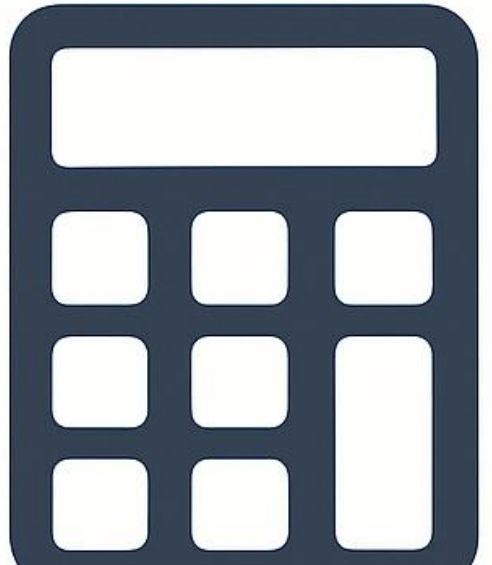
MODULE 2, LESSON 2.1: INTRODUCTION TO NUMPY FOR NUMERICAL OPERATIONS

NumPy

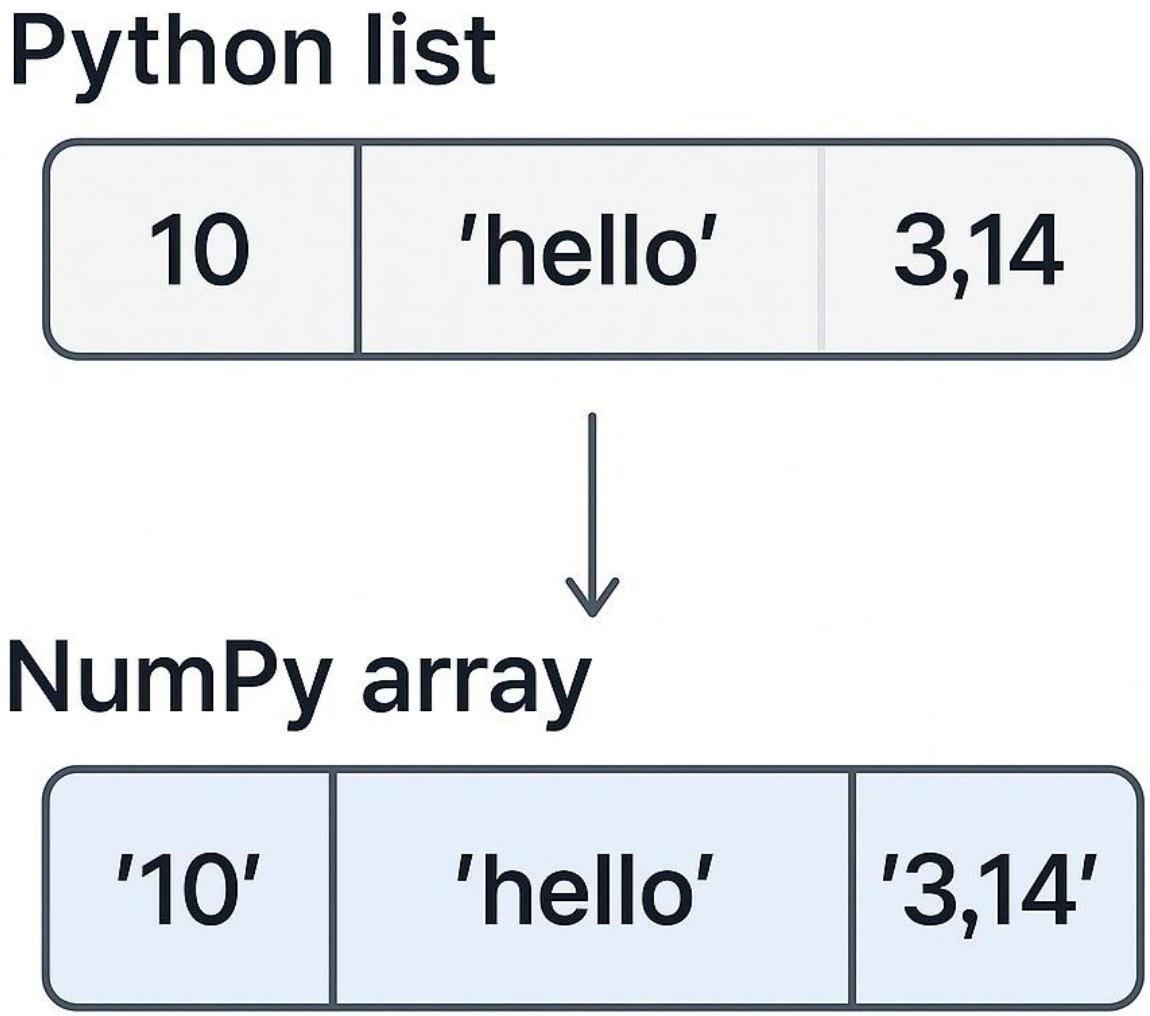
THE FOUNDATION FOR FAST NUMERICAL COMPUTING:

UNDERSTAND NUMPY ARRAYS AND THEIR EFFICIENCY FOR NUMERICAL DATA IN CLINICAL TRIALS.



