# 2885. Rename Columns

Easy

## Problem Description

In this problem, we are given the structure of a `DataFrame` named `students` which contains four columns: `id`, `first`, `last`, and `age`. The goal is to write a function that will rename these columns to `student_id`, `first_name`, `last_name`, and `age_in_years`, respectively. This task simulates common data manipulation operations where renaming columns is necessary for either clarity or to conform to certain naming conventions or requirements. The challenge lies in accurately mapping the old column names to the new ones and ensuring that the resulting `DataFrame` accurately reflects these changes.

## Intuition

To approach the solution for renaming columns in a `DataFrame`, we use the `rename` method provided by the pandas library. The `rename` method allows for either in-place updating of a `DataFrame` or can return a new `DataFrame` with updated column names. Here, the key idea is to pass a dictionary to the `columns` parameter of the `rename` method, where each key-value pair represents the old column name (key) and the new column name (value).

The `inplace=True` argument is used to apply the changes directly to the original `DataFrame` `students` rather than creating a new one, which can be more memory efficient and straightforward for this specific task.

## Solution Approach

The implementation of the solution involves directly interacting with the pandas `DataFrame` structure. To walk through the solution:

1. First, we define a function `renameColumns` that accepts a `DataFrame` named `students` as its parameter.
2. We then use the `rename` method on the `students` `DataFrame`. This method is part of the pandas library and is specifically designed to alter axis labels, which in this case are the column names.
3. We pass a dictionary to the `columns` parameter where each key-value pair maps an existing column name to the desired new name. In our case:
   - `'id'` is mapped to `'student_id'`
   - `'first'` is mapped to `'first_name'`
   - `'last'` is mapped to `'last_name'`
   - `'age'` is mapped to `'age_in_years'`
4. The `inplace=True` argument is included, indicating that the renaming should be performed in the original `DataFrame` without creating a new one. This results in changes being applied directly and immediately to `students`.
5. After renaming, the same `DataFrame` `students`, which now has its columns renamed, is returned from the function.

No additional algorithms, complex data structures, or unusual patterns are required for this task, as the pandas `DataFrame` and its methods provide all the necessary functionalities. The simplicity of the solution relies on the effective use of pandas as a powerful data manipulation tool, which allows such tasks to be performed succinctly and efficiently.

### Example Walkthrough

Let's illustrate the solution approach with a simple example. Assume we have a `DataFrame` named `students` with the following data:

| id | first | last | age |
|---|---|---|---|
| 1 | Alice | Smith | 22 |
| 2 | Bob | Jones | 23 |

Our goal is to rename the columns using the `renameColumns` function as described in the solution approach.

1. We initiate the `renameColumns` function by passing our `students` data.
2. We then call the `rename` method available in pandas DataFrame and pass the necessary arguments as follows:

```
students.rename(columns={
    'id': 'student_id',
    'first': 'first_name',
    'last': 'last_name',
    'age': 'age_in_years'
}, inplace=True)
```

3. The dictionary passed to the `columns` parameter inside the `rename` method specifies our desired name changes: `'id'` to `'student_id'`, `'first'` to `'first_name'`, `'last'` to `'last_name'`, and `'age'` to `'age_in_years'`.
4. Since we used `inplace=True`, the original `DataFrame` `students` is modified directly.

After running the `renameColumns` function, the `students` DataFrame now looks like this:

| student_id | first_name | last_name | age_in_years |
|---|---|---|---|
| 1 | Alice | Smith | 22 |
| 2 | Bob | Jones | 23 |

This demonstrates how simple and efficient renaming columns can be using the pandas library's built-in methods, specifically the `rename` method.

## Solution Implementation

### Python

```python
import pandas as pd  # Import the pandas library

def rename_columns(students_df: pd.DataFrame) -> pd.DataFrame:
    """
    This function renames specific columns of a DataFrame to standardized names.

    Parameters:
    students_df (pd.DataFrame): DataFrame containing student information with columns to be renamed.

    Returns:
    pd.DataFrame: A DataFrame with renamed columns.
    """
    # Define a dictionary mapping old column names to new standardized column names
    column_rename_mapping = {
        'id': 'student_id',
        'first': 'first_name',
        'last': 'last_name',
        'age': 'age_in_years',
    }

    # Rename the columns using the provided mapping
    # Note: inplace=False ensures that the original DataFrame is not modified and a new DataFrame is returned
    students_df_renamed = students_df.rename(columns=column_rename_mapping, inplace=False)

    return students_df_renamed

# The rename_columns function can be used as follows:
# Assume students_df is a DataFrame with columns 'id', 'first', 'last', and 'age'
# renamed_students_df = rename_columns(students_df)
```

