NumPy for Machine Learning

15-Minute Daily Lessons (with Colab Exercises)

Author: Erick Kendall

Prepared with assistance from ChatGPT

Lesson 00 Book Intro

NumPy for Machine Learning: 15-Minute Daily Lessons

Preface

This book is designed as a practical, hands-on introduction to **NumPy**, tailored specifically for readers preparing to work through *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*.

The lessons are short (≈15 minutes each), runnable in Google Colab, and intended for beginners with basic Python knowledge. They cover the essential NumPy concepts required to succeed in machine learning workflows.

How to Use This Book

- Each lesson is a standalone **Colab notebook**.
- Work through one per day (≈3 lessons per week = 6 weeks, or daily = 3 weeks).
- Each lesson has examples, explanations, and a short exercise.
- Open in Colab, run cells, and try the exercises.

Study Schedule

The 18 lessons can be studied over two to three weeks:

- **Week 1:** Lessons 1–6 (Foundations & Indexing)
- **Week 2:** Lessons 7–12 (Boolean logic, scaling, reshaping, broadcasting)
- **Week 3:** Lessons 13–18 (Aggregations, randomness, performance, advanced indexing, capstone)

Contents

License & Sharing

This book is intended to be shared freely. Feel free to adapt the lessons, improve the explanations, and share with other learners.

Happy learning!

Lesson 01 Arrays Basics

Lesson 1: Arrays & Basic Operations
Goal (~15 min): Create arrays, inspect shapes, and run basic math used in ML.

Setup

```
import numpy as np
```

Creating Arrays

```
x = np.array([1,2,3,4,5]); print("1D:", x)
X = np.array([[1,2,3],[4,5,6]]); print("\n2D:\n", X)
zeros = np.zeros((2,3)); ones = np.ones((3,2))
I = np.eye(3)
print("\nzeros:\n", zeros); print("\nones:\n", ones); print("\nI:\n", I)
```

Shapes & Attributes

```
print("shape:", X.shape); print("ndim:", X.ndim); print("size:", X.size); print("dtype:", X.dtype)
```

Elementwise Math & Dot Product

```
a = np.array([10,20,30]); b = np.array([1,2,3])
print("add:", a+b); print("mul:", a*b); print("div:", a/b)
print("dot:", np.dot(a,b))
```

Exercise
1) Build a (4×3) array with 1..12.
2) Column sums (axis=0).
3) Row means (axis=1).

```
arr = np.arange(1,13).reshape(4,3)
print(arr)
print("col sums:", arr.sum(axis=0))
print("row means:", arr.mean(axis=1))
```

Lesson 02 Indexing Slicing Masking

Lesson 2: Indexing, Slicing & Boolean Masking
Goal (~15 min): Select rows/cols, slice subarrays, and filter with masks.

Setup

```
import numpy as np
np.set_printoptions(edgeitems=4, linewidth=120)
```

Data

```
data = np.array([[25,50,72,2],
                 [32,80,88,6],
                 [41,120,55,15],
                 [29,60,45,4],
                 [36,90,95,8],
                 [50,150,62,20]], dtype=float)
header = ["age","income_k","score","years_experience"]
print("data:\n", data)
```

Indexing & Slicing

```
print("row 0:", data[0])
print("col 2 (score):", data[:,2])
print("rows 1..3 & cols 1..2:\n", data[1:4, 1:3])
```

Boolean Masks

```
mask_age = data[:,0] > 35
print("age>35 mask:", mask_age)
print("rows age>35:\n", data[mask_age])
```

Exercise
1) Rows with income ≥ 80 and credit rating? (add a 5th col to test)
2) Set score=0 where age<30 (work on a copy).

Lesson 03 Boolean Indexing NewDataset

Lesson 3: Boolean Indexing & Masking (New Dataset)
Goal (~15 min): Filter, combine conditions, and modify with masks.

