8. Aggregation & Statistics (NumPy)

1. Topic Overview

What this topic is

Aggregation means **reducing many values into fewer values**.
Statistics means **summarizing numeric data** using math functions.

In NumPy, aggregation and statistics work on **arrays** and often **reduce dimensions**.

Why it exists

Raw data is large.
Models and analysis need **summaries**, not raw numbers.

One real-world analogy

You have marks of 100 students.
Instead of reading all marks, you calculate **average, max, min**.

2. Core Theory (Deep but Clear)

Key idea

Aggregation functions:

- Take an array
- Apply an operation
- Return **one value or a smaller array**

Important concept: Axis

NumPy arrays have dimensions.

- 1D array → only one direction
- 2D array → rows and columns

`axis` tells NumPy **which direction to reduce**.

- `axis=0` → reduce rows (column-wise)
- `axis=1` → reduce columns (row-wise)

Internal NumPy thinking

NumPy stores:

- Data in **continuous memory**
- With fixed **dtype**
- With a defined **shape**

Aggregation:

- Loops in C (fast)
- Reads memory sequentially
- Produces new array with reduced shape

3. Syntax & Examples

We will cover these sub-topics:

- sum
- mean
- min / max
- std / var
- argmin / argmax
- cumulative functions
- nan-aware functions

A. `np.sum()`

Syntax

```
np.sum(array, axis=None)
```

Example 1: 1D

```
import numpy as np

a = np.array([1, 2, 3, 4])
print(np.sum(a))
```

Output

```
10
```

Explanation

- Adds all elements
- Returns a single number

Example 2: 2D with axis

```
b = np.array([[1, 2, 3],
              [4, 5, 6]])

print(np.sum(b, axis=0))
print(np.sum(b, axis=1))
```

Output

```
[5 7 9]
[ 6 15]
```

Explanation

- `axis=0` : column-wise sum
- `axis=1` : row-wise sum

B. `np.mean()`

Syntax

```
np.mean(array, axis=None)
```

Example

```
x = np.array([10, 20, 30])
print(np.mean(x))
```

Output

```
20.0
```

Explanation

- Sum divided by count
- Always returns float

C. `np.min()` and `np.max()`

Syntax

```
np.min(array, axis=None)
np.max(array, axis=None)
```

Example

```
y = np.array([[3, 7, 2],
              [8, 1, 5]])

print(np.min(y))
print(np.max(y))
```

Output

```
1
8
```

Explanation

- Finds smallest and largest value

D. `np.std()` and `np.var()`

Syntax

```
np.std(array, axis=None)
np.var(array, axis=None)
```

Example

```
z = np.array([2, 4, 6, 8])
print(np.var(z))
print(np.std(z))
```

Output

```
5.0
2.23606797749979
```

Explanation

- Variance measures spread

- Std is square root of variance

E. `np.argmin()` and `np.argmax()`

Syntax

```
np.argmin(array, axis=None)
np.argmax(array, axis=None)
```

Example

```
p = np.array([10, 5, 20])
print(np.argmin(p))
print(np.argmax(p))
```

Output

```
1
2
```

Explanation

- Returns index, not value

F. Cumulative functions

Functions

- `np.cumsum()`
- `np.cumprod()`

Example

```
q = np.array([1, 2, 3, 4])
print(np.cumsum(q))
```

Output

```
[ 1  3  6 10]
```

Explanation

- Running total
- Shape stays same

G. NaN-aware functions

Problem

Normal functions break with NaN.

