OBJECT ORIENTED PROGRAMMING

Project OOP and Pandas

SUBMITTED BY:

Rama Kamal (DS-051/2024-25)

Shanzay Khan (DS-056/2025)

1. Object-Oriented Programming (OOP) Principles

Encapsulation

Definition: Encapsulation is the bundling of data and the methods that operate on that data within a class, while restricting direct access to some of the object's components. It protects the internal state of an object and enforces a controlled interface for interaction.

- **Real-World Analogy:** An ATM allows you to withdraw cash, check your balance, or deposit money through a simple interface, but you cannot directly access the bank's internal database or cash storage system.

- **Data Science Analogy:** In pandas, a `DataFrame` encapsulates a 2D data structure along with metadata and internal methods, but users interact with it using methods like `.loc[]` and `.groupby()` instead of manipulating internal arrays directly.

Abstraction

Definition: Abstraction hides complex internal implementation details and exposes only the necessary parts through a simplified interface. It reduces cognitive load for users.

- **Real-World Analogy:** You can drive a car using the steering wheel and pedals without knowing how the engine or transmission works.

- **Data Science Analogy:** Calling `.plot()` on a `Series` or `DataFrame` allows visualization without needing to understand how matplotlib or seaborn is used internally.

Inheritance

Definition: Inheritance allows a class (child or derived class) to inherit attributes and methods from another class (parent or base class). This promotes code reuse and consistency.

- **Real-World Analogy:** A "savings account" inherits the properties of a general "bank account" but may add features like interest accumulation.

- **Data Science Analogy:** `DataFrame` and `Series` both inherit from `NDFrame` in pandas, which provides shared functionality like `.copy()`, `.shape`, and `.head()`.

Polymorphism

Definition: Polymorphism allows methods to have the same name but behave differently depending on the object they are acting on. This enables flexibility and generalized code.

- **Real-World Analogy:** The "start" button functions differently in a microwave, a computer, and a car.
- **Data Science Analogy:** The `.sum()` method behaves differently in `Series` (returns a single scalar) versus `DataFrame` (returns a `Series` of column-wise sums).

2. Detailed Class Analysis

Class: `DataFrame`

- **Encapsulation:**
  - Internally, a `DataFrame` stores data in memory-efficient blocks (`BlockManager`), manages labels with `Index` objects, and uses optimized internal routines for performance.
  - However, end-users don’t need to handle these. Instead, they use intuitive interfaces like `.iloc[]`, `.loc[]`, `.at[]`, and `.groupby()`.
- **Abstraction:**
  - Complex operations like pivoting, joining, or calculating statistics are abstracted into one-line commands like `.pivot_table()`, `.merge()`, and `.describe()`, hiding the underlying logic.
- **Inheritance:**
  - Inherits from `NDFrame`, which provides methods such as `.shape`, `.copy()`, and `.head()`, ensuring consistent interfaces with `Series`.
- **Polymorphism Example:**
  - `.sum()` on a `DataFrame` returns column-wise totals.
  - `.plot()` on a `DataFrame` shows multiple lines or bars based on all columns.
- **Shared Behavior:**

    - Shares many methods with `Series` due to common base class, including alignment logic, type checking, and metadata handling.

