2878. Get the Size of a DataFrame

Easy

Leetcode Link

Problem Description

This problem involves working with a `pandas` DataFrame representing information about players. The DataFrame is given as an input and contains various columns, such as `player_id`, `name`, `age`, `position`, and possibly others that are not fully detailed (`...` suggests there could be additional columns). Our task is to write a function that calculates two simple metrics: the number of rows and the number of columns in the DataFrame.

To clarify:

- The number of rows represents how many players there are in the dataset.
- The number of columns represents how many attributes or pieces of information we have for each player.

The output should be returned as an array with two elements: `[number of rows, number of columns]`. This is a straightforward problem that requires knowledge of how to work with pandas DataFrames, particularly how to access information about their size or shape.

Intuition

The intuition behind the solution is to use the inbuilt properties of pandas DataFrames. Every DataFrame in `pandas` has a `.shape` attribute, which returns a tuple containing the number of rows and columns (in that order). Since `pandas` is designed for data manipulation and analysis, accessing the dimensions of a DataFrame is a common task that is made simple by this attribute.

Here's how we can think about arriving at the solution:

1. Access the `.shape` attribute of the DataFrame, which gives us the size of the DataFrame as a tuple. For example, if there are 10 rows and 5 columns, `players.shape` would return `(10, 5)`.
2. Since the return type expected is a list, we convert this tuple to a list with `list(players.shape)`. This is necessary because the problem description specifies that the result should be an array, which in Python terms, corresponds to a list.
3. We return this list as the final result.

This problem doesn't require any complicated logic or algorithmic thinking. It's an application of built-in functionality provided by the `pandas` library.

Solution Approach

The implementation of the solution is quite straightforward, as it leverages the pandas library's built-in functionality. Here's a step-by-step explanation of what happens in the code:

1. The function `getDataframeSize` is defined with one parameter, `players`, which is expected to be a pandas DataFrame.
2. Inside the function, we use `players.shape` to access the shape of the DataFrame. The `shape` attribute is a standard feature of pandas DataFrames and directly gives us the number of rows and columns as a tuple.
3. We then convert this tuple into a list using `list(players.shape)` because the expected output format is a list, according to the problem specification `[number of rows, number of columns]`.
4. Finally, the list is returned as the result.

The data structures and patterns used in this solution include:

- The **DataFrame** from pandas: It's a two-dimensional, size-mutable, and potentially heterogeneous tabular data structure with labeled axes (rows and columns).
- **Tuple**: A simple Python data structure used to store a sequence of immutable Python objects. The `shape` attribute of a DataFrame returns a tuple representing the dimensions of the DataFrame.
- **List**: A built-in Python data structure that is used to return the final result. Lists are mutable sequences, which makes them suitable for returning an array-like output.

While the solution does not involve any complex algorithms, the understanding of data structures like tuples and lists is important for effectively working with and manipulating data in Python. The solution approach is almost entirely reliant on the capabilities provided by pandas, highlighting how the library simplifies data-related operations.

There is no pseudocode needed for such a simple implementation, and the actual Python code provided is self-explanatory given the explanation above.

Example Walkthrough

Let's suppose we have a small pandas DataFrame `players` that represents information about a soccer team's players. The `players` DataFrame might look like this:

```
   player_id  name          age  position
0  1          Alice Johnson  22   Forward
1  2          Bob Smith      29   Midfielder
2  3          Charlie Davis  24   Defender
```

As per the solution approach detailed, we are going to use the `.shape` property of the DataFrame to determine the number of rows and columns. Let's perform the steps:

1. First, we access the `.shape` attribute of our DataFrame `players`. The `shape` attribute provides the dimensions of the DataFrame as a tuple of the form `(rows, columns)`.

   By calling `players.shape`, we obtain the tuple `(3, 4)` signifying that our DataFrame has 3 rows and 4 columns.

2. Next, we need to convert this tuple into a list because we need to return our result as an array (which correlates to a list in Python). Thus, we use the `list()` function on our shape tuple.

   So, `list(players.shape)` will convert our tuple `(3, 4)` to the list `[3, 4]`.

