Easy

Problem Description

are duplicate records based on the email column. The task is to remove these duplicate rows, but crucially, we must keep only the first occurrence of each email. The DataFrame may contain unique customer_id values and name values, but some emails are associated with more than one customer. The goal is to return a DataFrame where all email addresses appear only once, preserving the record of the first customer who had that email. ntuition

The problem presents a DataFrame named customers with columns customer_id, name, and email. We are informed that there

the subset parameter, allowing us to specify on which columns to check for duplicates. In this case, we'll set the subset to ['email'], so the function will only consider the email column for finding duplicates. By default, drop_duplicates keeps the first occurrence of a duplicated row, which aligns perfectly with our requirements. We do not need to set the keep argument as its default value is 'first'. We can also safely ignore the inplace parameter, or set it to

The drop_duplicates function from the pandas library immediately comes to mind for this task. This function is tailor-made for

situations like this, where we need to remove duplicate rows based on one or multiple column values. The function comes with

False, as we want to return a new DataFrame rather than modify the original customers DataFrame in place. This provides a simple and efficient solution to the problem that can be implemented and understood with minimal code. **Solution Approach**

The solution is straightforward due to the robust capabilities of the Pandas library in Python, which is designed to handle and

The key steps of the solution are:

Import the Pandas library to use its functionalities. Define a function dropDuplicateEmails, which takes a customers DataFrame as its argument.

The drop_duplicates method identifies and removes duplicate rows with the following considerations:

The method returns a new DataFrame with the duplicates removed, which we immediately return from our function.

Use the drop_duplicates method available in the Pandas library to remove duplicate rows based on specific column values.

parameter explicitly.

exact purpose.

complexity.

• The subset parameter specifies the columns to consider for identifying duplicate rows. In our case, this is set to ['email'] so that the method looks for duplicates only in the email column.

Suppose we have a small customers DataFrame that looks like this:

alice@example.com

bob@example.com

alice@example.com

bob@example.com

charlie@example.com

email

Following the solution, this is what we would do:

2. Create the dropDuplicateEmails function.

1. Import the pandas library so that we can work with DataFrames.

3. Use the drop_duplicates method on the customers DataFrame with <a href="subset=['email'].

and 2) and discards the other occurrences (the rows with customer_id 3 and 5).

The DataFrame returned by the <a href="mailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropDuplicateEmailto:dropD

name

Alice

Bob

Charlie

David

manipulate data in DataFrame structures.

- By default, the keep parameter is set to 'first', which means that if duplicates are found, the first occurrence is kept while the subsequent duplicates are removed. This behavior is exactly what we need to solve this problem, so we do not need to specify this
- Therefore, the core algorithm involves no complex loops or conditionals due to the high-level abstraction provided by Pandas. The data structure used is the DataFrame itself, and the pattern applied is the direct use of a library function designed for this
- The implementation is as follows: import pandas as pd
- dropDuplicateEmails(customers: pd.DataFrame) -> pd.DataFrame: return customers.drop_duplicates(subset=['email'])

Example Walkthrough Let's walk through a small example to illustrate how the solution approach effectively removes duplicate email addresses and retains only the first occurrence of each.

This elegant and concise implementation leverages Pandas' capabilities to solve the problem with minimal code and time

Alice 3

customer_id

customer_id

Python

Java

Parameters:

4

5

Here, we can see that Alice appears twice with the same email (alice@example.com), and so does Bob with his email (bob@example.com). We would like to have only the first occurrence of each unique email address.

When we apply the function dropDuplicateEmails to our DataFrame, the drop_duplicates method processes the DataFrame as

• When it finds the duplicate values alice@example.com and bob@example.com, it keeps the first occurrence of each (the rows with customer_id 1

follows: • It checks the email column for duplicate values since we've set subset=['email'].

Alice alice@example.com

import pandas as pd # Importing the pandas library

dropDuplicateEmails(customers: pd.DataFrame) -> pd.DataFrame:

Use the drop duplicates method on the 'customers' dataframe,

unique_customers = customers.drop_duplicates(subset=['email'])

Return the resulting DataFrame with duplicates removed.

Remove duplicate rows from the customers DataFrame based on the 'email' column.

customers (pd.DataFrame): DataFrame containing customer data with an 'email' column.

specifying 'email' as the subset to identify duplicates by the 'email' column only.

// Equals and hashCode methods are overridden to use 'email' field for comparison

// and to ensure consistent behavior when used as a key in a HashMap.

name

email

- Bob bob@example.com Charlie charlie@example.com
- and the use of Pandas' high-level functionality, which makes the code clean, readable, and efficient. Solution Implementation

So the final DataFrame correctly contains only one record for each email address, and the first customers with those emails have

been kept. The duplicate rows have been removed, achieving our desired result. The elegance of this solution lies in its simplicity

Returns: pd.DataFrame: A new DataFrame without duplicate emails.

import java.util.Objects; class Customer { String email: // Other customer fields can be added here

public Customer(String email) {

public boolean equals(Object o) {

// Define a custom comparison operator

// Define a custom equality operator

return first.getEmail() < second.getEmail();</pre>

return first.getEmail() == second.getEmail();

DataFrame dropDuplicateEmails(DataFrame& customers) {

// hence the need for the sort operation beforehand.

