18. NumPy → Pandas Bridge (LAST STEP)

1. Topic Overview

What this topic is

This topic explains how **NumPy arrays connect to Pandas objects**.
It shows how Pandas is built on top of NumPy.

Why it exists

Real data science work uses Pandas.
But Pandas internally uses NumPy arrays.
If you do not understand this bridge, debugging and performance suffer.

One simple real-world analogy

NumPy is the **engine**.
Pandas is the **car body with controls**.
You drive the car, but the engine does the real work.

2. Core Theory (Deep but Clear)

2.1 Relationship Between NumPy and Pandas

- Pandas **Series** is a 1D NumPy array with labels.
- Pandas **DataFrame** is multiple NumPy arrays combined with labels.
- Pandas stores data internally as NumPy `ndarray` or `ExtensionArray`.

Key idea:

- NumPy handles **data storage and computation**.
- Pandas handles **labels, alignment, and missing data**.

2.2 NumPy Array → Pandas Series

Internal view:

- Data is stored as a NumPy array.
- Index is stored separately.
- Dtype is still NumPy dtype.

Important:

- No data copy by default.
- Both can point to same memory.

2.3 NumPy Array → Pandas DataFrame

Internal view:

- Each column is usually a separate NumPy array.
- Columns can have different dtypes.
- Shape must match column count.

Memory note:

- Mixed dtypes force column-wise storage.
- Homogeneous data is faster.

2.4 Pandas → NumPy Conversion

Two main methods:

- `.values`
- `.to_numpy()`

Difference:

- `.to_numpy()` is safer and recommended.
- Handles missing values better.

2.5 Missing Values Handling

NumPy:

- Uses `np.nan` for floats.
- Cannot represent missing in integers.

Pandas:

- Uses `NaN`, `NaT`, or nullable dtypes.
- Converts missing data when moving to NumPy.

This conversion can change dtype.

2.6 Alignment vs Position

NumPy:

- Pure position-based.
- No labels.

Pandas:

- Label-based alignment.
- Can reorder data automatically.

When converting:

- Labels are lost.
- Only raw array remains.

3. Syntax & Examples

3.1 NumPy Array → Pandas Series

Basic Syntax

```
pd.Series(array, index=None)
```

Example 1

```
import numpy as np
import pandas as pd

arr = np.array([10, 20, 30])
s = pd.Series(arr)

print(s)
```

Output

```
0    10
1    20
2    30
dtype: int64
```

Explanation:

- Data comes from NumPy array.
- Index is auto-generated.
- dtype is inherited.

Example 2

```
s = pd.Series(arr, index=['a', 'b', 'c'])
print(s)
```

Output:

```
a    10
b    20
c    30
dtype: int64
```

Explanation:

- Same NumPy data.
- Custom labels added.
- Memory is still shared.

3.2 NumPy Array → Pandas DataFrame

Basic Syntax

```
pd.DataFrame(array, columns=None)
```

Example 1

```
arr = np.array([[1, 2], [3, 4], [5, 6]])
df = pd.DataFrame(arr, columns=['A', 'B'])

print(df)
```

Output:

```
   A  B
0  1  2
1  3  4
2  5  6
```

Explanation:

- 2D NumPy array.
- Each column is a NumPy array.
- Shape must match.

Example 2

```
print(df.dtypes)
```

Output:

```
A    int64
B    int64
dtype: object
```

Explanation:

- Each column keeps NumPy dtype.
- Stored column-wise.

3.3 Pandas → NumPy

Basic Syntax

```
df.to_numpy()
```

Example 1

```
arr_back = df.to_numpy()
print(arr_back)
```

Output:

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

Explanation:

- Labels removed.
- Pure NumPy array returned.

Example 2 (Missing Values)

```
df2 = pd.DataFrame({'A': [1, 2, None]})
print(df2.to_numpy())
```

Output:

```
[[ 1.]
 [ 2.]
 [nan]]
```

Explanation:

- Integer column becomes float.
- NumPy cannot hold missing int.

3.4 Memory Sharing Check

```
arr = np.array([1, 2, 3])
s = pd.Series(arr)

arr[0] = 100
print(s)
```

Output:

```
0    100
1      2
2      3
dtype: int64
```

Explanation:

- Same memory.
- Changes reflect both sides.

4. Why This Matters in Data Science

Data Cleaning

- Pandas cleaning uses NumPy operations.
- Wrong dtype causes slow cleaning.

Feature Engineering

- Feature arrays sent to ML models are NumPy.
- Pandas columns become NumPy matrices.

Model Input Preparation

- Scikit-learn expects NumPy arrays.
- Shape mismatch causes training failure.

ML / DL Pipelines

- TensorFlow and PyTorch require NumPy-like arrays.
- Bad conversion leads to silent bugs.

What breaks if you don’t know this

- Unexpected dtype changes.
- Memory leaks.
- Slow pipelines.
- Wrong model inputs.

5. Common Mistakes (VERY IMPORTANT)

1. Assuming Pandas does not use NumPy
   Why: abstraction hides internals
   Avoid: always check `.to_numpy()`
2. Ignoring dtype changes during conversion
   Why: missing values
   Avoid: print `dtype` before and after

3. Assuming labels exist after conversion
   Why: NumPy has no index
   Avoid: save index separately
4. Modifying NumPy array without realizing Pandas changes
   Why: shared memory
   Avoid: use `.copy()`
5. Passing DataFrame directly to ML models blindly
   Why: shape or dtype issues
   Avoid: explicitly convert and validate

6. Performance & Best Practices

When this is fast

- Homogeneous numeric data
- Direct NumPy operations
- Minimal conversions

When this is slow

- Mixed dtypes
- Frequent back-and-forth conversion
- Large object dtype columns

Warnings

- `.values` can behave inconsistently
- Use `.to_numpy()` always
- Watch memory duplication with `.copy()`

7. 20 Practice Problems (MANDATORY)

Easy (5)

1. Convert a NumPy array to Series with custom index.

2. Create DataFrame from a 2D NumPy array.
3. Convert Series back to NumPy array.
4. Check dtype before and after conversion.
5. Verify memory sharing using modification.

Medium (7)

6. Convert DataFrame with missing values to NumPy.
7. Handle dtype change after conversion.
8. Stack two NumPy arrays into DataFrame.
9. Convert labeled data to NumPy for ML input.
10. Detect shape mismatch errors.
11. Compare `.values` vs `.to_numpy()`.
12. Preserve index during conversion.

Hard (5)

13. Optimize conversion for large numeric dataset.
14. Debug silent dtype promotion.
15. Identify memory copy vs view.
16. Prepare NumPy input for sklearn pipeline.
17. Fix performance issue caused by object dtype.

Industry-Level Tasks (3)

18. Build feature matrix from Pandas for ML training.
19. Clean Pandas data and export NumPy arrays safely.
20. Debug model bug caused by Pandas to NumPy conversion.

8. Mini Checklist

- Pandas is built on NumPy
- Data lives in NumPy arrays
- Conversion can change dtype
- Labels are lost in NumPy
- Memory can be shared
- Use `.to_numpy()`

- Always check shape and dtype before ML