2891. Method Chaining

Easy

Problem Description

descending order so the heaviest animals appear first. The DataFrame is structured with columns for each attribute of the animals, and each row corresponds to a distinct animal. We are interested in filtering the rows based on a particular column (weight) and then manipulating the DataFrame to return a specific subset of its data (the name column).

strictly greater than 100 kilograms. After finding the relevant animals, we need to sort this list by the animals' weight in

In this problem, we are given a DataFrame named animals that contains information about different animals, including their name,

species, age, and weight. Our task is to write a Python function that uses Pandas to list the names of animals that have a weight

In the context of the problem, we are also asked to leverage method chaining in Pandas which allows us to execute multiple operations in a compact and readable one-liner. This is efficient and elegant, minimizing the need for creating temporary

variables and making the code easier to understand at a glance. ntuition

The intuition behind the solution involves two main steps, which we can implement in Pandas through method chaining: Filtering: First, we need to filter the DataFrame to include only those rows where the animals' weight is more than 100 kilograms. In Pandas, this is achieved with a boolean indexing operation, where we compare the weight column against the

value 100. The comparison generates a boolean Series that we use to filter out rows that don't meet the condition.

- Sorting and Selecting Columns: After filtering the rows, we should sort them by the weight column in descending order to meet the requirement of listing heavier animals first. The sort_values function in Pandas can be used for this purpose by specifying the ascending=False parameter. Once sorted, we need to select the name column as this is what we want to return. By indexing the DataFrame with a list of column names (['name']), we can select the required column(s).
- **Solution Approach** The solution is implemented using a Python function that expects a Pandas DataFrame as an input and returns a DataFrame as an output. Here is a step-by-step breakdown of the one-liner solution within the function:

Sorting Values:

In animals [animals ['weight'] > 100], we perform a boolean indexing operation. This creates a boolean Series by comparing each value in the weight column to the number 100. This Series is then used to filter the DataFrame, keeping only the rows where the condition (weight greater than 100) is True.

The sort_values('weight', ascending=False) method is chained after the boolean indexing. This call sorts the filtered

DataFrame by the weight column in descending order (ascending=False). The resulting DataFrame maintains only the filtered rows, now sorted so that the heaviest animals are at the top.

Selecting Columns:

Filtering with Boolean Indexing:

constrains the output to contain only the names of the heavy animals, as requested. By following these steps, the function returns the names of animals that weigh more than 100 kilograms, sorted by their weight in descending order. The entire process is a demonstration of method chaining in Pandas and showcases how expressive and

The last part of the chain [['name']] selects only the name column of the sorted DataFrame. This indexing operation

efficient this approach can be for data manipulation tasks. The algorithm's complexity essentially depends on the filtering and sorting operations. The filtering runs in O(n) time, where n is

species

kangaroo

elephant

rabbit

cow

fish

name

Daisy

Bubbles

Boomer

Zeus

Fluffy

weight

200

22

85

500

4

Using this Series to filter the DataFrame, we get:

age

5

10

age

10

5

weight

200

500

weight

500

200

age

5

1

3

10

2

Filtering with Boolean Indexing:

True

False

False

the number of rows in the DataFrame, as it involves checking each weight once. Sorting can be expected to run on average in 0(n log n) time. Consequently, the overall complexity of the operation would be dominated by the sorting step, resulting in an average time complexity of O(n log n).

Example Walkthrough Let's illustrate the solution approach with a small example: Suppose we have the following DataFrame named animals:

We want to extract the names of animals weighing more than 100 kilograms, sorted by their weight in descending order. **Step-by-Step Walkthrough**

We apply the boolean indexing operation animals['weight'] > 100 to create the following boolean Series:

True Zeus Fluffy **False**

Daisy

Bubbles

name

Daisy

Zeus

name

Zeus

Daisy

name

Zeus

Daisy

Expected Output:

name

Zeus

Daisy

Boomer

Sorting Values: We then sort the filtered results by the weight column in descending order:

species

elephant

cow

Selecting Columns:

elephant

species

cow

Finally, we select just the name column:

The Code def heavy_animals(df): return df[df['weight'] > 100].sort_values('weight', ascending=False)[['name']] # Now, let's use our `animals` DataFrame as an input to our function result = heavy_animals(animals) print(result)

Solution Implementation

import java.util.Comparator;

import java.util.stream.Collectors;

this.name = name;

this.weight = weight;

// Class to represent an animal with a name and weight

List<Animal> heavyAnimals = animals.stream()

for (Animal animal : heavyAnimals) {

public Animal(String name, int weight) {

import java.util.List;

class Animal {

String name;

int weight;

This output matches our criteria, listing the names of the animals that weigh more than 100 kilograms, sorted in descending order by weight. With the above approach, we are able to efficiently filter, sort, and select the necessary data using method chaining in

Pandas.

