# 2889. Reshape Data Pivot

Easy

Leetcode Link

## Problem Description

In this problem, we have a DataFrame `weather` with three columns: `city`, `month`, and `temperature`. The `city` column contains the name of a city as a string (or 'object' type), the `month` column includes names of the months as strings, and the `temperature` column includes the recorded temperatures as integers for the given city and month. The task is to rearrange the data so that each row represents a specific month, and each column corresponds to a different city, where the cell values should be the temperatures recorded for each city-month combination.

To achieve this, the DataFrame needs to be "pivoted". Pivoting is a transformation operation that is used to reshape data in data analysis. It's similar to the "pivot table" feature in spreadsheet software like Microsoft Excel, where you rearrange data to get a more convenient or useful representation, especially for analysis purposes. The result of this pivot operation should have the months as the rows, cities as the column headers, and temperatures as the cell values in the corresponding city-month position.

## Intuition

To solve this problem, we use the `pivot` method from the Pandas library, which is designed for exactly this kind of operation. The pivot method takes three main arguments:

1. `index`: the column to set as the index of the pivoted DataFrame (the rows of our desired table). In our case, it's the 'month'.
2. `columns`: the column with values that will become the new column headers after pivoting. In our case, that's the 'city'.
3. `values`: the column containing values that will populate the new pivoted table. In this instance, it's the 'temperature'.

By setting these parameters, the `pivot` function will take every unique value in the 'month' column and create a corresponding row for each. Similarly, it will create a column for each unique value in the 'city' column. Finally, it populates these rows and columns with the corresponding 'temperature' values, based on the original rows in the DataFrame. In the end, we get a table where each row represents a each column represents a city, and the intersection of each row and column contains the temperature for that city in that month.

The `pivotTable` function we define simply calls the `pivot` method on the passed `weather` DataFrame with the appropriate parameters and returns the new pivoted DataFrame.

## Solution Approach

The solution to this problem is straightforward, thanks to the capabilities provided by the Pandas library, which is extensively used for data manipulation and analysis in Python.

Here's the approach broken down step by step:

- We identify the shape we want to achieve by looking at the input data structure. We want to pivot the DataFrame so that `month` values become the index (rows), `city` values become the columns, and `temperature` values fill the cells of the DataFrame.
- To implement this, we utilize the `pivot` function from Pandas, which is explicitly designed for reshaping or pivoting data.
- The `pivotTable` function is created, which accepts a DataFrame named `weather` as its argument. Inside this function, we call the `pivot` method on the `weather` DataFrame.
- The `pivot` method is called with three parameters:
  - `index='month'`: This sets the `month` column as the index of the new DataFrame. Each unique month now represents a different row.
  - `columns='city'`: This turns unique city names into column headers. Every unique city value from the `city` column now becomes a separate column in the new DataFrame.
  - `values='temperature'`: This specifies that the data to fill the DataFrame should be taken from the `temperature` column. This sets up our cell values according to the temperature of a city for a specific month.
- The `pivot` method returns the restructured DataFrame, which is then returned by the `pivotTable` function.
- No explicit loops, conditionals, or helper data structures are needed, since the `pivot` method takes care of the low-level data manipulation.

The beauty of using Pandas in this case is that it abstracts away much of the complexity that would come from doing such an operation manually. It internally does the heavy lifting, likely using efficient data structures like hashtables to quickly map the index and column labels to the corresponding values.

Here is the simple, yet powerful implementation of the `pivotTable` function using the pivot method:

```
import pandas as pd

def pivotTable(weather: pd.DataFrame) -> pd.DataFrame:
    return weather.pivot(index='month', columns='city', values='temperature')
```

This single line of code effectively replaces what might otherwise be a complex series of operations involving sorting, grouping, and restructuring of the data.

### Example Walkthrough

Let's say we have the following `weather` DataFrame representing temperature readings across different cities and months:

```
   city      month  temperature
0 New York  January   3
1 Chicago   January   -3
2 Los Angeles  January  15
3 New York  February  2
4 Chicago   February  -4
5 Los Angeles  February 14
```

We want to pivot this DataFrame so that we can see the temperature of each city as columns, with each row representing a different month. Here's how we can accomplish this using the solution approach:

1. We first define the pivotTable function by importing Pandas and creating a function that takes a DataFrame as its argument.
2. Inside the function, we call the pivot method of the Pandas library on our `weather` DataFrame. We set the `index` argument to `'month'`, `columns` to `'city'`, and `values` to `'temperature'`.
3. When we run our function with the example DataFrame, the pivot method rearranges the data. Each unique month becomes an index (row) of the resulting DataFrame, and each unique city becomes a column.

After we apply the `pivotTable` function to the example `weather` DataFrame, our new pivoted DataFrame will look like this:

```
          New York  Chicago  Los Angeles
month
January          3       -3           15
February         2       -4           14
```

Now, each month is a row with temperature readings in columns for New York, Chicago, and Los Angeles. The cell values are the temperatures recorded for each city-month combination.

By using the `pivot` method provided by Pandas, we avoid complex and manual data rearrangements and achieve our desired data structure in a simple and efficient manner.

