12. NumPy Random Module

1. Topic Overview

What this topic is

The NumPy Random module is used to generate random numbers and random samples.

Why it exists

Real data is not always available.
Random data is needed for testing, simulation, and ML experiments.

One real-world analogy

Like rolling dice many times to test game rules before real players join.

2. Core Theory (Deep but Clear)

How NumPy thinks about randomness

- Random values are **pseudo-random**
- Generated using algorithms, not true randomness
- Same seed → same results

Internally:

- Output is a NumPy array
- Array has:
  - shape
  - dtype (usually float64 or int64)
  - continuous memory block

Sub-topics we will cover

1. Random number generators
2. Seeding
3. Uniform distributions
4. Normal (Gaussian) distributions
5. Random integers
6. Random sampling
7. Shuffling and permutation

3. Syntax & Examples

3.1 Random Number Generator

Basic syntax

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

Example 1

```
rng.random()
```

Output

```
0.3745401188473625
```

Explanation

- Single float
- Range: 0 to 1
- dtype: float64

Example 2

```
rng.random((2, 3))
```

Output

```
[[0.77 0.43 0.86]
 [0.12 0.65 0.34]]
```

Explanation

- Shape: (2, 3)
- Each value is independent
- Stored in contiguous memory

3.2 Seeding (Reproducibility)

Basic syntax

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

Example

```
rng.random(3)
```

Output

```
[0.77395605 0.43887844 0.85859792]
```

Explanation

- Same seed → same output
- Critical for experiments and debugging

3.3 Uniform Distribution

Basic syntax

```
rng.uniform(low, high, size)
```

Example

```
rng.uniform(10, 20, 5)
```

Output

```
[13.4 18.9 11.2 15.6 19.1]
```

Explanation

- Values between 10 and 20
- All values equally likely

3.4 Normal Distribution

Basic syntax

```
rng.normal(mean, std, size)
```

Example

```
rng.normal(0, 1, 5)
```

Output

```
[-0.23 1.45 0.67 -1.12 0.09]
```

Explanation

- Mean = 0

- Standard deviation = 1
- Used heavily in ML

3.5 Random Integers

Basic syntax

```
rng.integers(low, high, size)
```

Example

```
rng.integers(1, 7, 10)
```

Output

```
[3 6 2 1 5 4 6 2 3 1]
```

Explanation

- Integers only
- High value is excluded
- dtype: int64

3.6 Random Sampling

Basic syntax

```
rng.choice(array, size, replace=True)
```

Example

```
data = np.array([10, 20, 30, 40])
rng.choice(data, 3)
```

Output

```
[20 40 10]
```

Explanation

- Samples from existing data
- Used in bootstrapping

3.7 Shuffle and Permutation

Shuffle (in-place)

```
rng.shuffle(data)
```

Example

```
data = np.array([1, 2, 3, 4])
rng.shuffle(data)
print(data)
```

Output

```
[3 1 4 2]
```

Explanation

- Modifies original array
- No new memory created

Permutation (new array)

```
rng.permutation(data)
```

4. Why This Matters in Data Science

Data cleaning

- Generate test data
- Simulate missing values

Feature engineering

- Add noise
- Random sampling for balancing

Model input preparation

- Train-test split logic
- Random initialization

ML / DL pipelines

- Weight initialization
- Data augmentation
- Cross-validation

What breaks if you don’t understand this

- Non-reproducible results
- Wrong distributions
- Biased models
- Debugging becomes impossible

5. Common Mistakes (VERY IMPORTANT)

1. Not setting seed
    - Causes different results every run
2. Using old `np.random` API blindly
    - Harder to control state
3. Confusing uniform and normal

    - Leads to wrong assumptions
4. Modifying data unintentionally with shuffle
    - Data leakage risk
5. Wrong shape passed to random functions
    - Model input errors

6. Performance & Best Practices

When it is fast

- Large vectorized generation
- Single RNG instance

When it is slow

- Loop-based random calls
- Creating RNG repeatedly

Memory warnings

- Large arrays consume RAM fast
- Always check shape

Best practices

- Use `default_rng`
- Set seed once
- Generate in batches

7. 20 Practice Problems

Easy (5)

1. Generate 10 random floats between 0 and 1
2. Generate a 3×3 random integer matrix

3. Set a seed and verify same output
4. Generate 5 numbers from normal distribution
5. Shuffle a small array

Medium (7)

6. Simulate daily temperatures for 365 days
7. Add Gaussian noise to a feature column
8. Randomly sample 80% data for training
9. Generate random labels for testing
10. Create synthetic feature matrix (1000×20)
11. Compare uniform vs normal visually
12. Randomly permute dataset indices

Hard (5)

13. Bootstrap sampling for mean estimation
14. Simulate class imbalance data
15. Random weight initialization matrix
16. Monte Carlo estimation of probability
17. Random dropout mask generation

Industry-Level Tasks (3)

18. Create reproducible ML experiment pipeline
19. Simulate sensor data for anomaly detection
20. Generate synthetic dataset for model demo

8. Mini Checklist

- Random is pseudo-random
- Always control seed

- Know distribution types
- Understand shape and dtype
- Avoid in-place shuffle errors
- Use `default_rng`
- Vectorize everything

NumPy Random Module (Advanced but Mandatory Parts)

1. Topic Overview

What this topic is

These are advanced random tools for **probability-based sampling and simulations**.