Setup & Headers (kept separate)

```
import numpy as np
header = ["age","income_k","score","years_experience","credit_rating"]
data = np.array([
    [25, 50, 72,  2, 600],
    [32, 80, 88,  6, 720],
    [41,120, 55, 15, 680],
    [29, 60, 45,  4, 590],
    [36, 90, 95,  8, 750],
    [50,150, 62, 20, 710]
], dtype=float)
print("Headers:", header); print("\nData:\n", data)
```

Single & Combined Conditions

```
mask_age = data[:,0] > 35
print("age>35:\n", data[mask_age])
mask_income_credit = (data[:,1] >= 80) & (data[:,4] >= 700)
print("\nincome>=80 & credit>=700:\n", data[mask_income_credit])
```

Modify with Masks

```
data_mod = data.copy()
data_mod[data_mod[:,3] > 10, 2] += 5  # score += 5 if years_exp>10
print("modified (score boosts where years_exp>10):\n", data_mod)
```

Exercise
1) credit<650 rows
2) income in [60,100]
3) score=0 for age<30 (on a copy)
4) Count score > mean(score)

Lesson 04 CSV Headers Sampling

Lesson 4: CSV with Headers & Random Sampling
Goal (~15 min): Read CSV, keep headers, skip header for NumPy, and sample rows.

Create a Sample CSV

```
import numpy as np
csv = """age,income_k,score,years_experience,credit_rating
25,50,72,2,600
32,80,88,6,720
41,120,55,15,680
29,60,45,4,590
36,90,95,8,750
50,150,62,20,710
"""
with open("sample_data.csv","w") as f: f.write(csv)
print("Wrote sample_data.csv")
```

Read Header Separately, Load Numeric Data

```
with open("sample_data.csv","r") as f:
    header = f.readline().strip().split(",")
data = np.loadtxt("sample_data.csv", delimiter=",", skiprows=1)
print("Headers:", header); print("\nData:\n", data)
```

Random Sampling & Train/Test Split

```
np.random.seed(42)
n = data.shape[0]; train_n = 4
train_idx = np.random.choice(n, size=train_n, replace=False)
test_idx  = np.setdiff1d(np.arange(n), train_idx)
train, test = data[train_idx], data[test_idx]
print("train idx:", train_idx); print("test idx:", test_idx)
print("\nTrain:\n", train); print("\nTest:\n", test)
```

Exercise
1) Load all but last col with `usecols`
2) Make 4-row train set (no duplicates)
3) Print which headers you kept

Lesson 05 Stacking Combining

Lesson 5: Stacking & Combining Arrays
Goal (~15 min): hstack/vstack; build features, add new derived columns.

Load sample & split X/y

```
import numpy as np
with open("sample_data.csv","r") as f: header = f.readline().strip().split(",")
data = np.loadtxt("sample_data.csv", delimiter=",", skiprows=1)
X = data[:, :4]; y = data[:, 4]
print("X:\n", X); print("\ny:\n", y)
```

Add Derived Feature (age>35 → 1 else 0)

```
age_flag = (X[:,0] > 35).astype(int).reshape(-1,1)
X2 = np.hstack([X, age_flag])
print("X with flag:\n", X2)
```

Add New Samples (vstack)

```
new_samples = np.array([[28, 70, 81, 3, 640],
                        [45,110, 68,12, 695]], dtype=float)
full = np.vstack([data, new_samples])
print("full shape:", full.shape)
```

Exercise
1) Normalize score 0–1 and append
2) vstack [40,100,78,10,705]
3) Print final shapes

Lesson 06 Stats Summaries

Lesson 6: Statistical Summaries
Goal (~15 min): Column-wise mean/std/min/max/percentiles and simple outlier flags.

Load Data

```
import numpy as np
with open("sample_data.csv","r") as f: header = f.readline().strip().split(",")
data = np.loadtxt("sample_data.csv", delimiter=",", skiprows=1)
print("Headers:", header); print("Shape:", data.shape)
```

Column Stats

```
means = data.mean(axis=0); stds = data.std(axis=0)
mins = data.min(axis=0); maxs = data.max(axis=0)
p25 = np.percentile(data, 25, axis=0); p75 = np.percentile(data, 75, axis=0)
print("means:", means); print("stds:", stds)
print("mins:", mins); print("maxs:", maxs)
print("p25:", p25); print("p75:", p75)
```

Z-score Example on credit_rating

```
credit = data[:,-1]; z = (credit - credit.mean())/credit.std()
print("z-scores:", np.round(z, 2))
print("outliers (|z|>2):\n", data[np.abs(z)>2])
```

Exercise
1) score mean/std; z-normalize score
2) 25th & 75th of income_k
3) Rows with income_k > 75th

Lesson 07 Lists vs Numpy Boolean Drills

Lesson 7: NumPy vs Python Lists + Mask Drills
Goal (~15 min): See vectorization benefits; practice Boolean indexing.

