numpy

Numbers in arrays: fast arithmetic, logs and exponentials, grids, random draws, linear algebra

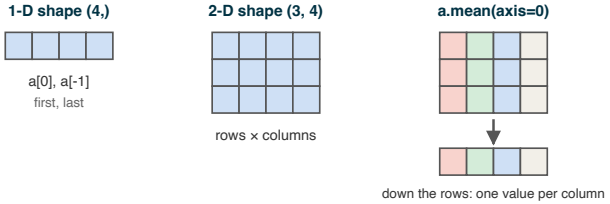
Python cheatsheet · MATH60033A · Quantitative Methods in IB · used in sessions 2 to 13

```
import numpy as np
```

pandas columns are numpy arrays underneath

1 · Arrays and their shape

An array is a grid of numbers of one type. Its shape says how many rows and columns.



`np.array([1.0, 2.5, 4.0])`
an array from a list.

`np.asarray(d["gdp"])` · `d.to_numpy()`
a pandas column or table as a plain array — what scikit-learn works on.

```
X = d[XCOLS].to_numpy(float)
```

`a.shape` · `a.ndim` · `len(a)`
(rows, columns) · number of dimensions · number of rows.

`a[0]` · `a[-1]` · `a[1:3]` · `a[:, 0]`
first · last · positions 1–2 · the first column of a 2-D array.

`a[a > 0]`
keep the elements that meet a condition.

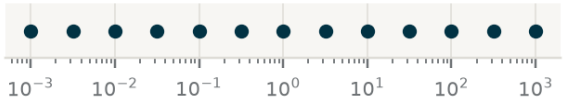
2 · Make arrays

`np.arange(0, 10, 2)`
0, 2, 4, 6, 8 — steps of 2, the end excluded.

`np.linspace(-3, 3, 200)`
200 evenly spaced points: the x-axis of a smooth curve.

`np.logspace(-3, 3, 60)`
60 values from 0.001 to 1000, even on a **log** scale: the grid of λ for ridge and lasso.

`np.zeros(5)` · `np.ones((3, 2))` · `np.full_like(y, y.mean())`
filled arrays; the last one is a "predict the mean" benchmark.



np.logspace(-3, 3, 13): evenly spaced on a log axis

3 · Arithmetic, element by element

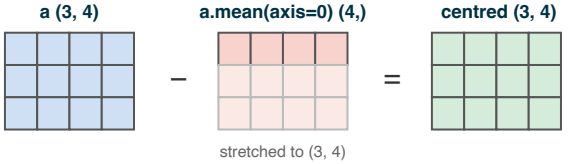
Operators and functions act on every element at once — no loop.

`a + b` · `a * 2` · `a ** 2`
element by element; arrays of different shapes are **broadcast**.

`np.sqrt(a)` · `np.abs(a)` · `np.sign(a)`
square root · absolute value · −1, 0 or +1.

```
rmse = np.sqrt(np.mean((y - y_hat) ** 2))
```

`np.round(a, 3)` · `np.clip(a, 0, None)`
round · cap below at 0 (the lasso's "keep only what is above λ").



shapes are matched from the right; a length-1 side repeats

4 · Summaries, and the axis

`np.mean(a)` · `np.median(a)` · `np.std(a, ddof=1)`
centre and spread; `ddof=1` divides by n − 1, as the sample SD does.

`np.sum(a)` · `np.cumsum(a)`
total · running total.

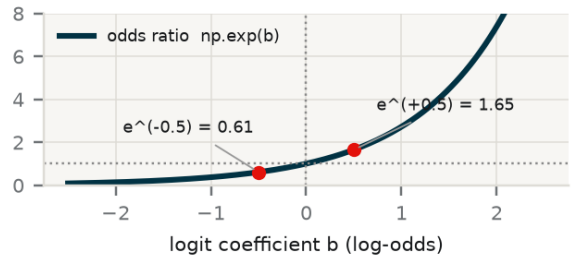
`a.mean(axis=0)` · `a.mean(axis=1)`
down the rows (one per column) · across the columns (one per row).

`np.sum(coef != 0)`
count what is True: here, how many coefficients the lasso kept.

```
kept = int(np.sum(np.abs(pipe[-1].coef_) > 1e-8))
```

`np.argmax(a)` · `np.argsort(-np.abs(a))`
position of the largest · positions from largest to smallest in size.

5 · Logs and exponentials



`np.log(x)`
natural log. A skewed outcome on a log scale: a slope then reads in %.

```
y = np.log(d.fdi_stock_musd)
```

`np.exp(b)`
undo the log. On a logit coefficient: log-odds → **odds ratio**.

```
np.exp(m2.params["fico"])     # 0.988: odds 1.2% lower per point
```

`np.exp(xb) / (1 + np.exp(xb))`
the logistic function: a linear index → a probability between 0 and 1.

