CHEAT SHEET · ALIJABBARY.COM

NumPy Quick Reference for ML

The NumPy idioms you'll see in every PyTorch / JAX / scikit-learn codebase — and the speed wins from using them right.

01 · CREATION

Building arrays without footguns

np.zeros((m, n))	All zeros, dtype=float64 by default — set dtype=np.float32 for ML
np.ones_like(x)	Match shape + dtype of an existing array
np.arange(start, stop, step)	Like Python range, but returns ndarray
np.linspace(start, stop, num)	num points evenly spaced; INCLUSIVE of stop
np.random.default_rng(seed)	Modern RNG — never use np.random.seed in new code
rng.standard_normal((n, d))	N(0, 1) samples; faster + better than np.random.randn

02 · SHAPE & AXES

The mental model that prevents 80% of bugs

Every NumPy bug is a shape bug. Make shapes explicit, name your axes, and reach for reshape / broadcasting before loops.

- x.shape returns a tuple — (m, n) is m rows, n columns
- axis=0 collapses rows (one result per column); axis=1 collapses columns (one result per row)
- Reshape with -1 to infer one dim: x.reshape(-1, 1) makes a column vector of any length
- np.newaxis (== None) adds a length-1 dim: x[:, None] turns (n,) into (n, 1)
- Use .squeeze() to drop length-1 dims; never assume a function returns 1-D vs 2-D

03 · BROADCASTING

The rule that makes vectorised code feel like magic

Two arrays broadcast together when, comparing shapes right-to-left, each dim is either equal or one of them is 1. That's the whole rule.

(m, n) + (n,)	→ (m, n) — adds the vector to every row
(m, n) + (m, 1)	→ (m, n) — adds the column vector down each column
(m, n) + (m,)	— make it (m, 1) first with x[:, None]
(m, n) - x.mean(axis=0)	Centre each column — broadcasting handles the shape

(m, n) / x.std(axis=0, keepdims=True)	Standardise — keepdims=True preserves the (1, n) shape for clean math

04 · INDEXING

Slicing, masks, and fancy indexing

- x[i:j, k:l] returns a VIEW (no copy) — mutations propagate
- x[[1, 3, 5]] returns a COPY — fancy/integer indexing always copies
- Boolean masks: x[x > 0] flattens to 1-D; x[mask, :] preserves columns
- np.where(cond, a, b) — vectorised ternary, faster than a loop or list comprehension
- np.argpartition(x, k) for top-k without full sort — O(n) vs O(n log n)

05 · LINEAR ALGEBRA OPS

The 8 ops that cover most ML code

a @ b or np.matmul	Matrix multiplication — prefer @ since Python 3.5
np.einsum('ij,jk→ik', a, b)	Explicit dims; readable, fast, fewer bugs than complex matmul chains
np.linalg.norm(x, axis=-1)	Vector norms — axis=-1 across the last dim is the ML convention
np.linalg.inv / pinv	Use pinv (pseudo-inverse) for non-square or near-singular matrices
np.linalg.svd(x)	U, s, Vh = np.linalg.svd(x); the workhorse of PCA + recommenders
np.linalg.solve(A, b)	Solve Ax = b — faster + more stable than inv(A) @ b

06 · PERFORMANCE

Vectorise, then profile, then maybe Cythonise

- Replace Python loops over ndarrays with NumPy ops — usually 10-100x speedup
- Use the right dtype: float32 doubles your throughput on most CPUs and all GPUs
- np.concatenate is O(N) every call — pre-allocate or use lists then convert once
- When NumPy isn't fast enough: try Numba @jit before reaching for Cython / Rust
- Memory-map large arrays with np.memmap; never load a 50 GB dataset into RAM

Beyond the basics

COMMON PITFALLS

- ! Mixing dtypes accidentally — np.array([1, 2.0, 3]) becomes float64; downcast explicitly with .astype(np.float32) for ML pipelines
- ! Modifying a view thinking it's a copy — use .copy() when you intend to fork the data
- ! Forgetting keepdims=True when normalising — shape mismatches downstream are silent until they're loud
- ! Using np.matrix — deprecated; the @ operator on ndarray does everything you wanted
- ! Re-seeding the global RNG with np.random.seed in a library — use a local Generator, never touch global state

FURTHER READING

- → NumPy user guide (official) (https://numpy.org/doc/stable/user/index.html)
- → From Python to NumPy — Nicolas P. Rougier (free book) (https://www.labri.fr/perso/nrougier/from-python-to-numpy/)
- → Visual broadcasting guide (https://numpy.org/doc/stable/user/basics.broadcasting.html)
- → 100 NumPy Exercises (rougier) (https://github.com/rougier/numpy-100)