Lists vs Arrays

```
import numpy as np
py_list = [1,2,3,4,5]
np_array = np.array([1,2,3,4,5])
try:
    py_list + 10
except Exception as e:
    print("List + 10 error:", e)
print("Array + 10:", np_array + 10)
try:
    py_list > 3
except Exception as e:
    print("List > 3 error:", e)
print("Array > 3:", np_array > 3)
```

Mini Dataset & Drills

```
data = np.array([[25, 50, 72, 2],
                 [32, 80, 88, 6],
                 [41,120, 55,15],
                 [29, 60, 45, 4],
                 [36, 90, 95, 8],
                 [50,150, 62,20]], dtype=float)
print("age>35:\n", data[data[:,0]>35])
print("income<100:\n", data[data[:,1]<100])
mask = (data[:,2]>=60)&(data[:,2]<=90); print("score in [60,90]:\n", data[mask])
```

Exercise
1) age ≤ 30
2) income ≥ 80 & score > 80
3) score=0 where income<70 (copy)
4) Count years_exp > mean(years_exp)

Lesson 08 Scaling Standardization

Lesson 8: Scaling & Standardization (fit/transform)
Goal (~15 min): Min–max and z-score scaling; fit on train, apply to test.

Clean Formula (LaTeX)

We will reference this baseline formula in later derivations:

$$
\text{baseline\_hours} = \left\lfloor \frac{\text{avg} - 70}{5} \right\rfloor
$$

Synthetic Data & Split

```
import numpy as np
np.random.seed(7)
N = 14
age = np.random.randint(22,61,size=N)
income_k = np.random.randint(40,181,size=N)
score = np.random.randint(40,101,size=N)
years_experience = np.clip((age-22)//3 + np.random.randint(-1,3,size=N), 0, None)
credit_rating = np.clip(500 + income_k*2 + (score-70)*4 + years_experience*3
                        + np.random.randint(-60,61,size=N), 300, 850)
X = np.column_stack([age,income_k,score,years_experience]).astype(float)
y = credit_rating.astype(float)
np.random.seed(42); n = X.shape[0]; tr = int(round(0.8*n))
idx = np.random.choice(n, size=tr, replace=False); te = np.setdiff1d(np.arange(n), idx)
X_train, X_test = X[idx], X[te]
```

Fit/Transform Functions

```
def fit_standardizer(X): m=X.mean(axis=0); s=X.std(axis=0); s[s==0]=1.0; return {"mean":m,"std":s}
def transform_standardizer(X, st): return (X - st["mean"]) / st["std"]
st = fit_standardizer(X_train)
X_train_std = transform_standardizer(X_train, st)
X_test_std  = transform_standardizer(X_test, st)
print("Train means ≈ 0:", X_train_std.mean(axis=0).round(3))
print("Train stds  ≈ 1:",  X_train_std.std(axis=0).round(3))
```

Exercise
Implement robust scaling with median/IQR; count test rows with any |val|>2.

Lesson 09 Split Concat Reshape

Lesson 9: Splitting, Concatenating & Reshaping
Goal (~15 min): hsplit/vsplit, hstack/vstack, ravel/reshape.

Dataset

```
import numpy as np; np.random.seed(0)
data = np.random.randint(10,100,size=(8,6))
print("shape:", data.shape); print(data)
```

Split & Concatenate

```
left, right = np.hsplit(data, [3]); print("left:\n", left); print("\nright:\n", right)
top, bottom = np.vsplit(data, [4]); print("\ntop:\n", top); print("\nbottom:\n", bottom)
new_col = np.arange(1,9).reshape(-1,1); with_col = np.hstack([data, new_col]); print("\nwith col:\n"
```

Reshape & Flatten

```
flat = data.ravel(); print("flat length:", flat.size)
reshaped = data.reshape(4,-1); print("reshaped 4x?:\n", reshaped)
```

Exercise
Split features/target, make train/test by rows, create score*years_exp feature, and report shapes.