## Python Solution

```
import pandas as pd  # Importing the pandas library

def pivot_table(weather_df: pd.DataFrame) -> pd.DataFrame:
    """
    Transform the given weather DataFrame into a pivot table.

    Parameters:
    weather_df -- A pandas DataFrame with at least the following columns:
                  'month', 'city', 'temperature'.

    Returns:
    A pivoted DataFrame indexed by 'month' with 'city' as columns and 'temperature' as values.
    """
    # Using the pivot method to reorganize the DataFrame.
    # 'month' is set as the index, 'city' as columns, and 'temperature' as values.
    return weather_df.pivot(index='month', columns='city', values='temperature')
```

## Java Solution

```
import java.util.*;

public class WeatherDataProcessor {

    /**
     * Transforms the given list of weather data into a pivot table-like structure.
     * This simplified version handles data represented as a list of maps, where each map is a row in the DataFrame.
     *
     * @param weatherData - A list of maps with at least the following keys: 'month', 'city', 'temperature'.
     * @return A map that represents a pivoted structure indexed by 'month' with 'city' as keys and 'temperature' as values.
     */
    public static Map<String, Map<String, Double>> pivotTable(List<Map<String, Object>> weatherData) {
        // A map to hold the pivoted structure.
        Map<String, Map<String, Double>> pivotedData = new TreeMap<>();

        // Iterate over each row (map) in the weather data.
        for (Map<String, Object> row : weatherData) {
            String month = (String) row.get("month");
            String city = (String) row.get("city");
            Double temperature = (Double) row.get("temperature");

            // If this month isn't already a key in the pivotedData map, put it with a new map as its value.
            pivotedData.putIfAbsent(month, new TreeMap<>());

            // Retrieve the inner map representing a row for the pivoted table.
            Map<String, Double> pivotRow = pivotedData.get(month);

            // Put the city and temperature in the inner map.
            pivotRow.put(city, temperature);
        }

        return pivotedData;
    }

    public static void main(String[] args) {
        // Sample usage with a simplified data set.
        List<Map<String, Object>> weatherData = new ArrayList<>();
        weatherData.add(createData("January", "New York", 3.0));
        weatherData.add(createData("January", "Los Angeles", 22.0));
        weatherData.add(createData("February", "Boston", 15.0));
        weatherData.add(createData("February", "New York", 33.5));
        // ...

        Map<String, Map<String, Double>> pivotedData = pivotTable(weatherData);

        // Code to print or otherwise use the converted data would go here.
    }

    // Helper function to create a weather data row.
    private static Map<String, Object> createData(String month, String city, Double temperature) {
        Map<String, Object> dataRow = new HashMap<>();
        dataRow.put("month", month);
        dataRow.put("city", city);
        dataRow.put("temperature", temperature);
        return dataRow;
    }
}
```

## C++ Solution

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

// Define the type alias for the pivot representation
using PivotTable = std::unordered_map<std::string, std::unordered_map<std::string, double>>;

// A simple structure to represent weather data entry
struct WeatherEntry {
    std::string month;
    std::string city;
    double temperature;
};

// Function to create a pivot table from a vector of WeatherEntries
PivotTable CreatePivotTable(const std::vector<WeatherEntry>& weather_data) {
    // The pivot table representation as a map of (month) -> (map of (city) -> (temperature))
    PivotTable pivot_table;

    // Iterate through the weather data
    for (const auto& entry : weather_data) {
        // Insert the temperature data into the pivot table
        pivot_table[entry.month][entry.city] = entry.temperature;
    }

    // Return the completed pivot table
    return pivot_table;
}

// Function to print the pivot table
void PrintPivotTable(const PivotTable& pivot_table) {
    // Iterate through the pivot table and print the values
    for (const auto& [month, city_map] : pivot_table) {
        std::cout << "Month: " << month << "\n";
        for (const auto& [city, temperature] : city_map) {
            std::cout << "  City: " << city << ", Temperature: " << temperature << "\n";
        }
    }
}

int main() {
    // Example usage:
    // Create a vector of weather data
    std::vector<WeatherEntry> weather_data = {
        {"January", "New York", -3.5},
        {"January", "Los Angeles", 13.7},
        {"February", "New York", -1.3},
        {"February", "Los Angeles", 15.9},
    };

    // Create the pivot table
    PivotTable pivot_table = CreatePivotTable(weather_data);

    // Print the pivot table
    PrintPivotTable(pivot_table);

    return 0;
}
```

## Typescript Solution

```
type WeatherEntry = {
    month: string;
    city: string;
    temperature: number;
};

type PivotedResult = {
    [month: string]: {
        [city: string]: number;
    }
};

/**
 * Transforms the given weather data into a pivot table structure.
 *
 * @param weatherData - An array of objects with at least the 'month', 'city', and 'temperature' properties.
 * @returns A pivoted data structure indexed by 'month' with 'city' as properties and 'temperature' as values.
 */
function pivotTable(weatherData: WeatherEntry[]): PivotedResult {
    const pivotedResult: PivotedResult = {};

    weatherData.forEach(entry => {
        if (!pivotedResult[entry.month]) {
            pivotedResult[entry.month] = {};
        }

        pivotedResult[entry.month][entry.city] = entry.temperature;
    });

    return pivotedResult;
}
```

## Time and Space Complexity

The time complexity of the pivot operation in pandas depends on the size of the input DataFrame `weather`. Assuming that the input DataFrame has `n` rows, then the complexity of the pivot operation primarily involves sorting the `index` and `columns`, which can be considered as O(n log n) in the average case for most sorting algorithms, including those used in Pandas operations.

Since each value from the `temperature` column is placed into a new cell in the resulting pivot table, the space needed for the output DataFrame is proportional to the number of unique `month` and `city` pairs. If there are `u` unique months and `c` unique cities, the resultant DataFrame could have up to `u * c` cells for the `temperature` values (excluding the memory required to store the DataFrame's index and column structures).

Thus, the space complexity can be expressed as O(m * c).

In summary:

- Time Complexity: $O(n \log n)$
- Space Complexity: $O(u * c)$