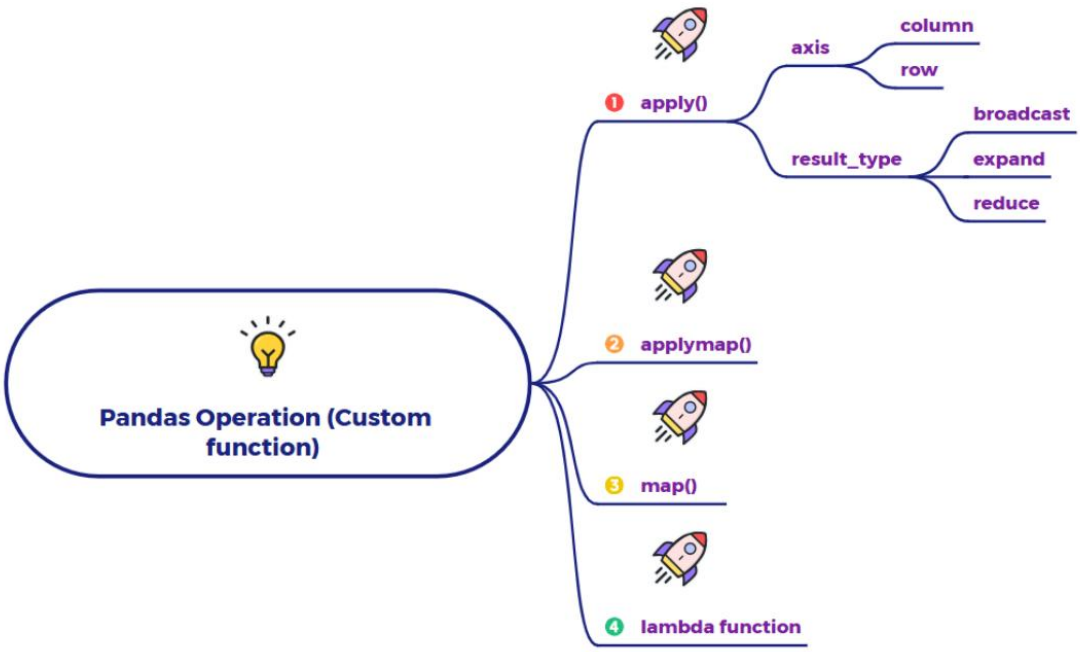
How to apply Custom functions in Numerical Variables?

Custom functions in Pandas allow you to extend the library's capabilities by applying user-defined logic to your numerical (or any) data. While Pandas offers a wide array of built-in operations for common tasks, custom functions provide the flexibility to perform highly specific or complex transformations and calculations that aren't natively available.



Purpose of Custom Functions

The primary **purpose** of using custom functions in Pandas is to **apply unique, specialized, or complex logic to your data** that goes beyond standard operations. This allows you to:

- Implement intricate business rules or domain-specific calculations.
- Perform multi-step transformations that combine various operations.
- Create new features based on complex relationships between existing columns.
- Clean or validate data using bespoke criteria.
- Achieve highly tailored data manipulation for unique analytical challenges.

How Custom Functions are Handled and Why They Are Required

Pandas provides several methods to apply custom functions, each suited for different granularities of operation:

1. **apply():**
    - **What it does:** This is the most versatile method for applying a custom function along an axis of a DataFrame or to a Series.
    - **How it works:**
        - **axis:** You specify whether the function should be applied row-wise (axis='row' or axis=1) or column-wise (axis='column' or axis=0).
            - When applied column-wise (default), the function receives each column as a Series.
            - When applied row-wise, the function receives each row as a Series.
        - **result_type:** This parameter controls the structure of the output when applying row-wise, allowing you to broadcast results (expand to same shape), expand a Series result into multiple columns, or reduce it to a single value.
    - **Why it's required:** apply() is ideal for operations that need to process an entire Series (a column or a row) at once. For example, if you need to calculate a weighted average of several columns for each row, or perform a complex statistical test on each column.
2. **applymap():**
    - **What it does:** This method applies a custom function **element-wise** across an entire DataFrame. It passes each individual value in the DataFrame through your function.
    - **How it works:** applymap() takes a function and applies it to every single cell (element) in the DataFrame. If your DataFrame has 100 rows and 5 columns, your function will be called 500 times, once for each value.

- **Why it's required:** Useful for transformations that operate independently on each cell, such as formatting numerical values (e.g., rounding all numbers to two decimal places, converting all negative numbers to their absolute value), or conditionally modifying individual entries based on simple logic.

3. **map():**
    - **What it does:** This method applies a custom function **element-wise** specifically to a **Series** (i.e., a single column). It's similar to applymap() but only for Series.
    - **How it works:** You use map() directly on a Series, and it passes each value of that Series through your custom function.
    - **Why it's required:** Most efficient for element-wise transformations on a single column. For instance, converting numerical status codes into descriptive text labels based on a mapping dictionary, or applying a mathematical function to every value in a specific numerical column.
4. **Lambda Function:**
    - **What it does:** Lambda functions are small, anonymous (unnamed) functions defined inline. They are often used as the custom function argument within apply(), applymap(), or map() when the logic is simple and doesn't warrant defining a separate named function.
    - **How it works:** A lambda function is defined on a single line and typically consists of an expression that is returned. For example, a lambda function to double a number might look like lambda x: x * 2.
    - **Why it's required:** They provide a concise way to define simple, one-off functions directly where they are used, making the code cleaner and more readable for straightforward transformations without the overhead of formal function definition.

Why Custom Functions are Required:

Custom functions are indispensable because they provide the bridge between Pandas' powerful data structures and the unique analytical logic that specific problems often demand. They are required when:

- **Complex Feature Engineering:** You need to create new features that are not simple sums or averages but involve custom formulas or conditional logic derived from multiple existing columns.
- **Domain-Specific Transformations:** Your analysis requires transformations that are unique to your industry or data type (e.g., a specific algorithm for calculating a proprietary score).
- **Flexible Data Cleaning:** You need to implement bespoke rules for data validation or imputation that go beyond standard missing value handling.
- **Extending Capabilities:** You want to integrate external Python libraries or complex algorithms into your Pandas workflow directly.

In essence, custom functions empower users to tailor Pandas operations precisely to their analytical needs, providing unmatched flexibility in data manipulation and analysis.