Python

```
import pandas as pd # Importing the pandas library with the alias 'pd'
# Define a function that finds animals weighing more than 100 units
def find_heavy_animals(animals_df: pd.DataFrame) -> pd.DataFrame:
```

Java import java.util.ArrayList; import java.util.Collections;

.filter(animal -> animal.getWeight() > 100)

.collect(Collectors.toList());

```
// Getters...
   public String getName() {
       return name;
    public int getWeight() {
        return weight;
   // You might also want to add setters and other utility methods if needed.
public class AnimalWeightFinder {
   // Function to find animals weighing more than 100 units
    public static List<String> findHeavyAnimals(List<Animal> animals) {
```

// Filter the list to include only animals weighing more than 100 units

// Sort the list of heavy animals by weight in descending order

Collections.sort(heavyAnimals, new Comparator<Animal>() {

return a2.getWeight() - a1.getWeight();

// Extract just the names of the sorted heavy animals

sortedHeavyAnimalNames.add(animal.getName());

// Return the list of sorted heavy animal names

List<String> sortedHeavyAnimalNames = new ArrayList<>();

// Comparator function for sorting Animals by weight in descending order

std::vector<std::string> FindHeavyAnimals(const std::vector<Animal> &animals) {

std::vector<std::string> heavy_animals_names; // Vector to keep names of heavy animals

// Iterate over the input vector and select animals that weigh more than 100 units

import { DataFrame } from 'pandas-js'; // Importing the DataFrame class from 'pandas-js'

* Identify and return an array with the names of animals that weigh more than 100 units.

* @param animalsDf A DataFrame with columns including 'name' and 'weight'.

// Filter the DataFrame to include only animals weighing more than 100 units

return sorted_heavy_animals # Return the sorted DataFrame with animal names

Time and Space Complexity

* @return An array with the names of heavy animals, sorted by weight.

bool compareByWeightDescending(const Animal &a, const Animal &b) {

// Define a function that finds animals weighing more than 100 units

heavy_animals_names.push_back(animal.name);

* The result is sorted by weight in descending order.

function findHeavyAnimals(animalsDf: DataFrame): string[] {

return a.weight > b.weight;

for (const auto &animal : animals) {

if (animal.weight > 100) {

public int compare(Animal a1, Animal a2) {

Identify and return a DataFrame with the names of animals that weigh more than 100 units.

Sort the filtered DataFrame by weight in descending order and select only the 'name' column

sorted_heavy_animals = heavy_animals.sort_values('weight', ascending=False)[['name']]

:param animals_df: A pandas DataFrame with columns including 'name' and 'weight'.

:return: A DataFrame with the names of heavy animals, sorted by weight.

Filter the DataFrame to include only animals weighing more than 100 units

return sorted_heavy_animals # Return the sorted DataFrame with animal names

The result is sorted by weight in descending order.

heavy_animals = animals_df[animals_df['weight'] > 100]

```
return sortedHeavyAnimalNames;
    // Main method for demonstration purposes (Optional)
    public static void main(String[] args) {
       // List of animals (simulating a DataFrame)
       List<Animal> animals = new ArrayList<>();
       animals.add(new Animal("Elephant", 1200));
       animals.add(new Animal("Tiger", 150));
       animals.add(new Animal("Rabbit", 5));
       animals.add(new Animal("Bear", 600));
       // Find and print names of heavy animals
       List<String> heavyAnimalNames = findHeavyAnimals(animals);
        System.out.println("Heavy Animals: " + heavyAnimalNames);
C++
#include <vector>
#include <algorithm>
#include <string>
// Assuming an Animal structure defined like this:
struct Animal {
    std::string name;
    double weight;
};
```

```
// Sort the names of the heavy animals by their weights in descending order
// Since we only have the names in the vector, we would need to reference back to the original vector
// Therefore, this step might require either keeping weights in the pair with names OR having a map for weights
// Here we assume we only sort by name just for demo purposes
std::sort(heavy_animals_names.begin(), heavy_animals_names.end(), [&](const std::string &name1, const std::string &name2) {
   double weight1 = std::find_if(animals.begin(), animals.end(), [&](const Animal &animal) {
        return animal.name == name1;
   })->weight;
   double weight2 = std::find_if(animals.begin(), animals.end(), [&](const Animal &animal) {
        return animal.name == name2;
   })->weight;
   return weight1 > weight2;
});
return heavy_animals_names; // Return the vector containing sorted heavy animals names
```

const heavyAnimals = animalsDf.filter((row: any) => row.get('weight') > 100);

TypeScript

/**

*/

```
// Sort the filtered DataFrame by weight in descending order
      const sortedHeavyAnimals = heavyAnimals.sort_values({ by: 'weight', ascending: false });
      // Select only the 'name' column and convert it to an array
      const heavyAnimalNames: string[] = sortedHeavyAnimals.get('name').to_json({ orient: 'records' });
      return heavyAnimalNames; // Return the array with animal names
  // Note that pandas-js might not have exact one-to-one mapping with the Python pandas library.
  // The provided functionality is based on typical usage of a JavaScript DataFrame library.
  // It is assumed that the 'pandas-js' library has a similar API to that of Python's pandas.
import pandas as pd # Importing the pandas library with the alias 'pd'
# Define a function that finds animals weighing more than 100 units
def find_heavy_animals(animals_df: pd.DataFrame) -> pd.DataFrame:
   Identify and return a DataFrame with the names of animals that weigh more than 100 units.
   The result is sorted by weight in descending order.
    :param animals_df: A pandas DataFrame with columns including 'name' and 'weight'.
    :return: A DataFrame with the names of heavy animals, sorted by weight.
   # Filter the DataFrame to include only animals weighing more than 100 units
   heavy_animals = animals_df[animals_df['weight'] > 100]
   # Sort the filtered DataFrame by weight in descending order and select only the 'name' column
   sorted_heavy_animals = heavy_animals.sort_values('weight', ascending=False)[['name']]
```

The time complexity of the findHeavyAnimals function involves several steps. First, we filter the animals DataFrame, which requires 0(n) time, where n is the number of rows in animals. Then we sort this filtered DataFrame, which takes 0(m log m) time where m is the number of rows with weight greater than 100. Finally, we slice the DataFrame to include only the name column, which is an O(m) operation. Therefore, the overall time complexity is $O(n + m \log m)$.