# 📊 Pandas DataFrame Attributes

### 1. Shape & Size

* df.shape → Returns (rows, columns).
* df.size → Total number of elements (rows × columns).
* df.ndim → Number of dimensions (2 for DataFrame).

### 2. Labels

* df.index → Row index labels.
* df.columns → Column labels.
* df.axes → List of row & column labels.

### 3. Values

* df.values → Numpy array of data.
* df.dtypes → Data types of each column.
* df.T → Transpose (rows ↔ columns).

### 4. Metadata

* df.empty → Checks if DataFrame is empty (True/False).

**Slicing in Pandas**

# 📊 Slicing in Pandas

Selecting specific **rows or columns** from a DataFrame using indexes or labels.

### 1. Row Slicing (by position using iloc)

**df.iloc[0]** # First row

**df.iloc[0:3]** # First 3 rows

**df.iloc[::2]**  # Every 2nd row

### 2. Row Slicing (by label using loc)

**df.loc[0]**  # Row with index label 0

**df.loc[1:3]** # Rows from index 1 to 3 (inclusive)

### 3. Column Slicing

**df.iloc[:, 0]**  # First column

**df.iloc[:, 0:2]** # First 2 columns

**df.loc[:, "A":"C"]**  # Columns from A to C

### 4. Row & Column together

**df.iloc[0:3, 0:2]**  # First 3 rows & 2 columns

**df.loc[0:2, ["A","B"]]**# Rows 0-2, columns A & B

# 📊 Filtering Data in Pandas

Extracting rows that satisfy a **condition**.

### 1. Filter by Single Condition

**df[df["Age"] > 25]**  # Rows where Age > 25

**df[df["City"] == "Delhi"]**# Rows where City is Delhi

### 2. Filter by Multiple Conditions

**df[(df["Age"] > 25) & (df["Salary"] > 40000)]**  # AND

**df[(df["Age"] < 30) | (df["City"] == "Mumbai")**] # OR

**df[~(df["City"] == "Delhi")]**  # NOT

# 📊 Vector Operations on DataFrames

Performing operations on entire columns **at once without loops**.

This makes Pandas faster and more efficient.

### 1. Arithmetic Operations (Element-wise)

Operations apply to all elements in the DataFrame.

**import pandas as pd**

**df1 = pd.DataFrame({"A": [10, 20, 30], "B": [40, 50, 60]})**

**df2 = pd.DataFrame({"A": [1, 2, 3], "B": [4, 5, 6]})**

**print(df1 + df2) # Addition**

**print(df1 - df2) # Subtraction**

**print(df1 \* 2) # Scalar multiplication**

**print(df1 / df2) # Division**

✅ Output:

**A B**

**0 11 44**

**1 22 55**

**2 33 66**

### 2. Column-wise Operations

Performing operations between columns.

**df1["C"] = df1["A"] + df1["B"]**

**print(df1)**

✅ Output:

**A B C**

**0 10 40 50**

**1 20 50 70**

**2 30 60 90**

### 3. Comparison Operations

Element-wise boolean results.

**print(df1 > 25)**

✅ Output:

**A B C**

**0 False True True**

**1 False True True**

**2 True True True**

# 📊 Aggregating Functions in Pandas

Functions that perform a calculation on a set of values and return a **single summary value**.

### 🔹 1. What is Aggregation?

* Aggregation = applying a **summary function** to a column or group of data.
* Reduces data → gives **single value/statistics** like sum, mean, count.

### 🔹 2. Common Aggregation Functions

| **Function** | **Use** |
| --- | --- |
| sum() | Sum of values |
| mean() | Average value |
| median() | Middle value |
| min() | Minimum |
| max() | Maximum |
| count() | Number of non-null values |
| nunique() | Count of unique values |
|  |  |
|  |  |
|  |  |

### 🔹 3. Examples

**import pandas as pd**

**df = pd.DataFrame({**

**"Name": ["Alice", "Bob", "Charlie", "David"],**

**"Age": [24, 27, 22, 32],**

**"Salary": [50000, 60000, 55000, 65000]**

**})**

**print(df["Age"].mean()) # Average age**

**print(df["Salary"].sum()) # Total salary**

**print(df.agg({"Age": ["min", "max"], "Salary": ["mean", "std"]}))**

✅ Output:

**26.25**

**230000**

**Age Salary**

**min 22.0 NaN**

**max 32.0 NaN**

**NaN NaN 57500.000000**

**NaN NaN 6454.972243**

### 🔹 4. Group-wise Aggregation

Splitting data into **groups based on one or more columns**, then applying **aggregate functions** on each group.

Use groupby() with aggregation.

**df = pd.DataFrame({**

**"Dept": ["IT", "IT", "HR", "HR"],**

**"Salary": [50000, 60000, 55000, 65000]**

**})**

**print(df.groupby("Dept")["Salary"].mean())**

✅ Output:

**Dept**

**HR 60000**

**IT 55000**

**Name: Salary, dtype: int64**