1)      # [3 5 7 9]
print(arr - 1)      # [1 3 5 7]

# Multiplicación y división
print(arr * 2)       # [ 4  8 12 16]
print(arr / 2)       # [1. 2. 3. 4.]

# Potencias y raíz
print(arr ** 2)      # [ 4 16 36 64]
print(np.sqrt(arr))  # [1.41421356 2.         2.44948974 2.82842712]
```

Operaciones entre arrays:

```
a = np.array([1, 2, 3])
b = np.array([10, 20, 30])
print(a + b)      # [11 22 33]
print(a * b)      # [10 40 90]
```

NumPy – Operaciones estadísticas y funciones avanzadas

Introducción

NumPy no solo sirve para crear arrays y hacer operaciones básicas, también incluye muchas funciones estadísticas, de generación de datos y de álgebra lineal. Estas funciones son esenciales para análisis de datos y cálculos científicos, ya que permiten obtener información resumida, generar datos sintéticos y resolver problemas matemáticos complejos de forma eficiente.

1. Operaciones estadísticas básicas

Promedio (**mean**), desviación estándar (**std**), suma (**sum**)

Estas funciones permiten obtener estadísticas rápidas sobre los datos contenidos en un array.

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])

print(np.mean(arr))  # Promedio: 3.0
print(np.std(arr))   # Desviación estándar: 1.41421...
print(np.sum(arr))   # Suma: 15
```

Para arrays multidimensionales, se puede especificar el eje (**axis**) sobre el cual calcular:

```
mat = np.array([[1, 2], [3, 4]])

print(np.mean(mat, axis=0)) # Promedio por columna: [2. 3.]
print(np.mean(mat, axis=1)) # Promedio por fila: [1.5 3.5]
```

2. Generación de datos: **arange**, **linspace**, **random**

arange y **linspace**

Estas funciones crean arrays numéricos con distintos criterios:

```
# arange: números enteros (o decimales) en un rango
a = np.arange(0, 10, 2)
print(a) # [0 2 4 6 8]

# linspace: números equiespaciados entre dos valores
b = np.linspace(0, 1, 5)
print(b) # [0.  0.25 0.5  0.75 1.  ]
```

Datos aleatorios con **random**

NumPy incluye funciones para generar números aleatorios, útiles para simulaciones y pruebas.

```
rng = np.random.default_rng(42)

# Array aleatorio de 3 números entre 0 y 1
rand_arr = rng.random(3)
print(rand_arr) # [0.77395605 0.43887844 0.85859792]

# Array aleatorio entero entre 0 y 9, forma (2, 2)
rand_int = rng.integers(0, 10, size=(2, 2))
print(rand_int)
# Resultado:
# [[8 0]
#  [2 6]]
```

3. Álgebra lineal

NumPy tiene funciones avanzadas para trabajar con matrices y resolver problemas de álgebra lineal.

Producto de matrices

```
A = np.array([[1, 2], [3, 4]])
B = np.array([[2, 0], [1, 2]])

# Producto matricial
prod = np.dot(A, B)
print(prod)
# Resultado:
# [[4 4]
#  [10 8]]
```

Desde Python 3.5 también se puede usar el operador `@` para multiplicación de matrices:

```
print(A @ B)
# Mismo resultado que np.dot
```

Transpuesta y determinante

```
print(A.T)          # Transpuesta
print(np.linalg.det(A)) # Determinante: -2.0
```

Inversa de una matriz

```
invA = np.linalg.inv(A)
print(invA)
# Resultado:
# [[-2.  1. ]
#  [ 1.5 -0.5]]
```

Solución de sistemas lineales

```
# Ax = b
b = np.array([5, 6])
x = np.linalg.solve(A, b)
print(x)
# Resultado: [ -4.  4.5 ]
```

Pandas - Creación y manipulación de DataFrames y Series

Introducción

Pandas es la biblioteca más utilizada para manejo y análisis de datos en Python. Sus dos estructuras principales son **DataFrames** (tabla de datos) y **Series** archivos, seleccionar columnas y filas, y trabajar fácilmente con diferentes tipos de datos.

1. Crear DataFrames y Series

¿Para qué sirve? Los DataFrames son como hojas de cálculo o tablas, y las Series son como una columna individual con etiquetas.

