5. Boolean Masking & Filtering (NumPy)

1. Topic Overview

What this topic is

Boolean masking is selecting elements from a NumPy array using conditions that return `True` or `False`.

Why it exists

Data Science data is messy.

You must remove bad values, keep valid rows, and filter data fast without loops.

One real-world analogy

Think of a security gate.

Only people with a valid badge can enter.

The badge check is the condition.

The gate is the mask.

2. Core Theory (Deep but Clear)

How NumPy thinks internally

- NumPy arrays store data in **continuous memory**
- Each element has a **dtype**
- Conditions create a **boolean array**
- Boolean arrays have the same **shape** as the original array
- `True` means keep
- `False` means drop

Sub-Topic 1: Conditions (`> < == !=`)

What happens internally

- NumPy compares each element
- Result is a boolean array
- dtype becomes `bool`

Example logic:

```
[10, 20, 30] > 15
→ [False, True, True]
```

Sub-Topic 2: Multiple Conditions (`& |`)

Important rule

NumPy does **element-wise logic**, not Python logic.

- Use `&` for AND
- Use `|` for OR
- Always use parentheses

Why:

- Python `and/or` work on single values
- NumPy needs vectorized operations

Sub-Topic 3: Filtering Arrays Using Conditions

What filtering really means

- Boolean mask is applied to the array
- Only `True` positions are selected
- Output is a **new array**
- Shape usually changes
- Memory is copied

3. Syntax & Examples

A. Conditions (`>` `<` `==` `!=`)

Basic Syntax

```
mask = array > value
```

Example 1

```
import numpy as np

x = np.array([10, 25, 30, 5])
mask = x > 20
print(mask)
```

Output

```
[False  True  True False]
```

Explanation

- 10 > 20 → False
- 25 > 20 → True
- 30 > 20 → True
- 5 > 20 → False

Example 2

```
print(x == 25)
```

Output

```
[False  True False False]
```

Example 3

```
print(x != 10)
```

Output

```
[False  True  True  True]
```

B. Multiple Conditions (& |)

Basic Syntax

```
mask = (array > a) & (array < b)
```

Example 1

```
mask = (x > 10) & (x < 30)
print(mask)
```

Output

```
[False  True False False]
```

Explanation

- Only 25 satisfies both conditions

Example 2

```
mask = (x < 10) | (x > 25)
print(mask)
```

Output

```
[False False  True  True]
```

Example 3

```
mask = (x == 10) | (x == 30)
print(mask)
```

Output

```
[ True False  True False]
```

C. Filtering Arrays Using Conditions

Basic Syntax

```
filtered = array[mask]
```

Example 1

```
filtered = x[x > 20]
print(filtered)
```

Output

```
[25 30]
```

Explanation

- Mask is `[False True True False]`
- Only `True` positions are selected

Example 2

```
filtered = x[(x > 10) & (x < 30)]
print(filtered)
```

Output

```
[25]
```

Example 3 (2D Array)

```
y = np.array([[1, 20], [30, 4]])
print(y[y > 10])
```

Output

```
[20 30]
```

Important

- Output becomes **1D**
- Shape is lost

4. Why This Matters in Data Science

Data Cleaning

- Remove negative values
- Remove invalid sensor readings
- Drop corrupted rows

Example:

```
data = data[data >= 0]
```

Feature Engineering

- Create filtered subsets
- Apply rules on numeric features
- Select ranges

Model Input Preparation

- Remove NaNs using conditions
- Remove outliers
- Ensure valid ranges

ML / DL Pipelines

- Batch filtering
- Mask invalid samples
- Apply conditional preprocessing

What breaks if you don’t know this

- Wrong training data
- Shape mismatch errors
- Silent bugs
- Bad model accuracy
- Slow Python loops

5. Common Mistakes (VERY IMPORTANT)

1. **Using `and` instead of `&`**
   Causes ValueError
   Always use `&`
2. **Forgetting parentheses**
   Operator precedence breaks logic
   Always wrap each condition
3. **Mask shape mismatch**
   Mask must match array shape
   Check `mask.shape`
4. **Expecting original shape after filtering**
   Filtering changes size
   Do not assume same length
5. **Filtering 2D arrays and losing structure**
   Boolean filtering flattens output
   Use row-wise masking carefully