> An odds ratio above 1 raises the odds; below 1 lowers them. It multiplies the odds, not the probability.

6 · Conditions

`np.where(x > 0, "up", "down")`
one value where the condition holds, another elsewhere.

`np.maximum(a, 0)`
element by element, the larger of two: here, negative values become 0.

7 · Random draws, reproducibly

Simulations make the bias–variance trade-off visible. Fix the seed, so a run can be repeated.

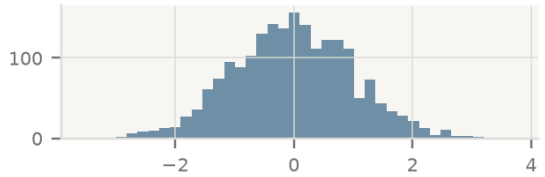
`rng = np.random.default_rng(7)`
a random generator with a seed: the same numbers on every machine.

`rng.normal(0, 1, size=n)`
n draws from a normal distribution, mean 0, sd 1.

`rng.integers(0, n, n)`
n row numbers drawn with replacement — a bootstrap resample.

```
idx = rng.integers(0, len(d), len(d)); d.iloc[idx]
```

`rng.multivariate_normal(mean, cov, n)`
correlated draws: two predictors at 0.95, for the ridge picture.



rng.normal(0, 1, 2000) — the same every run with seed 1

8 · Correlation and linear algebra

The arithmetic behind a regression, when you want to see it.

`np.corrcoef(x, y)`
the correlation matrix of two (or more) variables.

```
np.corrcoef(rd, patents)[0, 1]   # 0.95: near-twins
```

`X.T @ X · X.T @ y`
`.T` transposes, `@` multiplies matrices — the pieces of OLS.

`np.linalg.solve(X.T @ X, X.T @ y)`
the OLS coefficients, $(X^TX)^{-1}X^Ty$, without forming the inverse.

`np.linalg.solve(XtX + lam * np.eye(p), Xty)`
ridge: add λ on the diagonal — solvable even when X^TX is not.

`np.linalg.inv(A) · np.linalg.pinv(A)`
inverse · pseudo-inverse, which works on a nearly singular matrix.

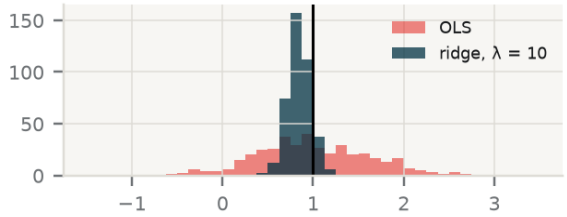
`np.linalg.eigvalsh(np.corrcoef(Z, rowvar=False))`
eigenvalues of a correlation matrix: the scree plot of PCA.

9 · A simulation in a few lines

Why ridge helps, shown with numpy alone: re-draw the data many times, re-fit, look at the spread.

`for _ in range(S): ...`
draw a sample, fit OLS and ridge, keep the slope.

```
Xs = rng.multivariate_normal([0, 0], cov, 30)
ys = Xs @ [1, 1] + rng.normal(0, 1, 30)
```



slope estimate in 400 samples (truth = 1)

10 · Combine and reshape

`np.vstack([a, b]) · np.c_[x, y]`
stack rows on top of each other · put arrays side by side as columns.

`a.reshape(3, 4) · a.ravel()`
the same numbers in a new shape · flattened back to 1-D.

`G1, G2 = np.meshgrid(g, g)`
every pair of two grids: the canvas for a contour plot (ridge and lasso budgets).

`np.diag(a) · np.eye(p)`
a diagonal matrix · the identity, as in $X^TX + \lambda I$.

11 · numpy and pandas

Most of the time pandas holds the data and numpy does the arithmetic on its columns.

`np.log(d["gdp"])`
numpy functions accept a pandas column and return one, index kept.

`d.to_numpy() · pd.DataFrame(a, columns=[...])`
table → array · array → table with names.

`m.predict(X).values`
`.values` takes the plain array out of a pandas result.

> Prefer pandas while the column names help you; drop to numpy for maths that has no name.

12 · When numpy complains

`ValueError: operands could not be broadcast`
the shapes do not line up: print `a.shape` and `b.shape`.

`nan in the result`
a missing value spreads: `np.nanmean(a)` skips it, or drop it in pandas first.

`RuntimeWarning: divide by zero in log`
`np.log(0)` is –inf: keep positive values only, or use `np.log1p`.