2880. Select Data

Easy

Leetcode Link

Problem Description

The given problem presents a DataFrame named `students`, which represents a simplified table structure as often found in databases. The DataFrame contains three columns: `student_id`, `name`, and `age`. Each row in the DataFrame represents a unique student with their corresponding ID, name, and age. The task is to query this DataFrame and extract information specifically for the student with the ID `101`. The output should be a new DataFrame containing only the name and age of that particular student, discarding other columns and rows that do not match the specified student ID.

Intuition

The intuition behind the solution emerges from understanding how DataFrames work in pandas, a popular data manipulation library in Python. To extract specific information from a DataFrame, the following operations are typically involved:

1. **Filtering**: We need to filter out the rows that do not meet our criteria, which is in this case the row where the `student_id` equals `101`. Filtering in pandas is often done by generating a boolean mask that is `True` for rows that match the condition and `False` for those that do not. Applying this mask to the DataFrame yields a new DataFrame comprised only of the rows where the mask is `True`.

2. **Selecting Columns**: After isolating the row or rows that meet our condition, we need to select only the columns of interest. In this problem, our interest lies in the `name` and `age` columns. This is done by specifying a list of the desired column names to the DataFrame after filtering.

By combining these two operations, we obtain the solution to the problem. We apply a filter to keep only the row where `student_id` is `101`, and immediately after that, we specify that we want to continue with just the columns `name` and `age`.

Here's a step-by-step intuition for the provided solution:

1. `students['student_id'] == 101` creates a boolean mask that only evaluates to `True` for the row where `student_id` is `101`.
2. `students[students['student_id'] == 101]` applies the mask to the `students` DataFrame, yielding a DataFrame that contains only the row with `student_id` `101`.
3. `[['name', 'age']]` is a list of column names indicating our intent to select only these columns from the filtered DataFrame.
4. Placing `[['name', 'age']]` after the filtered DataFrame completes the operation by returning only the desired columns for the student with `student_id` `101`.

Through these steps, we arrive at the final, succinct solution that performs the required operation effectively using pandas library capabilities.

Solution Approach

The implementation of the solution follows a straightforward approach using pandas, a powerful data manipulation library in Python, which is ideal for working with tabular data structures like DataFrames.

Here's the breakdown of the solution step by step:

1. **Filtering Rows**: The key operation begins with filtering the DataFrame to select only the row where the `student_id` is 101. This is achieved by using a boolean expression `students['student_id'] == 101`. This expression checks each `student_id` in the DataFrame against the value `101` and returns a Series of boolean values (`True` or `False`). This Series acts as a mask over the DataFrame.

2. **Applying the Filter**: The boolean mask is then applied to the `students` DataFrame. This is done by passing the mask as an indexer to the DataFrame: `students[students['student_id'] == 101]`. Pandas filters out any rows for which the mask is `False`, leaving only the rows where the mask is `True`, which in this case should be only the row with `student_id` 101 if `student_id` is unique.

3. **Selecting Columns**: After we have filtered the DataFrame to isolate the row with the desired `student_id`, we proceed to select only the columns that we want to include in our final output. This is done by passing a list of the desired column names to the DataFrame: `[['name', 'age']]`. This list tells pandas to keep only these columns and discard any others.

4. **Combining Operations**: Lastly, the row filtering and column selection operations are combined in a single line of code to produce the final DataFrame. This is the expression `students[students['student_id'] == 101][['name', 'age']]`. It filters the rows and selects the columns in one step, resulting in a DataFrame that contains only the `name` and `age` of the student with `student_id` 101.

In terms of data structures, the solution operates entirely on the `DataFrame` object, which is the main data structure in pandas. DataFrames are designed to mimic SQL table-like functionalities with rows and columns, where each column can have different data types.

The pattern used in this solution is common in pandas for querying and subsetting data. It is analogous to SQL's `SELECT ... FROM ... WHERE ...` statements, with the main difference being the syntax and the fact that we're using pandas' methods and indexing capabilities instead of SQL queries.

The provided solution, encapsulated in the `selectData` function, showcases the elegant and Pythonic approach to dealing with DataFrame operations and signifies the strength of pandas when it comes to data manipulation tasks.

Example Walkthrough

Let's consider a small dataset to demonstrate the workings of the solution approach outlined above.

Assume we have the following `students` DataFrame:

student_id	name	age
100	Alice	23
101	Bob	24
102	Charlie	22

And we want to extract the information specifically for the student with the `student_id` `101`.

Filtering Rows

The first step is to create a boolean mask that will help us filter the rows:

```python
mask = students['student_id'] == 101
```

This mask evaluates to:

student_id	name	age	mask
100	Alice	23	False
101	Bob	24	True
102	Charlie	22	False

Applying the Filter

Next, we apply the mask to the `students` DataFrame:

```python
filtered_students = students[mask]
```

This results in a temporary DataFrame that holds only the row(s) where the mask is `True`:

student_id	name	age
101	Bob	24

Selecting Columns

Now we proceed to select only the columns `name` and `age`:

```python
selected_data = filtered_students[['name', 'age']]
```

The resulting DataFrame is:

name	age
Bob	24

Combining Operations

We can combine the above steps into a single line of code in order to achieve our desired output:

```python
result = students[students['student_id'] == 101][['name', 'age']]
```

And `result` holds the final DataFrame:

name	age
Bob	24

This is exactly the output we were aiming for: a new DataFrame containing just the `name` and `age` of the student with `student_id` `101`. The transformation process filters out other students and discards irrelevant columns, employing pandas' slicing and filtering capabilities to retrieve the needed subset of data efficiently.

