2888. Reshape Data Concatenate

Easy

Problem Description

The given problem requires concatenating two DataFrames `df1` and `df2` vertically to create a single DataFrame. Each DataFrame consists of a `student_id`, `name`, and `age`. The purpose of the vertical concatenation is to stack the rows of `df2` below the rows of `df1` while maintaining the order and structure of the columns.

To accomplish the task effectively, the solution must ensure that both DataFrames are combined such that:

- The `student_id`, `name`, and `age` columns from `df2` append directly below the corresponding columns in `df1`.
- The resulting DataFrame should not have duplicate indexes; i.e., the index should be reset or ignored during the concatenation process to ensure a unique and ordered sequence from 0 through the last row.

Intuition

The intuition behind the solution stems from understanding the pandas library's capabilities in handling DataFrames. Concatenation is a common operation in pandas that enables us to combine multiple DataFrames along a particular axis - either row-wise (axis=0) or column-wise (axis=1).

Since we want to concatenate the DataFrames vertically, we will be concatenating along `axis=0`. The `pd.concat` function from pandas is used, with two key parameters passed into it:

1. The first parameter is a list of the DataFrames to concatenate: `[df1, df2]`. This indicates to the `pd.concat` function which DataFrames are involved in the operation.
2. The second parameter is `ignore_index=True`. This parameter tells pandas to ignore the existing index of both DataFrames and assign a new incremental index starting from 0 to the resulting DataFrame. This results in a clean, non-duplicated index across the combined DataFrame.

Solution Approach

To implement the solution, the pandas library is a prerequisite as it provides us with the `DataFrame` structure and the `pd.concat` function necessary for the operation. Here's how the algorithm works:

1. Import the pandas library, which is a powerful and flexible open-source data analysis/manipulation tool written in Python. It is used here to work with the `DataFrame` structure.
2. Define the `concatenateTables` function, which takes two `DataFrames`, `df1` and `df2`, as inputs.
3. Use the `pd.concat` function inside this function. The `pd.concat` function is a well-defined method in pandas that is specifically designed to concatenate any number of `DataFrame` or `Series` objects along a particular axis – by default, `axis=0` for vertical concatenation, which is exactly what we need.
4. Enclose `df1` and `df2` within a list and pass it as the first argument to `pd.concat`. This tells pandas to concatenate these two `DataFrame` objects.
5. Set the `ignore_index` parameter of `pd.concat` to `True`. This instructs pandas to assign a new continuous index to the resulting `DataFrame`, starting from `0`.
6. The `pd.concat` function returns the concatenated `DataFrame`.
7. The `concatenateTables` function, therefore, returns the result of the `pd.concat` operation, thus completing the vertical concatenation of `df1` and `df2` into one single `DataFrame`.

By using the pandas `pd.concat` function with `ignore_index=True`, our algorithm efficiently performs a vertical stack of two separate `DataFrame` objects without the need for any complex data structures or additional patterns. This method achieves the goal with minimal code and complexity.

The implementation of this solution does not require the development of new algorithms or the use of complex data structures due to the strength of the pandas library, which abstracts these details away. The high-level abstraction provided by the `pd.concat` function substantially simplifies the task, demonstrating the power of using the right tools for data manipulation tasks in Python.

Example Walkthrough

Let's illustrate the solution approach with a specific example. Suppose we have two DataFrames `df1` and `df2` with the following data:

DataFrame `df1` contains:

student_id	name	age
1	Alice	21
2	Bob	22

DataFrame `df2` contains:

student_id	name	age
3	Charlie	23
4	David	24

Our goal is to concatenate `df1` and `df2` vertically.

1. First, we import the pandas library as it provides us with the necessary `DataFrame` structure and concatenation functions.

```
import pandas as pd
```

2. We then define the `DataFrame` instances `df1` and `df2` for our example, which correspond to the tables above.

```
df1 = pd.DataFrame({
    'student_id': [1, 2],
    'name': ['Alice', 'Bob'],
    'age': [21, 22]
})

df2 = pd.DataFrame({
    'student_id': [3, 4],
    'name': ['Charlie', 'David'],
    'age': [23, 24]
})
```

3. Next, we define the function `concatenateTables` that will take two DataFrame objects and concatenate them vertically while resetting the index.

```
def concatenateTables(df1, df2):
    return pd.concat([df1, df2], ignore_index=True)
```

4. We call the `concatenateTables` function with `df1` and `df2` as arguments.

```
resulting_df = concatenateTables(df1, df2)
```

5. The `pd.concat` function called inside the `concatenateTables` function concatenates these two DataFrames and, at the same time, ignores the original indices and assigns a new sequence of index starting from 0 to the combined DataFrame.

The concatenated DataFrame `resulting_df` now looks like this:

	student_id	name	age
0	1	Alice	21
1	2	Bob	22
2	3	Charlie	23
3	4	David	24

As we can see, the `student_id`, `name`, and `age` columns from `df2` have been appended directly below the corresponding columns in `df1`, and the index has been reset to reflect the new order. The resulting DataFrame is well structured, just as required by the problem statement. This example demonstrates the effectiveness and simplicity of the approach using the `pd.concat` function from the pandas library.

Solution Implementation

Python

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

# Define a function to concatenate two dataframes.
def concatenateTables(dataframe1: pd.DataFrame, dataframe2: pd.DataFrame) -> pd.DataFrame:
    """
    Concatenate two pandas DataFrames into a single DataFrame.