Solution

Use:

- `np.nanmean`
- `np.nansum`
- `np.nanstd`

Example

```
r = np.array([1, 2, np.nan, 4])
print(np.mean(r))
print(np.nanmean(r))
```

Output

```
nan
2.3333333333333335
```

Explanation

- `mean` fails
- `nanmean` ignores NaN

4. Why This Matters in Data Science

Data cleaning

- Handle missing values using `nanmean`
- Detect outliers using min/max

Feature engineering

- Aggregated features
- Rolling statistics
- Group summaries before modeling

Model input preparation

- Normalize using mean and std
- Check feature ranges

ML / DL pipelines

- Loss monitoring
- Batch statistics
- Feature scaling

What breaks if you don't understand this

- Wrong axis → wrong results
- NaN contamination
- Silent data leakage
- Bad model performance

5. Common Mistakes (VERY IMPORTANT)

1. Confusing axis direction
    - `axis=0` vs `axis=1`
    - Always draw array shape

2. Ignoring NaN values
   - Using `mean` instead of `nanmean`
   - Leads to NaN outputs
3. Expecting same shape after aggregation
   - Aggregation reduces dimensions
4. Using Python loops instead of NumPy
   - Slow and error-prone
5. Assuming integer output
   - Mean and std return float

6. Performance & Best Practices

When it is fast

- Large arrays
- Contiguous memory
- Vectorized calls

When it is slow

- Python loops
- Repeated aggregations inside loops

Warnings

- Large arrays increase memory usage
- Casting dtype can change results
- Be careful with float precision

7. 20 Practice Problems

Easy (5)

1. Compute sum of a 1D array
2. Find mean of a 2D array column-wise

3. Get max value from an array
4. Find index of minimum value
5. Compute cumulative sum

Medium (7)

6. Normalize a feature using mean and std
7. Compute row-wise averages
8. Handle missing values using NaN-safe functions
9. Find feature with highest variance
10. Compute per-column min and max
11. Compare mean vs median for skewed data
12. Detect constant features using variance

Hard (5)

13. Standardize a dataset manually using NumPy
14. Compute batch statistics for ML training
15. Identify outlier rows using std thresholds
16. Aggregate time-series data
17. Debug wrong axis aggregation in a pipeline

Industry-Level Tasks (3)

18. Build feature summary for a dataset
19. Clean sensor data with missing readings
20. Prepare normalized input for a neural network

8. Mini Checklist

- Aggregation reduces dimensions
- Axis decides direction

- Mean and std return float
- Use NaN-safe functions
- Never guess axis, verify shape
- Aggregation is everywhere in ML

8. Aggregation & Statistics (Advanced but Mandatory)

These are **not optional**. Learn them properly.

1. Topic Overview

What this part is

These are **advanced aggregation and statistical helpers** built on top of basic mean and sum.

They help you:

- summarize distributions
- rank values
- compare samples
- reduce noise

Why it exists

Real data is:

- noisy
- skewed
- incomplete

Mean alone is not enough.

One real-world analogy

Average salary hides reality.
Median and percentiles show the truth.

2. Core Theory (Deep but Clear)

We cover these **mandatory sub-topics**:

1. `median`
2. `percentile / quantile`
3. `ptp` (range)
4. `average` (weighted mean)
5. `corrcoef`
6. `cov`
7. `unique + counts`
8. `any / all`

3. Syntax & Examples

A. `np.median()`

What it does

Finds the **middle value** after sorting.

Syntax

```
np.median(array, axis=None)
```

Example

```
import numpy as np

a = np.array([1, 100, 2, 3])
print(np.median(a))
```

Output

```
2.5
```

Explanation

- Sorted array → [1, 2, 3, 100]
- Middle average → (2 + 3) / 2

B. `np.percentile()` and `np.quantile()`

What it does

Shows **distribution position**.

- Percentile → 0 to 100
- Quantile → 0 to 1

Syntax

```
np.percentile(array, q)
np.quantile(array, q)
```

Example

```
b = np.array([10, 20, 30, 40, 50])

print(np.percentile(b, 25))
print(np.quantile(b, 0.5))
```

Output

```
20.0
30.0
```

Explanation

- 25th percentile → lower quarter
- 0.5 quantile → median

C. `np.ptp()` (Peak to Peak)

What it does

Computes **range** = max − min

Syntax

```
np.ptp(array)
```

Example

```
c = np.array([3, 10, 7])
print(np.ptp(c))
```

Output

```
7
```

Explanation

- Max = 10
- Min = 3
- Range = 7

D. `np.average()` (Weighted Mean)

What it does

Mean with **weights**.