// Erase the non-unique elements from the container

// Return the resulting DataFrame with duplicates removed.

function dropDuplicateEmails(customers: Customer[]): Customer[] {

Return the resulting DataFrame with duplicates removed.

customers.erase(lastUnique, customers.end());

* @returns A new array without duplicate emails.

bool compareByEmail(const Customer& first, const Customer& second) {

bool equalByEmail(const Customer& first, const Customer& second) {

// Remove duplicate rows from the customers DataFrame based on the 'email' column.

// and 'Customer' has a method 'getEmail()' to access the 'email' attribute.

// First, sort the customers by email using the custom comparison operator

// unique algorithm will require the duplicates to be next to each other

* Remove duplicate rows from the customers array based on the 'email' property.

* @param customers - Arrav containing customer objects with an 'email' property.

auto lastUnique = std::unique(customers.begin(), customers.end(), equalByEmail);

// Assuming 'customers' is a DataFrame containing 'Customer' objects

std::stable_sort(customers.begin(), customers.end(), compareByEmail);

// Then, use the unique algorithm with a custom equality operator

if (this == o) return true;

if (!(o instanceof Customer)) return false;

this.email = email;

public String getEmail() {

return email;

return unique_customers

import java.util.HashMap;

import java.util.ArrayList;

import java.util.Map;

import java.util.List;

// Constructor

// Email getter

@Override

```
Customer customer = (Customer) o:
        return Objects.equals(email, customer.email);
    @Override
    public int hashCode() {
        return Objects.hash(email);
    // A toString method for easy printing of customer information.
    @Override
    public String toString() {
        return "Customer{" +
                "email='" + email + '\'' +
public class DuplicateEmailsRemover {
    /**
     * Remove duplicate rows from the list of customers based on the 'email' field.
     * @param customers (List<Customer>): A list containing customer objects.
     * @return List<Customer>: A new list without duplicate emails.
    public static List<Customer> dropDuplicateEmails(List<Customer> customers) {
        // Use a HashMap to track unique emails.
        Map<String, Customer> uniqueCustomersMap = new HashMap<>();
        // Iterate over the list of customer objects
        for (Customer customers) {
            // If the email has not been seen before, add the customer to the map.
            if(!uniqueCustomersMap.containsKey(customer.getEmail())) {
                uniqueCustomersMap.put(customer.getEmail(), customer);
        // Return the unique customers as a new list (values of the map).
        return new ArrayList<>(uniqueCustomersMap.values());
    // A main method to test the functionality
    public static void main(String[] args) {
        // Create a list of customers with some duplicate emails
        List<Customer> customers = new ArrayList<>():
        customers.add(new Customer("alice@example.com"));
        customers.add(new Customer("bob@example.com"));
        customers.add(new Customer("charlie@example.com"));
        customers.add(new Customer("alice@example.com")); // Duplicate
        // Remove duplicates
        List<Customer> uniqueCustomers = dropDuplicateEmails(customers);
        // Print out the unique customer list
        uniqueCustomers.forEach(System.out::println);
C++
#include <iostream>
#include <vector>
#include <string>
#include <algorithm> // For std::unique and std::stable sort
#include "DataFrame.h" // Assuming DataFrame is a fictional class that needs to be included
```

interface Customer { [key: string]: any; // An index signature to allow any string as a key and its value can be anything email: string; // Ensure that 'email' is always a string

/**

TypeScript

return customers;

```
const seenEmails = new Set<string>(); // To track already encountered emails
    // Use filter to exclude duplicates
    const uniqueCustomers = customers.filter(customer => {
        // If seenEmails does not have this email, add it and keep the customer
        if (!seenEmails.has(customer.email)) {
            seenEmails.add(customer.email);
            return true;
        // Otherwise, it's a duplicate; exclude it by returning false
        return false;
   });
    // Return the filtered array without duplicate emails
    return uniqueCustomers;
import pandas as pd # Importing the pandas library
def dropDuplicateEmails(customers: pd.DataFrame) -> pd.DataFrame:
    Remove duplicate rows from the customers DataFrame based on the 'email' column.
    Parameters:
    customers (pd.DataFrame): DataFrame containing customer data with an 'email' column.
    Returns:
    pd.DataFrame: A new DataFrame without duplicate emails.
    # Use the drop duplicates method on the 'customers' dataframe,
    # specifying 'email' as the subset to identify duplicates by the 'email' column only.
    unique_customers = customers.drop_duplicates(subset=['email'])
```

The time complexity of the dropDuplicateEmails function primarily depends on the drop_duplicates method of the pandas DataFrame, which in turn depends on the size of the input DataFrame.

Time Complexity

Time and Space Complexity

return unique_customers

For pandas drop_duplicates method, the time complexity is generally O(n), where n is the number of rows in the DataFrame because it needs to process each row to check for duplicates.

Space Complexity

So, the time complexity for the dropDuplicateEmails function is also O(n).

The space complexity for drop_duplicates includes the space required to hold the DataFrame and the temporary data structures used to identify duplicates. Typically, it is also 0(n) since a new DataFrame is constructed to store the result without duplicates.