Why it exists

Real-world data does not follow simple uniform or normal rules.

One real-world analogy

Like choosing people based on probability, not randomly picking names.

2. Core Theory (Deep but Clear)

NumPy supports **probability distributions** beyond normal and uniform.

Internally:

- Output is still a NumPy array
- Shape defines sample count
- dtype usually float64

- Probability rules control value frequency

These are **statistical generators**, not just random numbers.

New sub-topics (no duplicates)

1. Binomial distribution
2. Poisson distribution
3. Exponential distribution
4. Multivariate normal
5. Random boolean masks
6. Random choice with probabilities

3. Syntax & Examples

3.1 Binomial Distribution

Used for success/failure experiments.

Basic syntax

```
rng.binomial(n, p, size)
```

- n = number of trials
- p = probability of success

Example

```
rng.binomial(10, 0.5, 5)
```

Output

```
[6 4 5 7 3]
```

Explanation

- Each value is count of successes
- Shape = (5,)
- Used in classification simulation

3.2 Poisson Distribution

Used for event counts per interval.

Basic syntax

```
rng.poisson(lam, size)
```

Example

```
rng.poisson(3, 6)
```

Output

```
[2 4 3 1 5 2]
```

Explanation

- lam = average events
- Used in traffic, call centers, failures

3.3 Exponential Distribution

Used for time between events.

Basic syntax

```
rng.exponential(scale, size)
```

Example

```
rng.exponential(2, 5)
```

Output

```
[0.83 1.12 3.45 0.29 2.67]
```

Explanation

- Models waiting time
- Common in reliability analysis

3.4 Multivariate Normal Distribution

Used for correlated features.

Basic syntax

```
rng.multivariate_normal(mean, cov, size)
```

Example

```
mean = [0, 0]
cov = [[1, 0.8], [0.8, 1]]

rng.multivariate_normal(mean, cov, 3)
```

Output

```
[[ 0.3  0.4]
 [-1.1 -0.9]
 [ 0.8  1.0]]
```

Explanation

- Shape: (samples, features)

- Correlation is preserved
- Very important for ML testing

3.5 Random Boolean Masks

Used for filtering and dropout.

Basic syntax

```
rng.random(size) < threshold
```

Example

```
mask = rng.random(10) < 0.7
print(mask)
```

Output

```
[ True False  True  True False  True  True False  True  True]
```

Explanation

- Boolean array
- Used in data filtering and dropout

3.6 Random Choice with Probabilities

Weighted sampling.

Basic syntax

```
rng.choice(values, size, p=probabilities)
```

Example

```
values = [0, 1]
probs = [0.9, 0.1]

rng.choice(values, 10, p=probs)
```

Output

```
[0 0 1 0 0 0 0 1 0 0]
```

Explanation

- Probabilities must sum to 1
- Used in imbalanced data simulation

4. Why This Matters in Data Science

Data cleaning

- Simulate missing events
- Noise modeling

Feature engineering

- Correlated synthetic features
- Event frequency modeling

Model input preparation

- Class imbalance simulation
- Realistic dataset creation

ML / DL pipelines

- Monte Carlo simulation
- Bayesian modeling

- Stress testing models

What breaks if you don’t learn this

- Unrealistic fake data
- Poor validation
- Weak model testing
- Wrong assumptions

5. Common Mistakes (VERY IMPORTANT)

1. Using wrong distribution for problem
2. Ignoring probability sum rule
3. Wrong covariance matrix shape
4. Assuming independence when correlated
5. Using random without understanding stats

6. Performance & Best Practices

Fast when

- Vectorized sampling
- Single RNG instance

Slow when

- Sampling inside Python loops
- Large covariance matrices

Memory warnings

- Multivariate outputs grow fast
- Always check shape

Best practices

- Match distribution to problem
- Validate mean and variance
- Visualize sampled data

7. 20 Practice Problems

Easy (5)

1. Simulate coin tosses using binomial
2. Generate daily call counts using Poisson
3. Create exponential waiting times
4. Generate boolean mask with 60% True
5. Sample values with given probabilities

Medium (7)

6. Simulate website click behavior
7. Create imbalanced binary labels
8. Generate correlated feature matrix
9. Simulate machine failure intervals
10. Compare Poisson vs binomial
11. Apply random mask to dataset
12. Generate synthetic classification data

Hard (5)

13. Monte Carlo probability estimation
14. Correlated noise injection
15. Fraud transaction simulation
16. Synthetic sensor stream generation
17. Bootstrapped sampling with weights

Industry-Level Tasks (3)

18. Load testing ML pipelines
19. Simulate real-world imbalance scenarios
20. Stress test model robustness

8. Mini Checklist

- Choose correct distribution
- Understand probability meaning
- Control correlation
- Validate randomness statistically
- Use weighted sampling when needed

NumPy Random Legacy Functions

Functions Covered

- `np.random.rand`
- `np.random.randn`
- `np.random.randint`
- `np.random.choice`
- `np.random.seed`

1. Topic Overview

What this topic is

These are **legacy random functions** from NumPy used to generate random data quickly.