3. Finally, this list `[3, 4]` is exactly what we want - it tells us there are 3 rows (players) and 4 columns (attributes like `player_id`, `name`, `age`, `position`) in our DataFrame.

And that's it! Using these steps, we have used the DataFrame's built-in `.shape` attribute to quickly determine the DataFrame's size, following the intuitive and streamlined solution approach to solving this problem.

Python Solution

```python
import pandas as pd
from typing import List

def getDataframeSize(players: pd.DataFrame) -> List[int]:
    # Return the dimensions of the dataframe
    # 'players.shape' returns a tuple (number of rows, number of columns)
    # The output is converted to a list [number of rows, number of columns]
    return list(players.shape)

```

Java Solution

```java
import java.util.ArrayList;
import java.util.List;

public class DataFrameUtil {

    /**
     * Gets the size of the two-dimensional data structure akin to a DataFrame.
     * In this particular function, it's assumed that 'players' is a two-dimensional array.
     *
     * @param players A two-dimensional array representing the "DataFrame".
     * @return A list containing two elements: the number of rows and the number of columns.
     */
    public List<Integer> getDataframeSize(String[][] players) {
        // Create an ArrayList to hold the dimensions
        List<Integer> dimensions = new ArrayList<>();

        // Check if the array 'players' is not null and has at least one row
        if (players != null && players.length > 0) {
            // Add the number of rows to the list
            dimensions.add(players.length);

            // Add the number of columns to the list (assuming all rows have the same number of columns)
            dimensions.add(players[0].length);
        } else {
            // If the array is null or empty, add 0 for both rows and columns
            dimensions.add(0);
            dimensions.add(0);
        }

        // Return the list of dimensions
        return dimensions;
    }
}

```

C++ Solution

```cpp
#include <vector>

// A custom DataFrame class (needs proper implementation to store data)
class DataFrame {
public:
    // Method to get the dimensions of the DataFrame
    // Assuming it returns a std::pair representing dimensions (rows, columns)
    std::pair<int, int> shape() const {
        // Placeholder implementation
        // This method needs a proper implementation to return the actual size
        return std::make_pair(0, 0); // Replace with actual data dimensions
    }
};

// Function to return the dimensions of the DataFrame as a vector
std::vector<int> getDataframeSize(const DataFrame& players) {
    // Get the dimensions of the DataFrame as a pair (rows, columns)
    std::pair<int, int> dimensions = players.shape();

    // Convert the pair to a vector [rows, columns] and return
    std::vector<int> size = {dimensions.first, dimensions.second};
    return size;
}

```

Typescript Solution

```typescript
// Define an interface that represents the structure of each player object;
// each player as a record can have multiple attributes.
interface Player {
    // Define potential player attributes
    [key: string]: any; // This allows any number of properties of any type
}

// This function takes an array of player objects and returns
// the dimensions of this "data frame" as an array [number of rows, number of columns]
function getDataframeSize(players: Player[]): [number, number] {
    // Check if the array is empty; if so, return [0, 0] as the size
    if (players.length === 0) {
        return [0, 0];
    } else {
        // Assume that all objects have the same number of keys (columns)
        // Get the number of rows from the length of the players array
        const numberOfRows: number = players.length;
        // Get the number of columns from the keys of the first player object
        const numberOfColumns: number = Object.keys(players[0]).length;
        // Return the dimensions as an array [number of rows, number of columns]
        return [numberOfRows, numberOfColumns];
    }
}

// Example usage:
// const players: Player[] = [{ name: 'Alice', score: 10 }, { name: 'Bob', score: 15 }];
// const size: [number, number] = getDataframeSize(players);
// console.log(size); // Output would be [2, 2] for the example given

```

Time and Space Complexity

Time Complexity:

The time complexity of `getDataframeSize` function is `O(1)`.

The function accesses the `.shape` attribute of a pandas DataFrame, which is an operation that runs in constant time regardless of the size of the DataFrame since it only returns the dimensions of the DataFrame that are already stored as metadata.

Space Complexity:

The space complexity of the `getDataframeSize` function is also `O(1)`.

The function returns the dimensions of the DataFrame as a list, which always contains two integers regardless of the size of the DataFrame. Therefore, the space used does not scale with the input size, making it constant space complexity.