6. Performance & Best Practices

When it is fast

- Large arrays
- Vectorized conditions
- No Python loops

When it is slow

- Repeated filtering inside loops
- Chained filtering creating copies

Memory Warnings

- Filtering creates a **new array**
- Large datasets can double memory
- Prefer in-place logic when possible

Best Practices

- Combine conditions in one mask
- Reuse masks
- Print mask shape during debugging

7. Practice Problems (NO SOLUTIONS)

Easy (5)

1. Filter values greater than 50 from a 1D array
2. Select values equal to zero
3. Remove negative numbers
4. Create a boolean mask for even numbers
5. Count how many values are greater than 100

Medium (7)

6. Filter salaries between 30k and 80k
7. Remove rows where age < 18
8. Select values outside a given range
9. Filter a 2D array using a condition
10. Create a mask for NaN values
11. Keep only positive floating values
12. Filter using three conditions combined

Hard (5)

13. Remove outliers using percentile limits
14. Filter rows where any value is negative
15. Create class-wise masks for labels
16. Filter samples based on feature threshold
17. Apply mask and keep original shape logic

Industry-Level Tasks (3)

18. Clean sensor data with invalid ranges
19. Remove corrupted samples before training
20. Filter real-time streaming data efficiently

8. Mini Checklist

- Conditions return boolean arrays
- Mask shape must match data shape
- Use `& |` , never `and/or`
- Parentheses are mandatory
- Filtering creates new arrays
- 2D filtering flattens output
- Used everywhere in data cleaning

Advanced Boolean Masking & Filtering Techniques (NumPy)

These are **part of the same topic**.
They are **not optional** if you want to work like a data scientist.

1. Topic Overview

What this topic is

These are advanced Boolean-based filtering tools built on top of boolean masks.

Why they exist

Real datasets have:

- Missing values
- Infinite values
- Category filtering
- Conditional replacement
- Complex logic

Basic comparisons are not enough.

One real-world analogy

Think of a factory quality check.
Some items are rejected.
Some are repaired.
Some are ignored.
Different rules. Same inspection system.

2. Core Theory (Deep but Clear)

How NumPy thinks internally

- All these methods return **boolean masks**
- Masks must match array shape
- Operations are vectorized
- Output depends on:
  - shape
  - dtype
  - memory copy vs view

Technique 1: `np.where()` (Conditional Selection)

What it does

- Applies an `if-else` logic element-wise
- Can:
  - Filter values
  - Replace values
  - Create new arrays

Technique 2: `np.isin()` (Category Filtering)

What it does

- Checks if elements exist in a given list
- Used for categorical filtering

Technique 3: `np.isnan()` and `np.isfinite()`

What they do

- Detect missing values
- Detect infinite values
- Mandatory for numeric datasets

Technique 4: `np.logical_and()`, `np.logical_or()`, `np.logical_not()`

What they do

- Safe logical operations

- Used when conditions get complex
- Cleaner than `&` `|`

Technique 5: Boolean Mask Assignment (In-place Filtering)

What it does

- Modifies array values using masks
- No new array created
- Memory efficient

3. Syntax & Examples

1. `np.where()`

Basic Syntax

```
np.where(condition, value_if_true, value_if_false)
```

Example 1

```
import numpy as np

x = np.array([10, 25, 5, 40])
y = np.where(x > 20, x, 0)
print(y)
```

Output

```
[ 0 25  0 40]
```

Explanation

- Values > 20 kept
- Others replaced with 0

Example 2

```
y = np.where(x < 10, -1, x)
print(y)
```

Output

```
[10 25 -1 40]
```

Example 3

```
idx = np.where(x > 20)
print(idx)
```

Output

```
(array([1, 3]),)
```

2. `np.isin()`

Basic Syntax

```
np.isin(array, values)
```

Example 1

```
x = np.array([1, 2, 3, 4, 5])
mask = np.isin(x, [2, 4])
print(mask)
```