NUMPY ARRAYS: BEYOND PYTHON LISTS

- **What is a NumPy Array?** A grid of values, all of the same type (homogeneous).
- **Comparison to Python Lists:**
  - Lists can hold mixed data types; Arrays are homogeneous.
  - Arrays are designed for numerical operations; Lists are general-purpose.
- **Why it Matters:** Clinical data often involves large numerical datasets (lab values, vital signs, patient measurements).

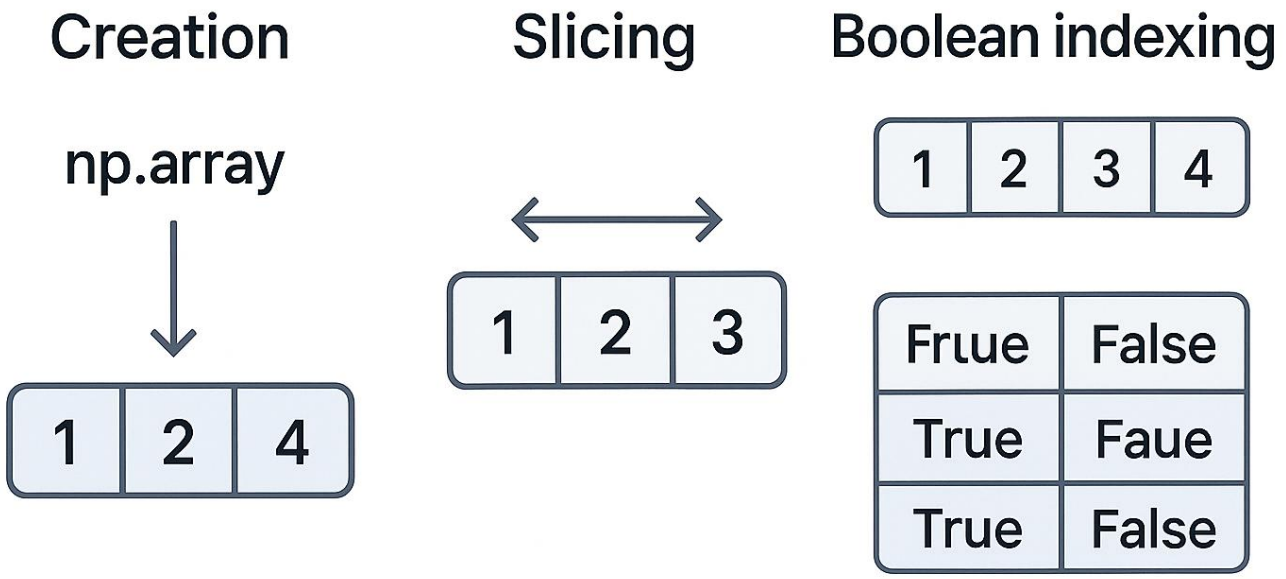


ADVANTAGES OF NUMPY: SPEED, MEMORY, VECTORIZATION

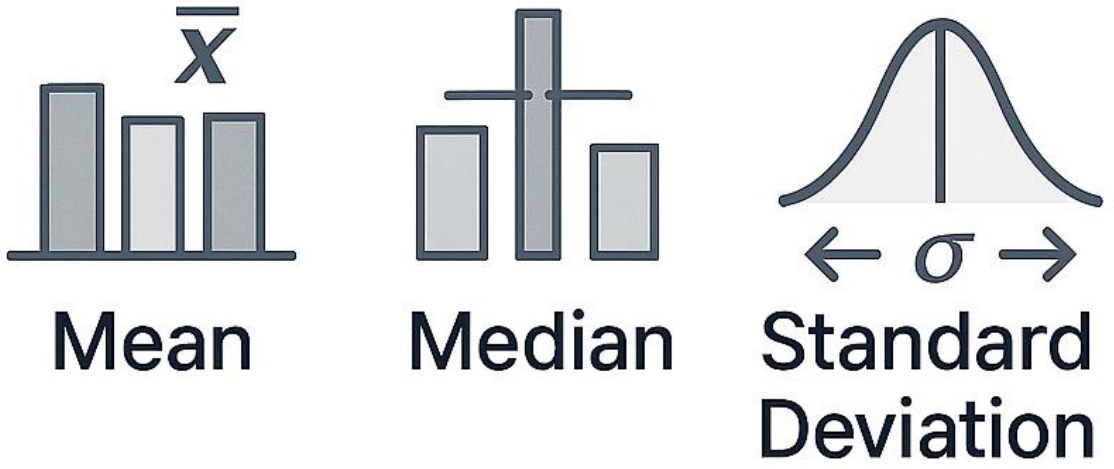
- **Speed:** Operations are implemented in C, making them significantly faster for large numerical computations.
- **Memory Efficiency:** Less memory consumption due to homogeneous data storage.
- **Vectorized Operations:** Perform operations on entire arrays at once, without explicit loops. This is concise and fast.
  - *Clinical Example:* Multiplying all blood pressure readings by a conversion factor, or summing a column of lab values.