### Java

```java
import java.util.Map;
import java.util.HashMap;
import org.apache.commons.collections4.map.HashedMap; // Apache Commons Collections library for the map

public class DataFrameUtils {

    /**
     * This method renames specific columns of a DataFrame to standardized names.
     *
     * @param studentsDf A DataFrame object containing student information with columns to be renamed.
     * @return A DataFrame with renamed columns.
     */
    public static DataFrame renameColumns(DataFrame studentsDf) {
        // Define a mapping from old column names to new standardized column names
        Map<String, String> columnRenameMapping = new HashMap<>();
        columnRenameMapping.put("id", "student_id");
        columnRenameMapping.put("first", "first_name");
        columnRenameMapping.put("last", "last_name");
        columnRenameMapping.put("age", "age_in_years");

        // Create a new DataFrame for the renamed columns
        DataFrame studentsDfRenamed = new DataFrame();

        // Iterate over each column, renaming as necessary
        for (String column : studentsDf.getColumns()) {
            if (columnRenameMapping.containsKey(column)) {
                // If the column is in the mapping, rename it
                studentsDfRenamed.renameColumn(column, columnRenameMapping.get(column));
            } else {
                // Otherwise, keep the original name
                studentsDfRenamed.renameColumn(column, column);
            }
        }

        return studentsDfRenamed;
    }

    // DataFrame here is assumed to be a custom class similar to the pandas DataFrame.
    // This custom class should have methods for getting column names and renaming columns.

    // The renameColumns method can be used as follows:
    // Assume studentsDf is a DataFrame with columns 'id', 'first', 'last', and 'age'
    // DataFrame renamedStudentsDf = DataFrameUtils.renameColumns(studentsDf);
}
```

### C++

```cpp
#include <string>
#include <unordered_map>
#include <iostream>
// Include header for DataFrame if using a third-party library

class DataFrame {
  // This is a placeholder for the actual DataFrame implementation.
  // It should provide capabilities similar to pandas.DataFrame in Python.
public:
    void rename_columns(const std::unordered_map<std::string, std::string>& column_rename_mapping) {
        // Implementation of column renaming would go here.
        // This is a stub to illustrate how it might work.
    }
};

DataFrame rename_columns(DataFrame& students_df) {
    // This function renames specific columns of a DataFrame to standardized names.
    // It takes a DataFrame by reference and returns a new DataFrame with the columns renamed.

    // Define a mapping from old column names to new standardized column names
    std::unordered_map<std::string, std::string> column_rename_mapping = {
        {"id", "student_id"},
        {"first", "first_name"},
        {"last", "last_name"},
        {"age", "age_in_years"}
    };

    // Create a new DataFrame for the results
    DataFrame students_df_renamed = students_df; // This assumes we have a copy constructor

    // Rename the columns using the provided mapping
    students_df_renamed.rename_columns(column_rename_mapping);

    return students_df_renamed;
}

// Usage example
/*
DataFrame students_df; // Assume students_df is initialized and populated with student data and the appropriate columns
DataFrame renamed_students_df = rename_columns(students_df);
*/
```

### TypeScript

```typescript
// Import statement would not be needed in TypeScript as we're not using a direct equivalent of pandas.

// Define an interface for the student object to specify the structure and types of the input data.
interface Student {
  id: number;
  first: string;
  last: string;
  age: number;
}

// Define an interface for the student object with standardized names.
interface RenamedStudent {
  student_id: number;
  first_name: string;
  last_name: string;
  age_in_years: number;
}

function renameColumns(students: Student[]): RenamedStudent[] {
    /**
     * This function renames specific properties of objects within an array to standardized names.
     *
     * @param students An array of Student objects containing student information with properties to be renamed.
     * @returns An array of RenamedStudent objects with renamed properties.
     */
    // Create a new array to hold the renamed student objects.
    const renamedStudents: RenamedStudent[] = students.map(student => {
      // Create a new object with the standardized property names.
      const renamedStudent: RenamedStudent = {
        student_id: student.id,
        first_name: student.first,
        last_name: student.last,
        age_in_years: student.age
      };
      return renamedStudent;
    });

    // Return the new array with the renamed objects.
    return renamedStudents;
}

// Example usage:
// Assume students is an array of Student objects with properties 'id', 'first', 'last', and 'age'
// const renamedStudents = renameColumns(students);
```

```python
import pandas as pd  # Import the pandas library

def rename_columns(students_df: pd.DataFrame) -> pd.DataFrame:
    """
    This function renames specific columns of a DataFrame to standardized names.

    Parameters:
    students_df (pd.DataFrame): DataFrame containing student information with columns to be renamed.

    Returns:
    pd.DataFrame: A DataFrame with renamed columns.
    """
    # Define a dictionary mapping old column names to new standardized column names
    column_rename_mapping = {
        'id': 'student_id',
        'first': 'first_name',
        'last': 'last_name',
        'age': 'age_in_years',
    }

    # Rename the columns using the provided mapping
    # Note: inplace=False ensures that the original DataFrame is not modified and a new DataFrame is returned
    students_df_renamed = students_df.rename(columns=column_rename_mapping, inplace=False)

    return students_df_renamed

# The rename_columns function can be used as follows:
# Assume students_df is a DataFrame with columns 'id', 'first', 'last', and 'age'
# renamed_students_df = rename_columns(students_df)
```

## Time and Space Complexity

The given function `renameColumns` takes a pandas DataFrame and renames several of its columns. The analysis of time and space complexity for this operation is as follows:

- **Time complexity:** The time complexity of the `rename` function in pandas primarily depends on the number of columns being renamed since the renaming operation is typically a mapping of column names. For the given function, the time complexity is $O(1)$ because the number of columns to rename is constant and does not grow with the size of the input DataFrame.
- **Space complexity:** The space complexity is $O(1)$ because the renaming operation does not allocate additional space that grows with the input size. The `inplace=True` parameter ensures that the changes are made in the original DataFrame without creating a new one, which means no additional memory proportional to the size of the DataFrame is used.