Clase 021 — NumPy: aleatoriedad y semillas

Parte: 0 — Prerrequisitos · Fuente: Numerical Recipes cap. 7 (Random Numbers) · NumPy random.Generator docs. · Duración estimada: 60 min.

Clase 021 — NumPy: aleatoriedad y semillas

> *Parte: 0 — Prerrequisitos · Fuente: Numerical Recipes cap. 7 (Random Numbers) · NumPy random.Generator docs. · Duración estimada: 60 min.*

Objetivo

Que el alumno genere números aleatorios reproduciblemente con el API moderno (np.random.default_rng(seed)), use las distribuciones más comunes (uniforme, normal, Bernoulli, Poisson, exponencial), y entienda por qué la reproducibilidad es no-negociable en ciencia de datos.

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Crear un Generator con np.random.default_rng(seed) y usarlo para reproducibilidad.
2. Generar muestras de uniforme, normal, integers, binomial, Poisson, exponencial.
3. Permutar y muestrear sin/con reemplazo con permutation y choice.
4. Reproducir un experimento exactamente con el mismo seed.
5. Saber por qué np.random.seed() (API legacy) es deprecated en favor de Generator.

Temas

#	Tema	Por qué importa
1	np.random.default_rng(seed)	El API moderno (Generator-based, PCG64).
2	Distribuciones continuas: uniform, normal,	Las más usadas en simulación.
3	Distribuciones discretas: integers, binomi	Conteos y procesos.
4	permutation y choice	Mezclar y muestrear.
5	Reproducibilidad: por qué importa	Misma cosa con mismo seed.
6	Múltiples generadores independientes	Evita interferencia entre experimentos.

Definiciones y características

Generador pseudoaleatorio (PRNG)

: Algoritmo determinístico que produce secuencia que parece aleatoria. Con la misma semilla (seed) produce la misma secuencia → reproducible. NumPy 2026 usa PCG64 por default.

Seed (semilla)

: Estado inicial del PRNG. Mismo seed → misma secuencia, siempre, en cualquier máquina. Es la base de la reproducibilidad científica.

np.random.default_rng(seed)

: API moderno (NumPy 1.17+). Devuelve Generator independiente — no toca estado global, múltiples generadores no se interfieren. Reemplaza a np.random.seed() + funciones globales (deprecated).

Distribuciones continuas comunes

: uniform(lo, hi), normal(μ, σ), exponential(scale), gamma(shape, scale), beta(a, b). Cada una con parámetros que controlan ubicación y dispersión.

Distribuciones discretas

: integers(lo, hi) (uniforme discreto), binomial(n, p) (éxitos en n trials), poisson(λ) (eventos en intervalo).

Bootstrap

: Técnica: resampleas con reemplazo del sample original B veces, recalculas el estadístico → obtienes distribución empírica del estimador. Sin asumir CLT ni normalidad.

Dataset / recursos

Sintético: simulación Monte Carlo. Sin descarga.

Ejercicios

1. Reproducibilidad. Crea 2 rngs con seed=42, genera 1000 normales con cada uno. Verifica que son idénticos.

2. Distribuciones. Genera 10000 muestras de: uniforme [0,1], normal(5,2), exponential(λ=1/3), poisson(λ=4). Calcula media y std empírica y compara con teórica.

3. Monte Carlo de π. Estima π lanzando puntos en un cuadrado 2×2 y contando cuántos caen dentro del círculo unitario. Compara con π real.

4. Bootstrap. Dado un sample de 30 valores, estima la distribución de la media por bootstrap (1000 resamples con reemplazo).

5. Permutación. Mezcla un array de 100 elementos con permutation. Verifica que es la misma cuando usas el mismo seed.

Homework verificable

Notebook con: (a) Monte Carlo de π con N=10k, 100k, 1M reportando error; (b) bootstrap de la media de un sample (95% CI vs CLT); (c) demo de reproducibilidad con dos rngs; (d) tabla comparando momento empírico vs teórico para 4 distribuciones.

Criterio de aceptación: MC converge a π. Bootstrap CI similar al CLT. Reproducibilidad exacta.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Pongo seed pero el resultado cambia entre	Probablemente otra librería (sklearn, pyto
np.random.seed(42) no funciona como antes	Está deprecated en favor de default_rng. A
rng.choice(arr, size=N, replace=False) fal	Pediste más muestras únicas que elementos
Genero millones de números 'aleatorios' y	Estás materializando lista en RAM. Fix: si
Bootstrap CI parece muy estrecho	Pocos resamples (B). Fix: usa B ≥ 1000 par

Preguntas frecuentes

¿Por qué un generador independiente y no el global?

(1) No interfiere con libs que también usan random global. (2) Múltiples experimentos simultáneos sin conflicto. (3) API más limpia. (4) Algoritmo más rápido (PCG64). El legacy es solo por compatibilidad.

¿Mismo seed da mismos números en Linux/Windows/Mac?

Sí — PCG64 es determinístico cross-platform. La única diferencia podría venir de orden de operaciones float (paralelismo), pero default_rng es single-threaded.

¿Qué seed elegir?

Cualquier entero. 42 es convención (Hitchhiker's Guide). 0 funciona pero genera secuencias 'menos aleatorias' en los primeros bits con algunos PRNG (no PCG64). Lo importante es registrarlo para reproducir.

¿Bootstrap mejor que t-test?

Diferente caso. t-test: asume normalidad, da p-valor analítico. Bootstrap: sin asunción, da distribución empírica de cualquier estadístico (media, mediana, ratio). Para datos no-normales o estadísticos complejos, bootstrap gana.

¿Cuántas muestras necesito para Monte Carlo?

El error decrece como 1/√N. Para 1 decimal de precisión: N ≈ 100. Para 2 decimales: N ≈ 10k. Para 3: N ≈ 1M. Verifica convergencia haciendo N=100, 1k, 10k, 100k y mirando que el resultado se estabilice.

Referencias

- NumPy random.Generator
- NEP 19 — Random number generator policy
- Press et al., Numerical Recipes 3e — cap. 7 Random Numbers.

Siguiente clase

Clase 022 — Pandas: Series y DataFrame

Apéndice: notebook (primer bloque)

Primera celda ejecutable del notebook de la clase.

```
import numpy as np
import math
rng = np.random.default_rng(seed=42)
```

Archivos complementarios

- notebook.ipynb