Class: `Series`

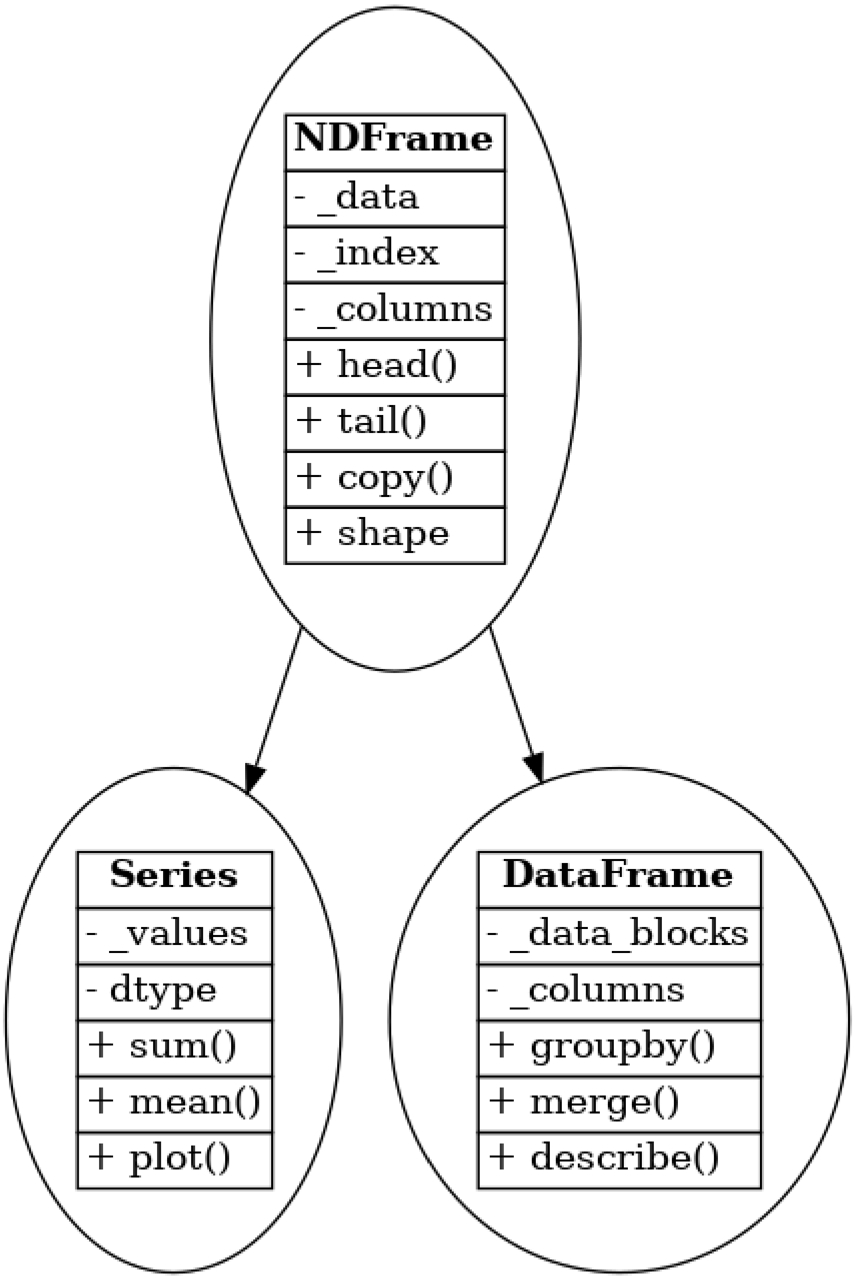
- **Encapsulation:**
    - Internally stores values in a 1D NumPy array and uses an `Index` object for labeling. Users access data using `.iloc[]`, `.iat[]`, or `.loc[]` without needing to manipulate the raw array.
- **Abstraction:**
    - Statistical functions like `.mean()`, `.std()`, `.value_counts()` simplify data analysis by abstracting away NumPy or manual computation logic.
- **Inheritance:**
    - Also inherits from `NDFrame`. This gives it consistency with `DataFrame` for methods like `.head()`, `.copy()`, and `.astype()`.
- **Polymorphism Example:**
    - `.sum()` on a `Series` returns a single value.
    - `.plot()` on a `Series` defaults to a simple line graph.
- **Shared Behavior:**
    - Inherits indexing, data alignment, missing value handling, and I/O methods from `NDFrame`, making it fully interoperable with `DataFrame`.

3. OOP Pillar Mapping Table

Class	Encapsulation	Abstraction	Inheritance	Polymorphism Example
`DataFrame`	Hides data in block structures	One-liner methods for complex operations	Inherits from `NDFrame`	`.sum()` returns column-wise totals
`Series`	Hides internal array and index	Exposes statistical and indexing tools	Inherits from `NDFrame`	`.sum()` returns a single scalar

4. UML Class Diagram

This diagram shows the inheritance relationship between the `NDFrame` base class and the `Series` and `DataFrame` derived classes.


NDFrame
- _data
- _index
- _columns
+ head()
+ tail()
+ copy()
+ shape
Series
- _values
- dtype
+ sum()
+ mean()
+ plot()
DataFrame
- _data_blocks
- _columns
+ groupby()
+ merge()
+ describe()