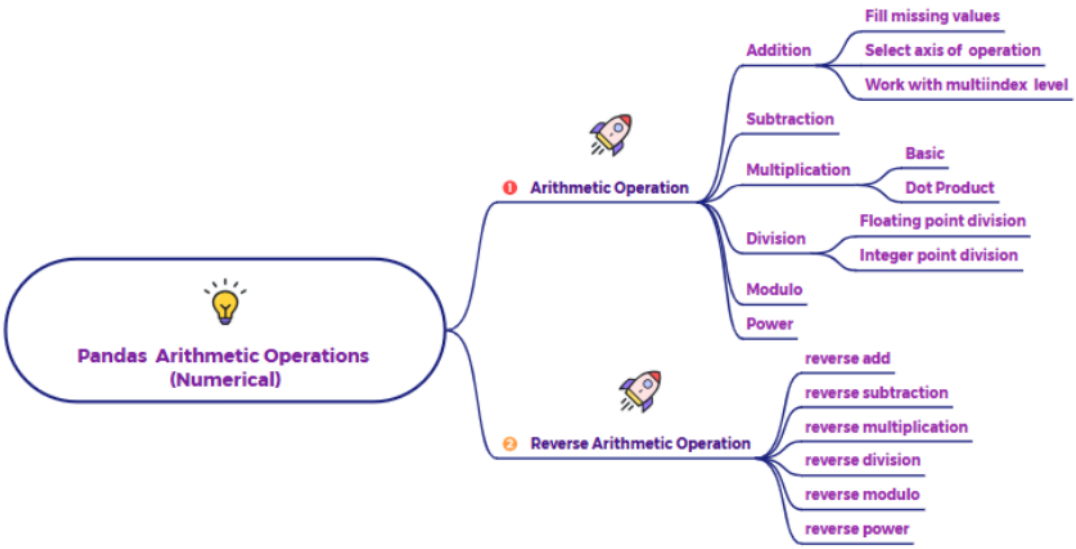
How to perform arithmetic operation in numerical variables

Arithmetic operations in Pandas refer to performing standard mathematical calculations (like addition, subtraction, multiplication, division, etc.) on numerical data within Series or DataFrames. These operations are fundamental for data cleaning, feature engineering, and deriving new insights from your numerical columns.



Purpose of Arithmetic Operations

The primary **purpose** of arithmetic operations is to **create new meaningful numerical variables, transform existing ones, or adjust values** based on specific analytical or business requirements. This allows you to:

- Calculate totals, differences, ratios, or percentages.
- Convert units (e.g., from meters to centimeters).
- Adjust values based on a baseline or factor.
- Prepare data for statistical modeling or visualization where derived metrics are needed.

How Arithmetic Operations are Handled and Why They Are Required

Pandas handles arithmetic operations very efficiently, often leveraging NumPy's capabilities. The image highlights two main categories: "Arithmetic Operation" and "Reverse Arithmetic Operation."

1. **Arithmetic Operation (Standard Operations):**
    - **What it does:** These are the straightforward mathematical calculations you'd expect. Pandas allows you to perform these operations element-wise (value by value) between:
        - A numerical Series/DataFrame and a single scalar number (e.g., sales_column + 10).
        - Two numerical Series (aligned by index) (e.g., revenue_column - cost_column).
        - Two numerical DataFrames (aligned by both index and columns).
    - **Specific Operations:**
        - **Addition:** + (e.g., adding a bonus to salaries). Pandas has flexible alignment, meaning it will try to match indices and columns before performing the operation. If there are missing matches, it will result in NaN (Not a Number) unless specified otherwise. This is useful for tasks like Fill missing values (e.g., using add(other_df, fill_value=0)). You can also Select axis of operation (e.g., adding row-wise or column-wise) and Work with multiindex level for more complex data structures.
        - **Subtraction:** - (e.g., calculating profit from revenue and cost).
        - **Multiplication:** * (e.g., converting quantity to total price per item by multiplying with unit price). This can be Basic element-wise or Dot Product for matrix multiplication (though typically done with NumPy for explicit matrix operations).
        - **Division:** / (floating-point division) and // (integer point division). Floating-point division gives a decimal result (e.g., 5 / 2 = 2.5), while integer division truncates to a whole number (e.g., 5 // 2 = 2).

        - **Modulo:** % (returns the remainder of a division, e.g., 5 % 2 = 1).
        - **Power:** ** (raises a number to a specified power, e.g., 2 ** 3 = 8).
    - **Why it's required:** These are fundamental for almost any quantitative analysis. They enable data transformation (e.g., converting raw counts to percentages), feature creation (e.g., calculating 'profit margin'), and data cleaning (e.g., adjusting for errors or biases).
2. **Reverse Arithmetic Operations:**
    - **What it does:** These are special methods (e.g., radd(), rsub(), rmul(), rdiv(), rmod(), rpow()) that perform the arithmetic operation with the operand order reversed. This is particularly useful when working with scalar values or Series/DataFrames that might have misaligned indices, and you want the other operand to be treated as the "left" operand.
    - **Why it's required:** While less commonly used directly than their standard counterparts, reverse operations become important in specific scenarios involving broadcasting rules or when you need to control how operations are performed with misaligned indices, especially when the scalar or smaller Series is on the left side of the operation. For example, if you have 5 - df['column'], the rsub method is implicitly called.

Broadcasting and Alignment: A key feature of Pandas arithmetic operations is **broadcasting** (similar to NumPy) and **automatic index/column alignment**. When you perform an operation between two Series/DataFrames, Pandas automatically aligns them based on their indices (and columns for DataFrames). If labels don't match, the result for that non-matching label will be NaN, unless a fill_value is specified. This automatic alignment prevents errors and simplifies operations on datasets with potentially different but related structures.

In summary, arithmetic operations in Pandas are vital for transforming raw numerical data into meaningful analytical features, enabling calculations, and preparing data for more complex statistical modeling and visualization.