Crear un DataFrame desde un diccionario o lista

```
import pandas as pd

# Desde diccionario
df = pd.DataFrame({'A': [1, 2, 3], 'B': ['a', 'b', 'c']})
print(df)
# Resultado:
#   A  B
# 0  1  a
# 1  2  b
# 2  3  c

# Desde lista de listas
df2 = pd.DataFrame([[10, 20], [30, 40]], columns=['X', 'Y'])
print(df2)
# Resultado:
#   X  Y
# 0 10 20
# 1 30 40
```

Crear una Series

```
s = pd.Series([4, 7, 9], index=['a', 'b', 'c'])
print(s)
# Resultado:
# a    4
# b    7
# c    9
# dtype: int64
```

2. Cargar datos desde archivo

¿Para qué sirve? Pandas puede leer datos directamente de archivos CSV, Excel, SQL, etc.


```
# Leer CSV
df = pd.read_csv('archivo.csv')
print(df.head()) #Ver las primeras filas

# Leer Excel
df_excel = pd.read_excel('archivo.xlsx')
print(df_excel.head())
```

3. Selección de columnas y filas

¿Para qué sirve? Permite acceder y manipular partes específicas del DataFrame para análisis o transformación

Seleccionar columnas

```
# Una Columna
print(df['A'])

# Varias Columnas
print(df[['A', 'B']])
```

Seleccionar filas (por índice, posición o condición)

```
# Primera fila por posición
print(df.iloc[0])
# Resultado
# A      1
# B      a
# Name: 0, dtype: object

# Fila por etiqueta (Si hay índices personalizados)
print(df.iloc[0])
# Resultado
# A      1
# B      a
# Name: 0, dtype: object

# Filas por condición
print(df[df['A'] > 1])
# Resultado
#   A B
# 1  2 b
# 2  3 c
```

4. Tipos de datos y conversión

¿Para qué sirve? Es importante conocer y convertir los tipos de datos para evitar errores y para aplicar funciones correctamente.

```
print(df.dtypes) # Ver tipo de cada columna
# Resultado
# A      int64
# B      object
# dtype: object

# Convertir tipo de columna
df['A'] = df['A'].astype(float)
print(df.dtypes)
# Resultado
# A      float64
# B      object
# dtype: object
```

Pandas - Operaciones avanzada en DataFrames

Introducción

Además de las funciones básicas, Pandas permite realizar operaciones avanzadas sobre DataFrames para analizar y transformar datos de manera eficiente. Se muestran ejemplos comunes y útiles para filtrar datos, agrupar y resumir información, combinar tablas, manejar valores nulos y exportar o importar datos.

1. Filtros

¿Para qué sirve? Seleccionar filas según condiciones, por ejemplo, extraer datos relevantes de una tabla grande.

```
import pandas as pd

df = pd.DataFrame({
    'A': [1, 2, 3, 4, 5],
    'B': ['x', 'y', 'x', 'y', 'x']
})

# Filas donde A es mayor que 2
print(df[df['A'] > 2])
# Resultado
#   A B
# 2 3 x
# 3 4 y
# 4 5 x

# Filas donde B es 'x'
print(df[df['B'] == 'x'])
# Resultado
#   A B
# 0 1 x
# 2 3 x
# 4 5 x
```

2. Agrupación y resumen (**groupby**)

¿Para qué sirve? Agrupar datos por categorías y calcular sumas, promedios u otras estadísticas dentro de cada grupo.

```
# Agrupar por columna B y calcular promedio de A
grouped = df.groupby('B')['A'].mean()
print(grouped)
# Resultado:
# B
# x    3.0
# y    3.0
# Name: A, dtype: float64
```

También se pueden usar otras funciones como `sum()`, `count`, `max`, etc.

