2886. Change Data Type

Easy

Problem Description

The given problem presents a data structure scenario, where we have a `DataFrame` named `students` consisting of several columns: `student_id`, `name`, `age`, and `grade`. The `student_id` is of type integer, `name` is an object (which usually means string in this context), `age` is an integer, and the `grade` is a float. The task at hand is to convert the data type of the `grade` column from float to integer. The conversion should be reflected in the `students` DataFrame, and the updated DataFrame should be returned as an output.

Intuition

In this solution, the focus is to implement a type conversion of one column within a DataFrame using the Pandas library in Python. The motivation behind the solution is the need to standardize or comply with a certain data format requirement. The column `grade` is initially in a floating-point (decimal) format, which might be more precise than needed. The goal is to transform this column to an integer format, which can be easier to handle for certain computations or when presenting the data.

Solution Approach

The implementation of the solution relies on the Pandas library, which is a powerful and flexible tool for data manipulation in Python. The primary components of Pandas used here are DataFrames, which are two-dimensional data structures with labeled axes (rows and columns). DataFrames allow you to store and manipulate tabular data where the manipulation includes operations like filtering, grouping, and, as required by our problem, type conversion of columns.

The solution provided follows these steps:

1. The `astype` function is invoked on the `grade` column of the `students` DataFrame. The `astype` function is a built-in pandas function specifically designed for type conversion of series-like data structures.
2. The argument passed to `astype` is `int`, which indicates the desired target data type for the `grade` column.
3. The result of the `astype` function, which is a Series with the `grade` data converted to integers, replaces the original `grade` column within the `students` DataFrame.
4. The updated `students` DataFrame, now with the `grade` column containing integer types, is returned from the function.

This method is effective and efficient because:

- **Data Integrity**: Using `astype(int)` ensures that all the values will be converted to integers, as long as they can be represented as such. If a value cannot be converted (for example, a non-numeric string or `NaN` values in the column), `astype` would raise an error, alerting the user to the presence of incompatible data.
- **Simplicity**: The `astype` method is straightforward to use and doesn't require writing a custom conversion function or loop, making the code cleaner and more readable.
- **Performance**: Since `astype` is a method provided by Pandas, it is heavily optimized and can convert types very quickly, even for large datasets.

Consequently, this specific problem does not introduce algorithmic complexity. It is an application of a Pandas function that abstracts away the complexity of type conversion. The focus is more on understanding how to use the library function rather than implementing an algorithm from scratch.

Example Walkthrough

Let's suppose we have a small `DataFrame` named `students` with the following data:

```
  student_id     name  age  grade
0          1    Alice   17   89.5
1          2      Bob   18   91.3
2          3  Charlie   17   78.0
3          4    Diana   18   72.8
```

We aim to convert the `grade` column from float to integer data type.

Here's an illustrative example of the solution approach:

1. We initially have the `grade` column with float data types, which in our example are `89.5`, `91.3`, `78.0`, and `72.8`.
2. We apply the `astype(int)` function on the `grade` column. The DataFrame manipulation looks like this in code:
   ```
   students['grade'] = students['grade'].astype(int)
   ```
3. After applying the `astype(int)` function, the floating-point numbers are truncated to integers, which means, for instance, `89.5` would become `89`, and `72.8` would become `72`.
4. Our `students` DataFrame is now updated. The `grade` column data type is changed from float to integer, resulting in the following DataFrame:

```
  student_id     name  age  grade
0          1    Alice   17     89
1          2      Bob   18     91
2          3  Charlie   17     78
3          4    Diana   18     72
```

This method efficiently achieves the conversion of the entire column without the need for iterating over each row, which is both simpler and more performant, especially with large data sets.