Output

```
[False  True False  True False]
```

Example 2

```
print(x[mask])
```

Output

```
[2 4]
```

Example 3 (Categories)

```
labels = np.array(['A', 'B', 'C', 'A'])
print(np.isin(labels, ['A', 'C']))
```

Output

```
[ True False  True  True]
```

3. `np.isnan()` and `np.isfinite()`

Basic Syntax

```
np.isnan(array)
np.isfinite(array)
```

Example 1

```
x = np.array([1.0, np.nan, 3.0])
print(np.isnan(x))
```

Output

```
[False  True False]
```

Example 2

```
print(x[~np.isnan(x)])
```

Output

```
[1. 3.]
```

Example 3

```
x = np.array([1, np.inf, -np.inf, 5])
print(np.isfinite(x))
```

Output

```
[ True False False  True]
```

4. `np.logical_and()` / `np.logical_or()`

Basic Syntax

```
np.logical_and(cond1, cond2)
```

Example 1

```
x = np.array([10, 20, 30])
mask = np.logical_and(x > 10, x < 30)
print(mask)
```

Output

```
[False  True False]
```

Example 2

```
mask = np.logical_or(x < 15, x > 25)
print(mask)
```

Output

```
[ True False  True]
```

Example 3

```
print(np.logical_not(x > 20))
```

Output

```
[ True  True False]
```

5. Boolean Mask Assignment (In-place)

Basic Syntax

```
array[condition] = value
```

Example 1

```
x = np.array([10, -5, 30, -2])
x[x < 0] = 0
print(x)
```

Output

```
[10  0 30  0]
```

Example 2

```
x[x > 20] = 100
print(x)
```

Output

```
[ 10   0 100   0]
```

Example 3

```
x[np.isnan(x)] = 0
```

4. Why This Matters in Data Science

Data Cleaning

- Remove NaN
- Replace invalid values
- Filter categories

Feature Engineering

- Conditional feature creation
- Bucketization
- Label filtering

Model Input Preparation

- Clean arrays before feeding models
- Prevent NaN crashes
- Stable training

ML / DL Pipelines

- Mask samples

- Apply rules per batch
- Fast preprocessing

What breaks if you skip this

- Model training fails
- Silent NaN propagation
- Wrong labels
- Memory waste
- Debugging hell

5. Common Mistakes

1. Using Python `if` instead of vectorized logic
2. Forgetting `~` for negation
3. Ignoring NaN before filtering
4. Creating many temporary arrays
5. Modifying arrays unintentionally in-place

6. Performance & Best Practices

Fast

- Vectorized masks
- In-place assignment
- Single-pass logic

Slow

- Nested conditions
- Python loops

- Multiple `where` chains

Best Practices

- Use `np.isfinite()` before training
- Combine masks
- Prefer in-place when safe

7. Practice Problems (NO SOLUTIONS)

Easy (5)

1. Replace negative values with zero
2. Filter values present in a list
3. Remove NaN values
4. Mark values above threshold as 1 else 0
5. Count finite values

Medium (7)

6. Replace outliers with median
7. Filter dataset by allowed categories
8. Clean sensor values using limits
9. Remove infinite values
10. Conditional scaling using `where`
11. Multi-condition filtering using logical ops
12. Create binary target using threshold

Hard (5)

13. Remove rows with any NaN
14. Clip values using boolean masks

15. Conditional feature generation
16. Mask invalid samples in batch
17. Combine numeric and categorical masks

Industry-Level Tasks (3)

18. Clean financial time-series data
19. Prepare ML-ready dataset from raw array
20. Build preprocessing pipeline using masks

8. Mini Checklist

- `np.where()` is vectorized if-else
- `np.isin()` for category filtering
- Always handle NaN and Inf
- Logical functions are safer than `& |`
- In-place masking saves memory
- Masks control everything in preprocessing

Reality check

If you master these techniques, you stop writing loops.
If you skip them, you will struggle in real projects.