2877. Create a DataFrame from List

Easy

Leetcode Link

Problem Description

In this problem, we are tasked with constructing a DataFrame using the `pandas` library in Python. `pandas` is a powerful data manipulation library that provides numerous tools to work with structured data. A DataFrame is one of the core structures in `pandas`, designed to mimic the functionality of a database table or an Excel spreadsheet – essentially, it's a labeled two-dimensional array where each column can be of a different data type.

We are given a 2-dimensional list `student_data` where each sub-list contains two elements: the first element represents a student's ID, and the second is the student's age. The challenge is to convert this 2D list into a DataFrame with two columns. Those columns are to be named `student_id` and `age`.

The DataFrame must maintain the same row order as it was in the provided 2D list. For example, if the input is `[[1, 20], [2, 22]]`, the resulting DataFrame should look like this:

student_id	age
1	20
2	22

Creating DataFrames from various types of data is a common operation in data analysis, as it facilitates easy data manipulation, filtering, and analysis.

Intuition

The intuitive approach for solving this problem is to leverage the `DataFrame` constructor method provided by `pandas`. This method can accept various forms of input data including lists, dictionaries, and other DataFrames. When given a 2D list, each sub-list is interpreted as a row in the DataFrame.

In order to ensure that the columns of the DataFrame are correctly labeled, we specify the `columns` parameter when calling the `DataFrame` constructor, passing in a list with the desired column names: `['student_id', 'age']`.

Since the task does not involve any complex transformations, the provided code simply passes the `student_data` list and the column names to the `DataFrame` constructor and returns the newly created DataFrame. The rationale is that by directly utilizing the built-in functionality of `pandas`, we can create the required DataFrame object in an efficient and readable way.

Solution Approach

The solution approach is straightforward, utilizing the `pandas` library's data structure and functionality without employing complex algorithms or design patterns.

Here are the steps for the implementation:

1. Import the `pandas` library to gain access to the `DataFrame` construction capability.

```
import pandas as pd
```

2. Define the function `createDataframe` which takes one argument, `student_data`. This argument is expected to be a 2-dimensional list where the inner lists consist of exactly two integers, the student's ID and the student's age.

3. Inside the function, the `DataFrame` constructor of `pandas` is called with two parameters. The first parameter is the `student_data` list itself, and the second is the `columns` parameter, which is a list containing the column names, namely `['student_id', 'age']`.

```
return pd.DataFrame(student_data, columns=['student_id', 'age'])
```

4. The `DataFrame` constructor interprets each sub-list in `student_data` as a row in the DataFrame, with the first integer in the sub-list being placed under the `student_id` column, and the second under the `age` column.

By adhering to the Pythonic principle of "simple is better than complex," this solution avoids overengineering, relying on the efficient and well-tested internal mechanisms of the `pandas` library to achieve the desired result.

No special algorithm or additional data structures are required. The pattern used is one of directly mapping the input data to the DataFrame structure. This makes the code highly readable, easy to maintain, and efficient for the problem at hand.

In summary, the approach takes advantage of `pandas`'s built-in functions to transform a 2D list into a structured DataFrame with minimal code, achieving the goal with elegance and efficiency.

Example Walkthrough

Let's walk through a small example to illustrate the solution approach. Suppose we have the following 2D list which represents `student_data`:

```
student_data = [[101, 18], [102, 19], [103, 20]]
```

Each sub-list contains a student's ID and the student's age. We want to create a DataFrame from this list where each student's information is a row, with `student_id` and `age` as column headers.

By following the steps outlined in the solution approach:

1. First, we import the `pandas` library, which is essential for creating DataFrames.

```
import pandas as pd
```

2. We define a function `createDataframe` that will accept `student_data` as an argument.

```
def createDataframe(student_data):
```

3. Within the function, we call the `DataFrame` constructor of `pandas`, passing the `student_data` list and providing the column names `['student_id', 'age']` through the `columns` parameter.

```
return pd.DataFrame(student_data, columns=['student_id', 'age'])
```

4. As a result, the `DataFrame` constructor interprets each inner list from `student_data` as a row. For our example, the constructor will create a DataFrame that looks like this:

student_id	age
101	18
102	19
103	20

This process creates a pandas DataFrame with the correct row order and the specified column names. Finally, calling our `createDataframe` function with `student_data`:

```
df = createDataframe(student_data)
print(df)
```

Will output:

```
   student_id  age
0         101   18
1         102   19
2         103   20
```

Our example demonstrates the simplicity and elegance of the solution approach, relying on the powerful `pandas` library to efficiently create a DataFrame from a 2D list with the desired structure and labels.