Why it exists

Older NumPy versions did not have generator objects.
These functions gave simple, fast access to randomness.

One real-world analogy

Like old machines that still work and are still used in factories.

2. Core Theory (Deep but Clear)

Internal NumPy view

- Uses a **global random state**
- All functions share the same state
- `np.random.seed()` controls this state

Important consequences:

- Order of calls matters
- Harder to manage in large systems
- Still deterministic if seeded

Internally:

- Output is a NumPy array
- dtype:
  - `rand` , `randn` → float64
  - `randint` → int64
- Shape is defined by arguments, not tuples

3. Syntax & Examples

3.1 `np.random.rand`

What it does

Generates random floats from **uniform distribution [0, 1)**.

Basic syntax

```
np.random.rand(d0, d1, ..., dn)
```

Example 1

```
np.random.rand(5)
```

Output

```
[0.41 0.73 0.12 0.89 0.55]
```

Explanation

- Shape: (5,)
- Range: 0 to 1
- dtype: float64

Example 2

```
np.random.rand(2, 3)
```

Output

```
[[0.2  0.8  0.5]
 [0.9  0.1  0.4]]
```

Explanation

- Shape: (2, 3)

- Arguments define dimensions directly

3.2 `np.random.randn`

What it does

Generates random floats from **standard normal distribution**.

Basic syntax

```
np.random.randn(d0, d1, ..., dn)
```

Example

```
np.random.randn(4)
```

Output

```
[-0.32  1.45  0.08 -1.12]
```

Explanation

- Mean ≈ 0
- Std ≈ 1
- Used in ML weight initialization

3.3 `np.random.randint`

What it does

Generates random integers.

Basic syntax

```
np.random.randint(low, high, size)
```

Example 1

```
np.random.randint(1, 10, 6)
```

Output

```
[3 9 1 7 4 6]
```

Explanation

- Range: [1, 10)
- dtype: int64

Example 2

```
np.random.randint(0, 2, (3, 3))
```

Output

```
[[0 1 0]
 [1 1 0]
 [0 0 1]]
```

Explanation

- Binary matrix
- Used in labels and masks

3.4 `np.random.choice`

What it does

Randomly samples values from an array.

Basic syntax

```
np.random.choice(a, size, replace=True, p=None)
```

Example

```
data = np.array([10, 20, 30])
np.random.choice(data, 5)
```

Output

```
[20 10 30 20 20]
```

Explanation

- Sampling with replacement by default
- Output shape: (5,)

Example with probabilities

```
np.random.choice([0, 1], 10, p=[0.8, 0.2])
```

3.5 `np.random.seed`

What it does

Sets the global random state.

Basic syntax

```
np.random.seed(value)
```

Example

```
np.random.seed(0)
np.random.rand(3)
```

Output

```
[0.5488135  0.71518937 0.60276338]
```

Explanation

- Same seed → same output
- Affects ALL `np.random.*` calls

4. Why This Matters in Data Science

Data cleaning

- Generate dummy data
- Test pipelines

Feature engineering

- Add noise
- Create synthetic features

Model input preparation

- Fake datasets
- Train-test splitting logic

ML / DL pipelines

- Weight initialization
- Reproducibility

What breaks if you don’t understand this

- Non-repeatable experiments
- Hidden bugs
- Unstable model results

5. Common Mistakes (VERY IMPORTANT)

1. Forgetting to set seed
2. Mixing `rand` and `randn` blindly
3. Assuming `randint` includes high value
4. Using `choice` without probabilities check
5. Relying on global state in large projects

6. Performance & Best Practices

When fast

- Vectorized generation
- Small to medium arrays

When slow

- Calling inside loops
- Re-seeding repeatedly

Memory warnings

- Large shapes allocate large blocks
- Always verify shape

Best practices

- Use for quick experiments
- Avoid in production systems
- Prefer `default_rng` for large projects

7. 20 Practice Problems

Easy (5)

1. Generate 10 uniform random numbers
2. Create 5 standard normal values
3. Generate random integers between 50 and 100
4. Sample 8 values from a list
5. Verify reproducibility using seed

Medium (7)

6. Create random feature matrix (100×5)
7. Generate random binary labels
8. Add Gaussian noise to dataset
9. Create random train-test split
10. Simulate dice rolls
11. Generate random dropout mask
12. Create weighted random samples

Hard (5)

13. Random weight initialization for NN
14. Simulate imbalanced classification labels
15. Monte Carlo simulation using `rand`
16. Bootstrap resampling using `choice`
17. Stress test model with random noise

Industry-Level Tasks (3)

18. Create reproducible ML experiment
19. Simulate synthetic dataset for demo
20. Debug randomness-related model bugs

8. Mini Checklist

- These use global random state
- Seed controls everything
- `rand` ≠ `randn`
- `randint` excludes high
- `choice` supports probabilities
- Not ideal for large systems