2887. Fill Missing Data

Easy

Leetcode Link

Problem Description

The task is to process a DataFrame named `products` which represents a collection of products with their names, quantities, and prices. The DataFrame has the following columns: `name` which is of type object to represent the product names, `quantity` which is an integer indicating how many of the products are available, and `price` which is also an integer signifying the cost of each product.

Our main goal is to identify rows in the `quantity` column where the quantity is missing (indicated by `None`) and then fill in these missing values with zeros (`0`). This operation is required to ensure that the dataset is cleaner or perhaps more consistent for subsequent processing, which could include database insertion, data analysis or any other operation that requires complete data.

As an example, consider that we have a DataFrame with products like Wristwatch and WirelessEarbuds having a missing `quantity`. After the operation, these missing values should be replaced by `0` without affecting any other columns or the other (non-missing) values in the `quantity` column.

Intuition

The solution to this problem leverages the functionality provided by the pandas library in Python, which is widely used for data manipulation and analysis. Specifically, it involves the use of the `fillna()` function, which is a convenient method to fill NA/NaN values in a DataFrame.

Here's the rationale:

1. The `fillna()` function allows us to specify a value that replaces the missing or NaN (Not a Number) entries in a DataFrame or Series. Since we want to replace missing values in the `quantity` column with zero (`0`), this function is a suitable choice.
2. This function is called on the specific column (`quantity`) of our DataFrame (`products`). We use `products['quantity']` to isolate this column and then apply the `fillna(0)`, with `0` being the value to substitute for missing entries.
3. The `fillna()` operation is done in-place, meaning that it directly modifies the input DataFrame without the need for assignment unless otherwise specified by the `inplace` parameter.
4. Finally, the modified DataFrame is then returned with the missing `quantity` values replaced by zeros.

The operation is straightforward and efficient, requiring only a single line of code to achieve the desired result.

Solution Approach

In our solution, the essential function is `fillna()`, part of the pandas library's `DataFrame` methods. It's used to fill NA/NaN values with a specified scalar value or a dictionary/array. The fill value for missing data in our case is `0`. The choice of this function is driven by the simplicity and effectiveness in dealing with missing data in pandas DataFrames.

Here is the breakdown of the approach used in the provided code snippet:

1. The function `fillMissingValues(products: pd.DataFrame) -> pd.DataFrame` is defined to take a DataFrame as an input parameter and returns a DataFrame with the missing values filled in.
2. Inside the function, we access the `quantity` column of the provided `products` DataFrame using `products['quantity']`.
3. We then call `fillna()` on this column with the argument `0`, which represents the value we want to use to replace the NaN (or None) values. The expression becomes `products['quantity'].fillna(0)`.
4. The `fillna()` function, by default, does not modify the existing DataFrame. Instead, it returns a new Series with the missing values filled. Therefore, we directly assign the result back to `products['quantity']` to update that column with the filled in values.
5. After the `fillna()` operation, the `quantity` column no longer has missing values; all such instances have been replaced with `0`.
6. The last step is to return the modified `products` DataFrame from the function, now with all the missing values in the `quantity` column filled with `0`.

We do not use additional data structures, algorithms, or patterns as the problem can be effectively solved using the DataFrame and its methods provided by pandas. This approach is very efficient because it utilizes highly optimized pandas library functions designed specifically to handle such data manipulation tasks.

Example Walkthrough

Let's visualize how the solution approach will work with a small example. Suppose we have the following initial `products` DataFrame:

name	quantity	price
Wristwatch	None	50
WirelessEarbuds	None	150
Notebook	20	5

According to the problem description, we need to replace the `None` values in the `quantity` column with `0`. By following the solution approach, this is what happens:

1. We define the function `fillMissingValues(products: pd.DataFrame) -> pd.DataFrame`.
2. Inside this function, we target the `quantity` column of `products` using `products['quantity']`.
3. We use `products['quantity'].fillna(0)` which generates a new Series:

quantity
0
0
20

4. This result is then assigned back to `products['quantity']`, effectively updating the original DataFrame.
5. The function then returns the updated DataFrame which now looks like this:

name	quantity	price
Wristwatch	0	50
WirelessEarbuds	0	150
Notebook	20	5

Observing the final result, we can confirm that the missing values in the `quantity` column were successfully replaced with `0`, achieving the objective of processing the data as required.

Python Solution

```python
import pandas as pd

# Define a function to fill missing values in the 'quantity' column of a DataFrame
def fillMissingValues(products: pd.DataFrame) -> pd.DataFrame:
    # Replace NaN values in the 'quantity' column with 0
    products['quantity'] = products['quantity'].fillna(0)
    # Return the DataFrame after filling in missing values
    return products

```

Java Solution

```java
import java.util.List;
import java.util.Objects;

public class Product {
    private String name;
    private Integer quantity;

    // Constructor, getters and setters for product

    public Product(String name, Integer quantity) {
        this.name = name;
        this.quantity = quantity;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Integer getQuantity() {
        return quantity;
    }

    public void setQuantity(Integer quantity) {
        this.quantity = quantity;
    }
}

public class ProductUtils {

    /**
     * Fills missing values in the 'quantity' field of a list of Product objects.
     *
     * @param products List of Product objects.
     * @return The same List of Product objects with 'quantity' missing values replaced with 0.
     */
    public static List<Product> fillMissingValues(List<Product> products) {
        for (Product product : products) {
            // If the quantity is null (equivalent to NaN in pandas), set it to 0
            if (product.getQuantity() == null) {
                product.setQuantity(0);
            }
        }
        // Return the list after filling in missing values
        return products;
    }
}

```

C++ Solution

```cpp
#include <vector>

// Define a struct to represent a product with a quantity attribute
struct Product {
    // You can add other product attributes here
    int quantity;

    // Constructor to initialize the product with a quantity
    Product(int qty): quantity(qty) {}
};

// Define a function to fill missing values in the 'quantity' field of a vector of Products
std::vector<Product> fillMissingValues(std::vector<Product> &products) {
    // Iterate over each Product in the vector by reference
    for (Product &p : products) {
        // Check if the quantity is marked as 'missing' using a negative value as the indicator
        if (p.quantity < 0) {
            // Replace the 'missing' value with 0
            p.quantity = 0;
        }
    }
    // Return the vector after filling in missing values
    return products;
}

```

Typescript Solution

```typescript
// Import your custom types or interfaces that support DataFrame operations
import { DataFrame } from './path-to-your-dataframe-definitions';

// Define a function to fill missing values in the 'quantity' column of a DataFrame
function fillMissingValues(products: DataFrame): DataFrame {
    // Check if 'products' has a 'quantity' column and fill NaN values with 0
    if (products.quantity) {
        // Assuming 'replaceNaNWithZero' is a method provided by your DataFrame library
        products.quantity = products.quantity.replaceNaNWithZero(0);
    }
    // Return the DataFrame after filling in missing values
    return products;
}

```

Time and Space Complexity

The time complexity of the code can be considered as `O(n)`, where `n` is the number of rows in the DataFrame `products`. This is because the `fillna` method needs to scan through the 'quantity' column and fill missing values with zeros.

The space complexity of the method is `O(1)`. This is because the `fillna` operation is done in place, and no additional space proportional to the size of the input data is used.