BASIC ARRAY OPERATIONS: CREATION, SLICING, FILTERING

- **Array Creation:** From Python lists (np.array()), np.zeros(), np.ones(), np.arange().
- **Arithmetic Operations:** Element-wise addition, subtraction, multiplication, division.
- **Slicing:** Selecting subsets of data (rows, columns, or specific elements).
    - *Clinical Example:* Extracting all glucose readings or specific patient's vital signs.
- **Filtering (Boolean Indexing):** Selecting elements based on a condition.
    - *Clinical Example:* Identifying all lab values above a certain threshold.



COMMON AGGREGATIONS WITH NUMPY



- **Purpose:** Summarize numerical data efficiently.
- **Functions:**
  - np.mean(): Average value.
  - np.median(): Middle value.
  - np.std(): Standard deviation.
  - np.sum(): Total sum.
  - np.min(), np.max(): Minimum and maximum values.
- **Clinical Application:** Calculating summary statistics for cohorts (e.g., mean age, standard deviation of baseline lab values).

Measure	Symbol	Example Value
Mean	μ o $\bar{x}$	25,4
Median	M	24,0
Standard Deviation	σ	3,2

NUMPY IN ACTION: CODE EXAMPLES

Content:

- **Dummy Data Description:** Numerical data for blood pressure and patient lab results, provided as CSV content in a separate "Dummy Data" immersive block.
- **SAS Code Equivalent (Conceptual):** Explain how SAS handles numerical operations inherently through data steps and procedures (PROC MEANS, WHERE clause).
- **Python Code:** Provide runnable Python code snippets demonstrating array creation (from loaded CSV), arithmetic, slicing, filtering, and aggregations.

DUMMY DATA: .CSV

```
USUBJID,BP_SYSTOLIC,BP_DIASTOLIC,GLUCOSE,CHOLESTEROL,CREATINE
SUBJ001,120,80,90,180,0.8
SUBJ002,135,85,105,210,1.2
SUBJ003,110,70,85,170,0.9
SUBJ004,140,90,110,200,1.1
SUBJ005,125,82,95,190,0.7
```

Python	SAS

`# --- 1. NumPy Arrays vs. Python Lists ---`
 	
`# Python List`
`python_list = [120, 135, 110, 140, 125]`
`print(f"Python List: {python_list}")`
`# Cannot directly perform element-wise operations without a loop`
`# python_list * 2 # This would repeat the list, not multiply elements`
	SAS doesn't have a direct "array" equivalent to NumPy's ndarray in the same programmatic sense, as it operates on datasets. However, the efficiency of SAS procedures for numerical tasks is analogous to NumPy's vectorized operations.

Python

```
# --- 2. Advantages: Speed, Memory, Vectorized Operations ---

 # Example of Vectorized Operations:

# Multiply all readings by a factor (e.g., for unit conversion, though not typical for BP)

bp_converted = bp_readings * 0.75

print(f"BP Readings * 0.75: {bp_converted}")


# Add a constant to all readings

bp_plus_10 = bp_readings + 10

print(f"BP Readings + 10: {bp_plus_10}")


# Element-wise operations between two arrays of the same shape

systolic_bp = np.array([120, 135, 110, 140])

diastolic_bp = np.array([80, 85, 70, 90])

mean_bp = (systolic_bp + diastolic_bp) / 2

print(f"Mean BP (Systolic + Diastolic)/2: {mean_bp}")
```

SAS

```
/* SAS Numerical Data & Operations */
/* Define a simple dataset for comparison */
data lab_values;
    input bp_systolic bp_diastolic;
    bp_mean = (bp_systolic + bp_diastolic) / 2;
    bp_diff = bp_systolic - bp_diastolic;
    datalines;
120 80
135 85
110 70
140 90
;
run;
proc print data=lab_values; run;
```

