2884. Modify Columns

Easy

Problem Description

In this problem, we have a pandas DataFrame named `employees` that contains two columns: `name` and `salary`. The `name` column is of type `object`, which typically means it contains string data, while the `salary` column contains integer values representing the salary of each employee in the company.

The task is to adjust the employee salaries by doubling each one. In other words, you must write a code that modifies the `salary` column by multiplying each salary by 2. This operation should update the original DataFrame in place.

The main objective is to return the updated DataFrame with the modified salaries.

Intuition

To solve this problem, we know that pandas DataFrames offer a very convenient way to perform operations on columns using simple arithmetic syntax. You can select a column and then apply an operation to each element (row) of the column efficiently.

Given that the problem statement requires the salary to be multiplied by 2, this can be accomplished by using the multiplication assignment operator (`*=`) in Python. This operator multiplies the left-hand operand (the salary column of the DataFrame) by the right-hand operand (the constant value 2) and assigns the result back to the left-hand operand.

Solution Approach

The solution approach for modifying the `salary` column in the `employees` DataFrame is quite straightforward and does not require complex algorithms or data structures. It simply leverages the features provided by the pandas library to manipulate DataFrame columns.

The steps for the implementation are as follows:

1. The pandas library is imported with the alias `pd`.
2. A function named `modifySalaryColumn` is defined, which takes a single parameter, `employees`. This parameter is expected to be a pandas DataFrame with a structure matching the one described in the problem (with `name` and `salary` columns).
3. Within the function, the `salary` column of the DataFrame is accessed using bracket notation: `employees['salary']`.
4. The multiplication assignment operator (`*=`) is applied to the `salary` column to multiply each value by 2. This operation is done in place, which means the existing `salary` column in the DataFrame is updated with the new values.
5. Finally, the updated `employees` DataFrame is returned.

The solution does not make use of complex patterns, since it is able to fully utilize pandas' ability to perform vectorized operations across an entire column, applying the multiplication to each salary in one succinct line of code. This is one of the advantages of using libraries like pandas, designed for efficient data manipulation.

Example Walkthrough

Let's walk through a small example to illustrate the solution approach:

Suppose we have the following `employees` DataFrame:

```
    name  salary
0  Alice   50000
1    Bob   60000
2  Carol   55000
```

To adjust the salaries by doubling each one, we will apply the steps outlined in the solution approach:

1. We start by importing pandas:

```
import pandas as pd
```

2. We define the `modifySalaryColumn` function that takes the `employees` DataFrame as a parameter:

```
def modifySalaryColumn(employees):
```

3. Inside the function, we access the `salary` column:

```
    employees['salary']
```

4. We use the multiplication assignment operator to double each salary:

```
    employees['salary'] *= 2
```

After executing the above line of code, the `salary` column in the `employees` DataFrame is updated to:

```
    name  salary
0  Alice  100000
1    Bob  120000
2  Carol  110000
```

5. Lastly, we return the updated `employees` DataFrame:

```
    return employees
```

Now, let's use this function with our example DataFrame:

```
# Initial DataFrame
employees = pd.DataFrame({
    'name': ['Alice', 'Bob', 'Carol'],
    'salary': [50000, 60000, 55000]
})

# Modify the salary column by doubling the salaries
modified_employees = modifySalaryColumn(employees)

# The resulting DataFrame
print(modified_employees)
```

The output will be:

```
    name  salary
0  Alice  100000
1    Bob  120000
2  Carol  110000
```

As demonstrated, each salary in the `salary` column has been successfully doubled, achieving the task of modifying the employee salaries in place. The `modifySalaryColumn` function is efficient and leverages the power of pandas for vectorized operations on DataFrame columns.

Solution Implementation

Python

```
import pandas as pd

def modifySalaryColumn(employees: pd.DataFrame) -> pd.DataFrame:
    # Double the values in the 'salary' column of the DataFrame
    employees['salary'] = employees['salary'].apply(lambda x: x * 2)

    # Return the modified DataFrame
    return employees
```

Java

```
import java.util.List;
import java.util.Map;

public class SalaryModifier {

    /**
     * Doubles the salary for each employee in the list.
     *
     * @param employees A list of maps, where each map represents an employee with the 'salary' key.
     * @return The list of employees with updated salaries.
     */
    public static List<Map<String, Object>> modifySalaryColumn(List<Map<String, Object>> employees) {
        for (Map<String, Object> employee : employees) {
            // Check if the employee map contains the 'salary' key
            if (employee.containsKey("salary")) {
                // Get the current salary
                Object salaryObj = employee.get("salary");
                if (salaryObj instanceof Number) {
                    // Double the salary and update the employee map
                    Number salary = (Number) salaryObj;
                    employee.put("salary", salary.doubleValue() * 2);
                }
            }
        }
        // Return the modified list of employees with updated salaries
        return employees;
    }
}
```

C++

```
#include <vector>
#include <utility> // For std::pair
#include <algorithm> // For std::transform
#include <string>

class Employee {
public:
    std::string name;
    float salary;

    Employee(std::string n, float s) : name(std::move(n)), salary(s) {}

    // Function to double the salary
    void doubleSalary() {
        salary *= 2;
    }
};

void modifySalaryColumn(std::vector<Employee>& employees) {
    // Double the salaries of all employees in the vector
    std::for_each(employees.begin(), employees.end(), [](Employee& e) {
        e.doubleSalary();
    });
}
```

TypeScript

```
// Define the employee type with at least a salary field
type Employee = {
    salary: number;
    // ... other fields
};

/**
 * Doubles the 'salary' property of each employee in an array
 * @param employees - Array of employees to be modified
 * @returns An array of employees with modified salaries
 */
function modifySalaryColumn(employees: Employee[]): Employee[] {
    // Double the values in the 'salary' property of each employee
    employees.forEach(employee => {
        employee.salary *= 2;
    });

    // Return the modified array of employees
    return employees;
}
```

This function would be used with an array of objects, like so:

```typescript
// Example array of employees
let employees: Employee[] = [
    { salary: 50000 /*, other properties */ },
    { salary: 60000 /*, other properties */ },
    // ... other employees
];

// Doubling the salary of each employee
employees = modifySalaryColumn(employees);
```

```
import pandas as pd

def modifySalaryColumn(employees: pd.DataFrame) -> pd.DataFrame:
    # Double the values in the 'salary' column of the DataFrame
    employees['salary'] = employees['salary'].apply(lambda x: x * 2)

    # Return the modified DataFrame
    return employees
```

Time and Space Complexity

Time Complexity

The time complexity of the `modifySalaryColumn` function is $O(n)$, where n is the number of elements in the 'salary' column of the `employees` DataFrame. This is because the function performs a single operation (multiplication by 2) on each element of the 'salary' column.