Python Solution

```
import pandas as pd
from typing import List

# Function to create a DataFrame from student data
# Parameters:
# student_data (List[List[int]]): A list of lists, where each inner list contains student_id, and age.
# Returns:
# pd.DataFrame: A DataFrame with columns 'student_id' and 'age' created from the input data.
def create_dataframe(student_data: List[List[int]]) -> pd.DataFrame:
    # Create DataFrame using the provided student_data
    # Assign column names 'student_id' and 'age' to the DataFrame
    dataframe = pd.DataFrame(student_data, columns=['student_id', 'age'])
    # Return the created DataFrame
    return dataframe
```

Java Solution

```
import java.util.List;
import java.util.ArrayList;
import javax.swing.table.DefaultTableModel;

public class DataFrameCreator {

    /**
     * Function to create a DefaultTableModel from student data.
     * It models the concept of a DataFrame in a way that's familiar to Java users.
     *
     * Parameters:
     * studentData (List<List<Integer>>): A list of lists, where each inner list contains student_id, and age.
     * Returns:
     * DefaultTableModel: A DefaultTableModel with columns 'student_id' and 'age' created from the input data.
     */
    public static DefaultTableModel createDataFrame(List<List<Integer>> studentData) {
        // Define the column names for the table model
        String[] columnNames = {"student_id", "age"};

        // Convert the List of Lists into an array of arrays, as DefaultTableModel requires it.
        // The outer array corresponds to the rows and the inner one to the columns.
        Object[][] dataArray = studentData.stream()
                .map(list -> list.toArray(new Object[0]))
                .toArray(Object[][]::new);

        // Create the DefaultTableModel with the data array and the column names
        DefaultTableModel tableModel = new DefaultTableModel(dataArray, columnNames);

        // Return the created table model
        return tableModel;
    }

    // Example usage
    public static void main(String[] args) {
        // List of lists representing student data (student_id, age)
        List<List<Integer>> studentData = new ArrayList<>();
        studentData.add(new ArrayList<Integer>() {{
            add(1); // student_id
            add(20); // age
        }});
        studentData.add(new ArrayList<Integer>() {{
            add(2); // student_id
            add(22); // age
        }});

        // Create a DataFrame (DefaultTableModel) from the student data
        DefaultTableModel dataFrame = createDataFrame(studentData);

        // Example of printing the data
        for (int row = 0; row < dataFrame.getRowCount(); row++) {
            for (int col = 0; col < dataFrame.getColumnCount(); col++) {
                System.out.print(dataFrame.getValueAt(row, col) + " ");
            }
            System.out.println();
        }
    }
}
```

C++ Solution

```
#include <iostream>
#include <vector>
#include <string>

// Function to print a simulated DataFrame from student data
// Parameters:
// studentData (const std::vector<std::vector<int>>&): A vector of vectors,
// where each inner vector contains student_id, and age.
void createDataFrame(const std::vector<std::vector<int>>& studentData) {
    // Check if studentData is not empty and each inner vector has a size of 2
    if (!studentData.empty() && studentData[0].size() == 2) {
        // Print column names
        std::cout << "student_id" << "\t" << "age" << std::endl;

        // Iterate over the studentData to print the values
        for (const auto& student : studentData) {
            // Print student_id and age from the inner vector
            std::cout << student[0] << "\t" << student[1] << std::endl;
        }
    } else {
        // Print an error message if studentData is empty or inner vectors do not have a size of 2
        std::cerr << "Error: studentData must be a non-empty vector of vectors with a size of 2." << std::endl;
    }
}

int main() {
    // Example student data: each inner vector contains student_id and age
    std::vector<std::vector<int>> studentData = {
        {1, 20},  // Student 1 is 20 years old
        {2, 22},  // Student 2 is 22 years old
        {3, 19}   // Student 3 is 19 years old
    };

    // Create a simulated DataFrame and print the student data
    createDataFrame(studentData);

    return 0;
}
```

Typescript Solution

```
// Required type definitions for clarity
type StudentData = {
  studentId: number;
  age: number;
};

// Function to create an array of student objects from student data
// studentData parameter: An array of arrays, where each inner array contains studentId, and age.
// Returns an array of objects that represent students with properties 'studentId' and 'age'.
function createStudentArray(studentData: Array<[number, number]>): StudentData[] {
  // Map each pair of student data to an object with 'studentId' and 'age' properties
  const students: StudentData[] = studentData.map(([studentId, age]) => ({
    studentId,
    age,
  }));

  // Return the array of student objects
  return students;
}
```

Time and Space Complexity

The time complexity of creating a dataframe using `pandas.DataFrame` is generally $O(n)$ where n is the total number of elements in the input list `student_data`. Each element (a sub-list in this case) is inserted into the DataFrame during creation.

The space complexity is also $O(n)$ since the DataFrame stores all elements of the input list. Each sub-list corresponds to a row in the DataFrame, and each element within a sub-list corresponds to a cell in the DataFrame.