Solution Implementation

Python

```python
import pandas as pd

def change_datatype(students: pd.DataFrame) -> pd.DataFrame:
    """
    Convert the datatype of the 'grade' column to integer.

    Parameters:
    students (pd.DataFrame): A DataFrame with a 'grade' column.

    Returns:
    pd.DataFrame: The updated DataFrame with the 'grade' column as integers.
    """
    # Assuming 'grade' column originally contains values that can be directly converted to integers
    students['grade'] = students['grade'].astype(int)
    return students
```

Java

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

public class DataConverter {

    // Method to change the datatype of the 'grade' column to Integer
    public static List<Map<String, Object>> changeDatatype(List<Map<String, Object>> students) {
        // Iterate over each record (i.e., map) in the list
        for (Map<String, Object> student : students) {
            // Assuming that the grades are stored as strings that can be parsed to integers
            String gradeAsString = (String) student.get("grade");
            // Convert the 'grade' string to an integer and update the map record
            int gradeAsInt = Integer.parseInt(gradeAsString);
            student.put("grade", gradeAsInt);
        }
        // Return the updated list of records with the 'grade' column as integers
        return students;
    }

    // The main method for testing the changeDatatype method
    public static void main(String[] args) {
        // You would initialize your List<Map<String, Object>> here and pass it to the changeDatatype method
        // List<Map<String, Object>> students = ...
        // List<Map<String, Object>> studentsWithIntGrades = changeDatatype(students);

        // Implement the test logic here, such as printing the updated list to verify the 'grade' column data type
    }
}
```

C++

```cpp
#include <vector>
#include <iostream>

// For simplicity, the following struct represents a student record.
struct Student {
    std::string name;
    int grade; // Here, we're assuming 'grade' will be stored as integer.
};

// Accept a vector of Student records and update the 'grade' field to integer if necessary.
std::vector<Student> changeDatatype(std::vector<Student> students) {
    // Assuming each student's grade is in a valid format that can be directly interpreted as an integer.
    // Hence, no actual conversion is performed in C++ because 'grade' is already an integer.
    // If a conversion was needed, we would need to parse the string to integer using std::stoi, for example.

    // Return the updated list of students.
    return students;
}

// Example usage:
int main() {
    // Create a vector of Student records.
    std::vector<Student> students = {{"Alice", 90}, {"Bob", 85}, {"Charlie", 92}};

    // Call the function to ensure grades are in integer datatype.
    students = changeDatatype(students);

    // Printing the students to show the result (grades are already integers; this will just print them out).
    for(const auto& student : students) {
        std::cout << "Student Name: " << student.name << ", Grade: " << student.grade << std::endl;
    }

    return 0;
}
```

TypeScript

```typescript
// Assuming that we have some global type definitions similar to Pandas DataFrame structure

// Define an interface to represent the structure of our students object.
interface StudentRecord {
    // Other properties for students would be defined here.
    grade: string; // Initially the grade is a string
}

// Define a type alias for an array of student records, mimicking DataFrame
type DataFrame = StudentRecord[];

// Function to change the datatype of the 'grade' property to a number
function changeDatatype(students: DataFrame): DataFrame {
    // Convert the 'grade' property of each student record from a string to an integer
    students.forEach((student) => {
        student.grade = parseInt(student.grade);
    });

    // Return the updated students array with 'grade' property as integers.
    return students;
}
```

```python
import pandas as pd

def change_datatype(students: pd.DataFrame) -> pd.DataFrame:
    """
    Convert the datatype of the 'grade' column to integer.

    Parameters:
    students (pd.DataFrame): A DataFrame with a 'grade' column.

    Returns:
    pd.DataFrame: The updated DataFrame with the 'grade' column as integers.
    """
    # Assuming 'grade' column originally contains values that can be directly converted to integers
    students['grade'] = students['grade'].astype(int)
    return students
```

Time and Space Complexity

The time complexity of the `changeDatatype` function primarily depends on the time it takes to change the data type of the 'grade' column in the pandas DataFrame. When using `astype(int)`, pandas internally iterates over each element to convert it to an integer. Therefore, the time complexity is `O(n)`, where `n` is the number of elements in the 'grade' column.