Lesson 10 Outliers Z IQR Viz

Lesson 10: Outlier Detection (Z-score & IQR) + Viz
Goal (~15 min): Detect outliers using two methods and visualize.

Income Dataset with Outliers

```
import numpy as np
np.random.seed(42)
income_k = np.random.normal(80,15,size=100).astype(int)
income_k[[5,20,75]] = [200,250,300]
print(income_k[:15])
```

Z-score Method

```
mean = income_k.mean(); std = income_k.std()
z = (income_k - mean)/std
z_mask = np.abs(z) > 2
print("mean/std:", round(mean,2), round(std,2))
print("z outliers:", income_k[z_mask])
```

IQR Method

```
q25, q75 = np.percentile(income_k, [25,75]); iqr = q75 - q25
lb, ub = q25 - 1.5*iqr, q75 + 1.5*iqr
iqr_mask = (income_k<lb)|(income_k>ub)
print("IQR bounds:", lb, ub)
print("IQR outliers:", income_k[iqr_mask])
```

Visualization

```
import matplotlib.pyplot as plt
plt.figure()
plt.hist(income_k, bins=20)
plt.scatter(income_k[z_mask], [0]*z_mask.sum(), marker="x", s=120)
plt.scatter(income_k[iqr_mask], [0.5]*iqr_mask.sum(), facecolors="none", s=120)
plt.title("Outliers: Z-score (x) vs IQR (circles)")
plt.xlabel("Income (k)"); plt.ylabel("Frequency")
plt.show()
```

Exercise
Create a score array with two extreme outliers; detect via both methods; compare results.

Lesson 11 Linear Algebra Basics

Lesson 11: Linear Algebra Basics
Goal (~15 min): Matrix multiplication, transpose, identity, solving Ax=b.

Matrices & Shapes

```
import numpy as np
A = np.array([[1,2,3],[4,5,6]])
B = np.array([[7,8],[9,10],[11,12]])
print("A:\n", A, "\nshape:", A.shape)
print("\nB:\n", B, "\nshape:", B.shape)
```

Matrix Multiplication & Dot

```
C = A @ B; print("A@B:\n", C)
x = np.array([1,2,3]); y = np.array([4,5,6])
print("dot(x,y):", np.dot(x,y))
```

Transpose & Identity

```
print("A.T:\n", A.T)
M = np.array([[2,3],[5,7]])
print("M@I == M:\n", M @ np.eye(2))
```

Solve Ax=b

```
A = np.array([[3,1],[1,2]], dtype=float)
b = np.array([9,8], dtype=float)
x = np.linalg.solve(A,b); print("x:", x); print("A@x:", A@x)
```

Exercise
1) Random 3x3 M (1..9); compute M@M.T
2) v=[2,1,-1]; compute M@v
3) Solve 2x+y=5; x–y=1

Lesson 12 Broadcasting Deep Dive

Lesson 12: Broadcasting Deep Dive
Goal (~15 min): Understand how NumPy aligns shapes to enable fast, vectorized math (scalars ↔ vectors ↔ matrices).

Setup & Data

```
import numpy as np
np.set_printoptions(precision=3, suppress=True)
A = np.arange(12).reshape(3,4).astype(float)  # 3 rows, 4 cols
row_vec = np.array([10, 20, 30, 40], dtype=float)   # shape (4,)
col_vec = np.array([[1.0],[2.0],[3.0]])             # shape (3,1)
print("A:\n", A)
```

Shapes & Simple Broadcasting

```
print("A shape:", A.shape)
print("row_vec shape:", row_vec.shape)
print("A + row_vec:\n", A + row_vec)   # add per-column
print("\ncol_vec shape:", col_vec.shape)
print("A * col_vec:\n", A * col_vec)   # scale per-row
```