Python Solution

```python
import pandas as pd

def select_data(students_df: pd.DataFrame) -> pd.DataFrame:
    # This function filters the provided DataFrame for a specific student by ID (101)
    # and returns only the 'name' and 'age' columns.

    # Filter the DataFrame for the student with 'student_id' equal to 101
    filtered_students = students_df[students_df['student_id'] == 101]

    # Select only the 'name' and 'age' columns
    selected_columns = filtered_students[['name', 'age']]

    return selected_columns

```

Java Solution

```java
import java.util.List;
import java.util.stream.Collectors;

// Define a Student class to represent student data with `studentId`, `name`, and `age` as properties.
class Student {
    int studentId;
    String name;
    int age;

    // Constructor for Student class
    public Student(int studentId, String name, int age) {
        this.studentId = studentId;
        this.name = name;
        this.age = age;
    }

    // Getter for studentId
    public int getStudentId() {
        return studentId;
    }

    // Getter for name
    public String getName() {
        return name;
    }

    // Getter for age
    public int getAge() {
        return age;
    }
}

// Define a class to represent operations on student data.
class DataSelector {

    // This method filters a list of students for a specific student by ID (101) and returns their 'name' and 'age'.
    public List<Student> selectData(List<Student> students) {
        // Filter the list for the student with 'studentId' equal to 101
        List<Student> filteredStudents = students.stream()
                .filter(student -> student.getStudentId() == 101)
                .collect(Collectors.toList());

        // Return only the 'name' and 'age' columns. In Java, we have to return the whole Student object,
        // but users of the method should only use the name and age properties if adhering to original intent.
        return filteredStudents;
    }
}

```

C++ Solution

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

// Assume there's a struct to represent a student
struct Student {
    int student_id;
    std::string name;
    int age;
};

// This function filters students by their ID and collects names and ages
std::vector<std::pair<std::string, int>> select_data(const std::vector<Student>& students) {
    std::vector<std::pair<std::string, int>> selected_data; // Pair will hold the name and age

    // Iterate through the list of students
    for (const auto& student : students) {
        // Check if the student has the ID 101
        if (student.student_id == 101) {
            // Add the student's name and age to the selected data
            selected_data.emplace_back(student.name, student.age);
        }
    }

    // Return the filtered and selected data
    return selected_data;
}

// Example usage
int main() {
    // Create a sample list of students
    std::vector<Student> students = {
        {100, "John Doe", 20},
        {101, "Jane Smith", 21},
        {102, "Bob Johnson", 22}
    };

    // Get the data for student with ID 101
    std::vector<std::pair<std::string, int>> data_for_student = select_data(students);

    // Print the result
    for (const auto& data : data_for_student) {
        std::cout << "Name: " << data.first << ", Age: " << data.second << std::endl;
    }

    return 0;
}

```

Typescript Solution

```typescript
// Import the necessary library for data handling
import * as pd from 'pandas';

// This function filters a DataFrame for a specific student by ID (101)
// and returns a new DataFrame containing only the 'name' and 'age' columns.
function selectData(studentsDf: pd.DataFrame): pd.DataFrame {

    // Filter the DataFrame for the student with a 'studentId' equal to 101
    const filteredStudents: pd.DataFrame = studentsDf.filter((row: any) => row['studentId'] === 101);

    // Select only the 'name' and 'age' columns from the filtered DataFrame
    const selectedColumns: pd.DataFrame = filteredStudents[['name', 'age']];

    // Return the DataFrame containing the selected columns
    return selectedColumns;
}

```

Time and Space Complexity

Time Complexity

The `selectData` function primarily involves the selection of rows based on a condition (`students['student_id'] == 101`) and the selection of specific columns (`['name', 'age']`). The time complexity of filtering the DataFrame by a condition is typically O(n), where n is the number of rows in the DataFrame, as each row has to be checked against the condition. Furthermore, selecting specific columns from the DataFrame is an O(1) operation since it is a simple indexing operation that does not require iteration over all rows or columns. Hence, the overall time complexity of the function is `O(n)`.

Space Complexity

Regarding space complexity, the function creates a new DataFrame that only contains the filtered rows with the selected columns. The space used will depend on the number of rows that satisfy the condition. However, since we are filtering by a unique `student_id`, at most, one row will satisfy the condition. The space complexity of the output will be O(1) as the filtered result is independent of the size of the input DataFrame. Moreover, `pandas` may internally optimize memory usage depending on the version and the configuration, but this does not affect the space complexity of the algorithm used in `selectData`. Thus, the space complexity of the function is `O(1)`.