Python

```
# --- 3. Basic Array Operations ---

# Creating 2D array (matrix) - e.g., lab results for multiple patients

# Rows: Patients, Columns: Lab Tests (Glucose, Cholesterol, Creatinine)

patient_lab_data = np.array([

   [90, 180, 0.8],    # Patient 1

   [105, 210, 1.2],   # Patient 2

   [85, 170, 0.9],    # Patient 3

   [110, 200, 1.1]    # Patient 4])

print(f"\nPatient Lab Data (2D Array):\n{patient_lab_data}")

print(f"Shape of array (rows, columns): {patient_lab_data.shape}")

 # Slicing: Accessing elements/subsets

print(f"\nFirst patient's lab data: {patient_lab_data[0, :]}")

# First row, all columns

print(f"Glucose levels for all patients: {patient_lab_data[:, 0]}") # All rows,
first column (Glucose)

print(f"Patient 2's Cholesterol: {patient_lab_data[1, 1]}")

# Row 2, Column 2

# Filtering (Boolean Indexing): Select patients with high glucose (>100)

high_glucose_patients = patient_lab_data[patient_lab_data[:, 0] > 100]

print(f"\nPatients with Glucose > 100:\n{high_glucose_patients}")
```

SAS

```
proc print data=lab_values; run;

/* SAS Aggregations (using PROC MEANS) */
proc means data=lab_values mean std min max;
    var bp_systolic bp_diastolic;
    title 'Summary Statistics for Blood Pressure (SAS)';
run;

/* SAS Filtering (WHERE statement) */
data high_systolic;
    set lab_values;
    where bp_systolic > 130;
run;
proc print data=high_systolic; run;
```

Python

```python
# --- 4. Common Aggregations ---

# Calculate summary statistics for systolic blood pressure readings
print(f"\nSummary Statistics for BP Readings ({bp_readings}):")
print(f"Mean: {np.mean(bp_readings):.2f}")
print(f"Median: {np.median(bp_readings):.2f}")
print(f"Standard Deviation: {np.std(bp_readings):.2f}")
print(f"Minimum: {np.min(bp_readings):.2f}")
print(f"Maximum: {np.max(bp_readings):.2f}")
print(f"Sum: {np.sum(bp_readings):.2f}")

# Aggregations on 2D array (e.g., mean glucose across all patients)
print(f"\nMean Glucose across all patients: {np.mean(patient_lab_data[:, 0]):.2f}")
print(f"Max Cholesterol across all patients: {np.max(patient_lab_data[:, 1]):.2f}")
```

SAS

```
/* Calculate mean, min, max, std dev using PROC MEANS
(analogous to NumPy aggregations) */
PROC MEANS DATA=mydata.bp_readings MEAN MIN MAX STD;
   VAR BP_VALUE;
RUN;

/* Perform a simple arithmetic operation (e.g., subtract a constant) in a DATA
step */
DATA mydata.bp_adjusted;
   SET mydata.bp_readings;
   BP_ADJUSTED = BP_VALUE - 10; /* Subtract 10 from each value */
RUN;

PROC PRINT DATA=mydata.bp_adjusted;
RUN;
```

Python	SAS
Explanation (Python): **import numpy as np:** The standard convention to import the NumPy library. **Creating Arrays:** np.array() is used to create arrays from Python lists. You can create 1D (vectors) or 2D (matrices) arrays. **shape and dtype:** Important attributes to understand the dimensions and data type of your array elements. **Vectorized Operations:** Notice how bp_array - 10 or bp_array ** 2 applies the operation to *every element* in the array without needing an explicit loop. This is NumPy's core strength and why it's so fast. **Slicing and Indexing:** Similar to Python lists, but more powerful for multi-dimensional arrays. bp_array[2:5] selects a range, and bp_array[bp_array > 125] uses a boolean condition to filter elements. **Aggregations:** Methods like .mean(), .std(), .min(), .max(), .sum() are directly available on NumPy arrays. For 2D arrays, axis=0 performs the operation column-wise, and axis=1 performs it row-wise	**Explanation (SAS):** **DATA step:** Used to create a simple dataset with a single variable BP_VALUE. **PROC MEANS:** A powerful procedure for calculating descriptive statistics across an entire variable (column), demonstrating SAS's inherent vectorized processing. **Arithmetic in DATA step:** Operations like BP_ADJUSTED = BP_VALUE - 10; apply to every row in the BP_VALUE column automatically, which is a form of vectorized operation in SAS.

CONCLUSION (LESSON 2.1)

- NumPy is the silent workhorse behind much of Python's data science capabilities. While you might not directly use NumPy arrays as often as Pandas DataFrames in your day-to-day clinical work, understanding its principles is crucial because Pandas DataFrames are essentially built on top of NumPy arrays. Its efficiency for numerical operations makes it indispensable for handling large clinical datasets.