Expanding Dimensions with `None`/`np.newaxis`

```
x = np.array([1,2,3])     # (3,)
y = np.array([4,5])       # (2,)
# Outer sum via broadcasting by adding axes
outer_sum = x[:, None] + y[None, :]
print("outer_sum shape:", outer_sum.shape)
print(outer_sum)
```

Center Each Column (subtract column means via broadcasting)

```
col_means = A.mean(axis=0)        # shape (4,)
A_centered = A - col_means        # broadcast subtract
print("col_means:", col_means)
print("A_centered first row:", A_centered[0])
```

Add Bias (intercept) Column via Broadcasting

```
ones = np.ones((A.shape[0],1))
A_with_bias = np.hstack([ones, A])
print("A_with_bias shape:", A_with_bias.shape)
```

Exercise
1) Given B = np.arange(6).reshape(2,3), add [100,200,300] to B by broadcasting.
2) Multiply B by column vector [[1],[10]] to scale rows.
3) Compute Z = (A - A.mean(axis=1, keepdims=True)) / A.std(axis=1, keepdims=True) (row-standardize).

Lesson 13 Aggregations Grouping Batching

Lesson 13: Aggregations & Grouping (Axis Logic) + Mini-batching
Goal (~15 min): Master axis semantics for sum/mean, and simulate simple group/batch ops used in ML.

Setup

```
import numpy as np
np.set_printoptions(precision=3, suppress=True)
X = np.arange(1,13).reshape(3,4).astype(float)  # 3 samples × 4 features
print("X:\n", X)
```

Aggregations by Axis

```
print("sum over features (per sample):", X.sum(axis=1))  # 3 values
print("mean over samples (per feature):", X.mean(axis=0))  # 4 values
```

Batch Means (simulate mini-batches)

```
def batch_iter(arr, batch_size):
    for i in range(0, len(arr), batch_size):
        yield arr[i:i+batch_size]

for batch in batch_iter(X, 2):
    print("batch:\n", batch, " batch mean:", batch.mean(axis=0))
```

Grouped Reductions with Labels (`np.bincount`, `np.add.at`)

```
# Suppose each row belongs to a group label
labels = np.array([0, 1, 0])  # 2 samples in group 0, 1 sample in group 1
# Sum features per group using add.at
group_sums = np.zeros((labels.max()+1, X.shape[1]))
np.add.at(group_sums, labels, X)
group_counts = np.bincount(labels)
group_means = group_sums / group_counts[:, None]
print("group_sums:\n", group_sums)
print("group_means:\n", group_means)
```

Exercise
1) Compute std per feature (axis=0) and per sample (axis=1).
2) Given labels=[1,1,0], recompute group_means.
3) Write a function grouped_mean(X, labels) that returns means for each group index 0..max(labels).

Lesson 14 Randomness Reproducibility

Lesson 14: Random Numbers & Reproducibility
Goal (~15 min): Use NumPy's random generators to sample, shuffle, and split reproducibly.

Setup — Generator API

```
import numpy as np
# Recommended modern API
rng = np.random.default_rng(seed=42)  # reproducible
print("rand 3 floats:", rng.random(3))
print("randint 0..9:", rng.integers(0,10, size=5))
print("normal(0,1):", rng.normal(size=3))
```

Shuffling & Permutations

```
arr = np.arange(10)
perm = rng.permutation(arr)
print("perm:", perm)
rng.shuffle(arr)  # in-place
print("shuffled arr:", arr)
```

Train/Test Split (indices)

```
N = 12
idx = rng.permutation(N)
train_idx, test_idx = idx[:8], idx[8:]
print("train_idx:", train_idx, "\ntest_idx:", test_idx)
```

Stratified-like Split (approximate)

```
# Given binary labels, keep roughly same proportion in train/test
labels = rng.integers(0,2,size=N)  # 0/1 labels
idx0 = np.where(labels==0)[0]; idx1 = np.where(labels==1)[0]
tr0, te0 = idx0[:len(idx0)*8//10], idx0[len(idx0)*8//10:]
tr1, te1 = idx1[:len(idx1)*8//10], idx1[len(idx1)*8//10:]
train_idx2 = np.concatenate([tr0, tr1]); test_idx2 = np.concatenate([te0, te1])
print("labels:", labels)
print("train_idx2:", np.sort(train_idx2))
print("test_idx2:", np.sort(test_idx2))
```

Exercise
1) Create rng with seed=7 and draw: 5 normals, 5 uniform(0,1), 5 ints 10..20.
2) Write split_indices(N, train_frac, seed) that returns (train_idx, test_idx).
3) Bonus: Given k classes in labels, write stratified_indices(labels, train_frac).