Syntax

```
np.average(array, weights=weights_array)
```

Example

```
scores = np.array([80, 90, 100])
weights = np.array([1, 2, 3])

print(np.average(scores, weights=weights))
```

Output

```
93.33333333333333
```

Explanation

- Higher weight → more importance
- Used in scoring systems

E. `np.cov()` (Covariance)

What it does

Measures **how two variables move together**.

Syntax

```
np.cov(x, y)
```

Example

```
x = np.array([1, 2, 3])
y = np.array([2, 4, 6])

print(np.cov(x, y))
```

Output

```
[[1. 2.]
 [2. 4.]]
```

Explanation

- Positive covariance → same direction
- Used in PCA and statistics

F. `np.corrcoef()` (Correlation)

What it does

Normalized covariance (range −1 to 1).

Syntax

```
np.corrcoef(x, y)
```

Example

```
print(np.corrcoef(x, y))
```

Output

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

Explanation

- 1 → perfect positive relation
- 0 → no relation

G. `np.unique()` with counts

What it does

Finds **unique values and frequency**.

Syntax

```
np.unique(array, return_counts=True)
```

Example

```
d = np.array([1, 1, 2, 3, 3, 3])
vals, counts = np.unique(d, return_counts=True)

print(vals)
print(counts)
```

Output

```
[1 2 3]
[2 1 3]
```

Explanation

- Used for class balance checks

H. `np.any()` and `np.all()`

What it does

Logical aggregation.

Syntax

```
np.any(condition)
np.all(condition)
```

Example

```
e = np.array([True, False, True])

print(np.any(e))
print(np.all(e))
```

Output

```
True
False
```

Explanation

- `any` → at least one True
- `all` → all must be True

4. Why This Matters in Data Science

Data cleaning

- Detect skew using median
- Find class imbalance using unique counts

Feature engineering

- Percentile-based features
- Weighted features
- Distribution-aware scaling

Model input preparation

- Remove low-variance features
- Handle outliers using percentiles

ML / DL pipelines

- PCA uses covariance
- Correlation for feature selection
- Dataset validation using any/all

What breaks if you don’t learn this

- Wrong assumptions about data
- Poor feature quality
- Unstable models
- Interview failure

5. Common Mistakes (VERY IMPORTANT)

1. Using mean instead of median for skewed data
2. Misinterpreting correlation as causation
3. Forgetting weights sum effect
4. Ignoring class imbalance
5. Using range instead of std blindly

6. Performance & Best Practices

Fast

- Vectorized operations
- Large numeric arrays

Slow

- Python loops
- Repeated percentile calls

Warnings

- Correlation is sensitive to outliers
- Percentiles on small data are unstable
- Use float dtype for statistics

7. 20 Practice Problems

Easy (5)

1. Find median of an array
2. Compute 75th percentile
3. Find range using ptp
4. Check if any value is negative
5. Count unique values

Medium (7)

6. Detect skew using mean vs median
7. Compute weighted average score
8. Find low-variance features
9. Identify majority class
10. Check correlation between features
11. Filter rows using any/all
12. Compare distributions using percentiles

Hard (5)

13. Build percentile-based normalization

14. Remove outliers using IQR
15. Feature selection using correlation
16. Prepare PCA input using covariance
17. Debug wrong correlation results

Industry-Level Tasks (3)

18. Validate dataset before training
19. Detect sensor drift using statistics
20. Build feature summary report

8. Mini Checklist

- Mean is not enough
- Median handles skew
- Percentiles show distribution
- Correlation ≠ causation
- Covariance used in PCA
- Unique counts matter for classes
- Aggregation drives ML quality