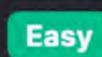
Leetcode Link



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.

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.

Given that the problem statement requires the salary to be multiplied by 2, this can be accomplished by using the multiplication

The solution takes advantage of the fact that pandas will automatically apply the multiplication to each element in the column, thus doubling every employee's salary as required by the problem.

The solution approach for modifying the salary column in the employees DataFrame is quite straightforward and does not require

Solution Approach

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

using libraries like pandas, designed for efficient data manipulation. Example Walkthrough

across an entire column, applying the multiplication to each salary in one succinct line of code. This is one of the advantages of

Suppose we have the following employees DataFrame:

name salary 50000

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

60000 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: 1 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']
- employees['salary'] *= 2

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

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

return employees

employees = pd.DataFrame({

Carol 110000

DataFrame columns.

Python Solution

return employees

11

12

13

14

15

16

17

Return the modified DataFrame

* @return The list of employees with updated salaries.

for (Map<String, Object> employee : employees) {

if (salaryObj instanceof Number) -

if (employee.containsKey("salary")) {

// Get the current salary

// Check if the employee map contains the 'salary' key

Object salaryObj = employee.get("salary");

name salary

Alice 100000

Bob 120000 Carol 110000

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

})

5

1 # Initial DataFrame

'name': ['Alice', 'Bob', 'Carol'],

'salary': [50000, 60000, 55000]

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

```
# Modify the salary column by doubling the salaries
   modified employees = modifySalaryColumn(employees)
10 # The resulting DataFrame
11 print(modified_employees)
The output will be:
      name salary
      Alice 100000
       Bob 120000
```

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)

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

```
Java Solution
   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.
```

```
// Double the salary and update the employee map
19
20
                       Number salary = (Number) salaryObj;
21
                       employee.put("salary", salary.doubleValue() * 2);
22
23
24
           // Return the modified list of employees with updated salaries
25
26
           return employees;
27
28 }
29
C++ Solution
 1 #include <vector>
2 #include <utility> // For std::pair
3 #include <algorithm> // For std::transform
   #include <string>
   class Employee {
   public:
       std::string name;
       float salary;
10
       Employee(std::string n, float s) : name(std::move(n)), salary(s) {}
11
12
```

public static List<Map<String, Object>> modifySalaryColumn(List<Map<String, Object>> employees) {

18 void modifySalaryColumn(std::vector<Employee>& employees) { 20 // Double the salaries of all employees in the vector std::for_each(employees.begin(), employees.end(), [](Employee& e) { 21

});

// Function to double the salary

void doubleSalary() {

e.doubleSalary();

salary *= 2;

13

15

16

22

23

24 }

*/

11

13

14

10

17 };

25 Typescript Solution // Define the employee type with at least a salary field type Employee = { salary: number; // ... other fields 6 /** * 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

}); 16 17 18 19

employees.forEach(employee => {

employee.salary *= 2;

```
// Return the modified array of employees
       return employees;
20 }
This function would be used with an array of objects, like so:
 1 // Example array of employees
   let employees: Employee[] = [
       { salary: 50000 /*, other properties */ },
       { salary: 60000 /*, other properties */ },
       // ... other employees
 6
   // Doubling the salary of each employee
```

Time and Space Complexity

employees = modifySalaryColumn(employees);

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.

complexity can be O(n) in practice.

Space Complexity The space complexity is 0(1) as the modification is done in place and does not require additional space that grows with the input size. However, if the pandas DataFrame internally opts to create a copy during the operation due to its settings, the space