Lesson 15 Performance Tips

Lesson 15: Performance Tips — Vectorization & Memory
Goal (~15 min): Prefer vectorized NumPy over Python loops; use boolean ops and dtypes wisely.

Setup

```
import numpy as np, time
N = 1_000_00  # 100k
x = np.random.rand(N)
y = np.random.rand(N)
```

Python Loop vs Vectorized

```
# Loop
t0 = time.time()
s = 0.0
for i in range(N):
    s += x[i]*y[i]
t1 = time.time()
loop_time = t1 - t0

# Vectorized
t0 = time.time()
s_vec = np.dot(x, y)
t1 = time.time()
vec_time = t1 - t0

print(f"loop dot={s:.4f} in {loop_time:.4f}s; vectorized dot={s_vec:.4f} in {vec_time:.6f}s")
```

Conditional Update: np.where vs Boolean Mask

```
arr = np.random.randint(0, 100, size=12).astype(float)
arr_where = np.where(arr > 50, arr, 0)     # values <=50 -> 0
arr_mask = arr.copy(); arr_mask[arr_mask <= 50] = 0
print("arr:", arr)
print("where:", arr_where)
print("mask :", arr_mask)
```

Dtypes & Casting

```
int_arr = np.arange(6, dtype=np.int16)
float_arr = int_arr.astype(np.float32)
print(int_arr.dtype, float_arr.dtype)
```

Exercise
1) Time a loop vs vectorized (square then sum) for N=200k.
2) Use np.where to cap values at 90 (else keep original).
3) Convert a float array to int32 and observe rounding/truncation.

Lesson 16 Advanced Indexing

Lesson 16: Advanced Indexing — Fancy, Mixed, and Updates
Goal (~15 min): Use integer lists/arrays for fancy indexing; mix boolean & integer; in-place updates.

Setup

```
import numpy as np
A = np.arange(24).reshape(6,4)
print("A:\n", A)
```

Fancy Row/Column Selection

```
rows = [0, 2, 5]; cols = [1, 3]
print("A[rows, :]:\n", A[rows, :])
print("A[:, cols]:\n", A[:, cols])
print("A[rows][:, cols]:\n", A[rows][:, cols])
```

Fancy with Pairing (row i, col j)

```
ri = np.array([0,1,2,3]); cj = np.array([0,1,2,3])
diag = A[ri, cj]   # picks (0,0),(1,1),(2,2),(3,3)
print("diag:", diag)
```

Mixed Boolean + Integer

```
mask = A[:,0] % 2 == 0  # even in first col
print("mask:", mask)
print("rows even first col, pick col 2:", A[mask, 2])
```

In-place Updates via Fancy Indexing

```
B = A.copy()
idx = np.array([1,3,4])
B[idx, 2] += 100  # add 100 to col 2 for selected rows
print("updated B col2 on rows [1,3,4]:\n", B)
```

Exercise
1) Select rows [0,3,5] and columns [0,2]; return that submatrix.
2) Set A rows [2,4] column 1 to -1.
3) Extract elements (0,3),(2,0),(5,1) in one call.

Lesson 17 Saving Loading

Lesson 17: Saving & Loading Data
Goal (~15 min): Use NumPy binary formats for speed/size, and text formats for portability.