    Parameters:
    dataframe1 (pd.DataFrame): The first DataFrame to concatenate.
    dataframe2 (pd.DataFrame): The second DataFrame to concatenate.

    Returns:
    pd.DataFrame: The concatenated DataFrame with the index reset.
    """
    # Use the pd.concat method to concatenate the two DataFrames.
    # The ignore_index=True parameter will reassign new index values to the
    # resulting dataframe, ranging from 0 to the length of the new dataframe minus one.
    concatenated_dataframe = pd.concat([dataframe1, dataframe2], ignore_index=True)

    # Return the concatenated DataFrame.
    return concatenated_dataframe
```

Java

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

public class DataFrameUtils {

    /**
     * Concatenates two lists of maps simulating dataframes into a single list.
     *
     * Each map in the list represents a row, and the keys of the map represent the column names.
     *
     * @param dataframe1 The first list of maps to concatenate.
     * @param dataframe2 The second list of maps to concatenate.
     * @return A new list containing all the elements from dataframe1 followed by all the elements of dataframe2.
     */
    public static List<Map<String, Object>> concatenateTables(
            List<Map<String, Object>> dataframe1, List<Map<String, Object>> dataframe2) {

        // Create a new ArrayList to hold the concatenated data.
        List<Map<String, Object>> concatenatedList = new ArrayList<>();

        // Add all rows from the first dataframe.
        concatenatedList.addAll(dataframe1);

        // Add all rows from the second dataframe.
        concatenatedList.addAll(dataframe2);

        // Return the concatenated list of maps which simulates a dataframe.
        return concatenatedList;
    }

    // Example usage from a main method.
    public static void main(String[] args) {
        // Example DataFrames as lists of maps.
        List<Map<String, Object>> dataframe1 = new ArrayList<>();
        List<Map<String, Object>> dataframe2 = new ArrayList<>();

        // Populate the dataframes with examples (omitted for brevity).

        // Concatenate the dataframes.
        List<Map<String, Object>> combinedDataframe = concatenateTables(dataframe1, dataframe2);

        // Print out the concatenated dataframe (omitted for brevity).
    }
}
```

C++

```
#include <pandas.h> // Assuming a hypothetical C++ equivalent of the Pandas library exists.

// Define a function to concatenate two dataframes.
DataFrame ConcatenateTables(const DataFrame& dataframe1, const DataFrame& dataframe2) {
    /**
     * Concatenate two DataFrames into a single DataFrame.
     *
     * Parameters:
     * dataframe1: The first DataFrame to concatenate.
     * dataframe2: The second DataFrame to concatenate.
     *
     * Returns:
     * A concatenated DataFrame with the index reset.
     */

    // Use the Concat method to concatenate the two DataFrames.
    // Pass true for ignoreIndex to reassign new index values to the resulting DataFrame,
    // ranging from 0 to the length of the new DataFrame minus one.
    DataFrame concatenatedDataFrame = pandas::concat({dataframe1, dataframe2}, true);

    // Return the concatenated DataFrame.
    return concatenatedDataFrame;
}
```

TypeScript

```
import { DataFrame } from 'pandas-js';

// Function to concatenate two dataframes.
function concatenateTables(dataframe1: DataFrame, dataframe2: DataFrame): DataFrame {
    /**
     * Concatenate two pandas-js DataFrames into a single DataFrame.
     *
     * @param {DataFrame} dataframe1 - The first DataFrame to concatenate.
     * @param {DataFrame} dataframe2 - The second DataFrame to concatenate.
     *
     * @return {DataFrame} - The concatenated DataFrame with the index reset.
     */

    // Use the concat method from pandas-js to concatenate the two DataFrames.
    // The reset_index option will assign new index values to the resulting DataFrame,
    // ranging from 0 to the length of the new DataFrame minus one.
    let concatenatedDataFrame = dataframe1.concat(dataframe2).reset_index(drop=true);

    // Return the concatenated DataFrame.
    return concatenatedDataFrame;
}

// Usage example (assuming DataFrame objects are created properly):
// let resultDataFrame = concatenateTables(dataFrame1, dataFrame2);
```

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

# Define a function to concatenate two dataframes.
def concatenateTables(dataframe1: pd.DataFrame, dataframe2: pd.DataFrame) -> pd.DataFrame:
    """
    Concatenate two pandas DataFrames into a single DataFrame.

    Parameters:
    dataframe1 (pd.DataFrame): The first DataFrame to concatenate.
    dataframe2 (pd.DataFrame): The second DataFrame to concatenate.

    Returns:
    pd.DataFrame: The concatenated DataFrame with the index reset.
    """
    # Use the pd.concat method to concatenate the two DataFrames.
    # The ignore_index=True parameter will reassign new index values to the
    # resulting dataframe, ranging from 0 to the length of the new dataframe minus one.
    concatenated_dataframe = pd.concat([dataframe1, dataframe2], ignore_index=True)

    # Return the concatenated DataFrame.
    return concatenated_dataframe
```

Time and Space Complexity

The time complexity of `concatenateTables` when using `pd.concat` with `ignore_index=True` depends on the combined size of the input DataFrames `df1` and `df2`. If `df1` has `n` rows and `df2` has `m` rows, then the time complexity is $O(n + m)$ because each row from both DataFrames needs to be copied into the new DataFrame.