3. Unir DataFrames (**merge** / **join**)

¿Para qué sirve? Combinar información de dos tablas según una columna común (como clave).

```
df1 = pd.DataFrame({'key': ['a', 'b', 'c'], 'value1': [1, 2, 3]})
df2 = pd.DataFrame({'key': ['a', 'b', 'c'], 'value2': [4, 5, 6]})

# Merge por columna 'key'
merged = pd.merge(df1, df2, on='key', how='outer')
print(merged)
# Resultado
#   key  value1  value2
# 0  a      1.0     4.0
# 1  b      2.0     5.0
# 2  c      3.0     NaN
# 3  d      NaN     6.0
```

4. Manejo de valores nulos

¿Para qué sirve? Identificar, eliminar o reemplazar datos faltantes para evitar errores en el análisis.

```
# Detectar valores nulos
print(df.isnull())

# Eliminar filas con valores nulos
print(df.dropna())

# Reemplazar nulos con un valor específico
print(df.fillna(0))
```

5. Exportación e importación de datos

¿Para qué sirve? Guardar resultados en archivos o cargar datos externos para análisis.

```
# Exportar a CSV
df.to_csv('mi_dataframe.csv', index=False)

# Importar desde CSV
nuevo_df = pd.read_csv('mi_dataframe.csv')
print(nuevo_df)
```

Pandas también soporta otros formatos como Excel (.xlsx), JSON, HTML, SQL, entre otros.

Fuentes

- Array creation — NumPy v2.3 manual. (s/f). Numpy.org. Recuperado el 26 de agosto de 2025, de <https://numpy.org/doc/stable/user/basics.creation.html>
- Google colab. (s/f). Google.com. Recuperado el 26 de agosto de 2025, de

<https://colab.research.google.com/drive/1y9wOg9nnFITXrY3lGtamlAhoPrmmGye0?authuser=1>

[Ejecuciones propias en Google Colab]

- Sherrill, D. [@CodeWithDerrick]. (s/f). Introduction to NumPy arrays for beginners - learn NumPy series [[Object Object]]. Youtube. Recuperado el 26 de agosto de 2025, de <https://www.youtube.com/watch?v=9fcTq8PDWWA>
- Google colab. (s/f). Google.com. Recuperado el 26 de agosto de 2025, de <https://colab.research.google.com/drive/1N8QNpHXhRpXT9cR7vjnGT1Z9xvYeEERU?usp=sharing>
- Linear algebra — NumPy v2.3 manual. (s/f). Numpy.org. Recuperado el 26 de agosto de 2025, de <https://numpy.org/doc/stable/reference/routines.linalg.html>
- Random sampling — NumPy v2.3 manual. (s/f). Numpy.org. Recuperado el 26 de agosto de 2025, de <https://numpy.org/doc/stable/reference/random/index.html>
- Statistics — NumPy v2.3 manual. (s/f). Numpy.org. Recuperado el 26 de agosto de 2025, de <https://numpy.org/doc/stable/reference/routines.statistics.html>
- Pandas documentation — pandas 2.3.2 documentation. (s/f). Pydata.org. Recuperado el 26 de agosto de 2025, de <https://pandas.pydata.org/docs>
- Google colab. (s/f). Google.com. Recuperado el 26 de agosto de 2025, de https://colab.research.google.com/drive/139UHZ81kjan6cPhKPKgZy_Rh65u0_ry?usp=sharing [Ejercicios]
- Pandas documentation — pandas 2.3.2 documentation. (s/f). Pydata.org. Recuperado el 26 de agosto de 2025, de <https://pandas.pydata.org/docs/>
- [Ejecuciones propias en Google Colab] Google colab. (s/f). Google.com. Recuperado el 26 de agosto de 2025, de <https://colab.research.google.com/drive/1KWDloVh-9NaxnYSaT6gerGY-0bkwldqs?usp=sharing>

LINK REPOSITORIO:

- <https://github.com/Juan-Diego-Ortiz-Baquero/machine-learning-conceptos/tree/IA-ML-Investigacion>

LINKS GOOGLE COLAB:

- A-numpy-arreglos. (2025). *Google Colaboratory notebook*. Google.
<https://colab.research.google.com/drive/1y9wOg9nnFITXrY3lGtamlAhoPrmmGye0?usp=sharing>
- B-numpy-estadística. (2025). *Google Colaboratory notebook*. Google.
<https://colab.research.google.com/drive/1N8QNpHXhRpXT9cR7vjnGT1Z9xvYeEERU?usp=sharing>
- C-pandas-dataframes. (2025). *Google Colaboratory notebook*. Google.
https://colab.research.google.com/drive/139UHZ81kjan6cPhKPKgZy_Rh65u0_ry?usp=sharing
- D-pandas-avanzado. (2025). *Google Colaboratory notebook*. Google.
<https://colab.research.google.com/drive/1KWDloVh-9NaxnYSaT6gerGY-0bkwldqs?usp=sharing>