Create Example Arrays

```
import numpy as np
X = np.random.rand(5,4)
y = np.random.randint(0,2,size=5)
print("X:\n", X); print("y:", y)
```

Save/Load NPY (single array)

```
np.save("X.npy", X)
X2 = np.load("X.npy")
print("loaded equal:", np.allclose(X, X2))
```

Save/Load NPZ (multiple arrays, compressed)

```
np.savez_compressed("dataset.npz", X=X, y=y)
d = np.load("dataset.npz")
print("keys:", list(d.keys()))
print("X==:", np.allclose(X, d["X"]), " y==:", np.array_equal(y, d["y"]))
```

CSV/Text Interop (human-readable)

```
np.savetxt("X.csv", X, delimiter=",", fmt="%.6f")
X_txt = np.loadtxt("X.csv", delimiter=",")
print("csv equal (approx):", np.allclose(X, X_txt))
```

Exercise
1) Save three arrays (A,B,C) into one NPZ and reload.
2) Write and read a small int array with savetxt/loadtxt using fmt='%d'.
3) Bonus: time np.save/.npy vs np.savetxt for a (1000×1000) array.

Lesson 18 Capstone NumPy Pipeline

Lesson 18: Capstone — NumPy-Only Mini ML Pipeline
Goal (~15–20 min): Build a tiny end-to-end regression pipeline using only NumPy.
- Load synthetic data
- Split train/test
- Scale (fit on train, apply to test)
- Feature engineer
- Fit linear regression (normal equation)
- Evaluate RMSE

1) Data Generation

```
import numpy as np
rng = np.random.default_rng(123)
N = 200
age = rng.integers(22, 61, size=N)
income_k = rng.integers(40, 181, size=N)
score = rng.integers(40, 101, size=N)
years_exp = np.clip((age-22)//3 + rng.integers(-1,3,size=N), 0, None)
# true target (credit-like), with noise
y = np.clip(500 + income_k*2 + (score-70)*4 + years_exp*3 + rng.integers(-60,61,size=N), 300, 850).a
X = np.column_stack([age, income_k, score, years_exp]).astype(float)
header = ["age","income_k","score","years_experience"]
```

2) Train/Test Split

```
idx = rng.permutation(N)
tr = idx[:160]; te = idx[160:]
X_tr, X_te = X[tr], X[te]; y_tr, y_te = y[tr], y[te]
print("train:", X_tr.shape, " test:", X_te.shape)
```

3) Fit/Transform Scaling (Standardization)

```
def fit_standardizer(X):
    m = X.mean(axis=0); s = X.std(axis=0); s[s==0] = 1.0
    return {"mean": m, "std": s}
def transform_standardizer(X, st):
    return (X - st["mean"]) / st["std"]
st = fit_standardizer(X_tr)
X_tr_std = transform_standardizer(X_tr, st)
X_te_std = transform_standardizer(X_te, st)
```

4) Feature Engineering

```
age_flag = (X_tr[:,0] > 35).astype(float).reshape(-1,1)
age_flag_te = (X_te[:,0] > 35).astype(float).reshape(-1,1)
# also add min-max normalized score (fit on train for stability)
sc = X_tr[:,2]; sc_min = sc.min(); sc_rng = np.ptp(sc) if np.ptp(sc)!=0 else 1.0
score_mm_tr = ((X_tr[:,2]-sc_min)/sc_rng).reshape(-1,1)
score_mm_te = ((X_te[:,2]-sc_min)/sc_rng).reshape(-1,1)
# build design matrices (standardized + engineered)
Xb_tr = np.hstack([np.ones((X_tr.shape[0],1)), X_tr_std, age_flag, score_mm_tr])
Xb_te = np.hstack([np.ones((X_te.shape[0],1)), X_te_std, age_flag_te, score_mm_te])
print("Xb_tr shape:", Xb_tr.shape)
```

5) Linear Regression via Normal Equation

```
# theta = (X^T X)^(-1) X^T y
XtX = Xb_tr.T @ Xb_tr
Xty = Xb_tr.T @ y_tr
theta = np.linalg.solve(XtX, Xty)
print("theta shape:", theta.shape)
```

6) Evaluate RMSE on Test

```
y_pred = Xb_te @ theta
rmse = np.sqrt(np.mean((y_te - y_pred)**2))
print("Test RMSE:", round(rmse, 3))
print("Pred sample:", y_pred[:5].round(2))
```

Exercise

1) Try replacing standardization with min–max scaling (fit on train).
2) Add an interaction feature: income_k * score (scaled) and refit.
3) Report new RMSE and compare.