How to Easily Identify and Remove Duplicates in Pandas DataFrames

Authored by

stats writer

December 2, 2025

ARABPSYCHOLOGY.COM

RECOMMENDED CITATION

stats writer (2025). *How to Easily Identify and Remove Duplicates in Pandas DataFrames*. PSYCHOLOGICAL SCALES. Retrieved from https://scales.arabpsychology.com/?p=103495

Managing data quality is a critical task in data science, and identifying redundant records is often the first step toward effective data cleaning. When working with Python's powerful data manipulation library, Pandas DataFrame, locating these duplicate rows is straightforward using specialized methods. This comprehensive guide details how to efficiently find and manage duplicate entries within your datasets.

The primary tool for this operation is the duplicated() method. When applied to a DataFrame, this method returns a Boolean series, where each element corresponds to a row in the original DataFrame, indicating whether that row is a duplicate of a previously observed row. Understanding how to leverage its optional parameters, such as `subset` and `keep`, is essential for granular control over duplicate detection.

Understanding the `duplicated()` Method Syntax

The core of duplicate detection in Pandas revolves around the `.duplicated()` method, which is applied directly to the DataFrame. This method is highly flexible, allowing you to check for exact duplicates across all columns or constrain the check to only a selection of columns.

When used without any arguments, `df.duplicated()` performs a check across every column, marking a row as `True` if it exactly matches any previous row in the dataset. This default behavior assumes that the first occurrence of a set of values is the original, and all subsequent matching rows are considered duplicates. The output is always a Boolean series that can be used for indexing and filtering the original DataFrame.

#find duplicate rows across all columns
duplicateRows = df

#find duplicate rows across specific columns
duplicateRows = df)]

Setting up the Example Pandas DataFrame

To demonstrate the practical application of the duplicate finding methods, we will initialize a sample Pandas DataFrame representing basic sports statistics. This dataset contains intentional duplicates that we will work to identify using various parameters of the `.duplicated()` method. Pay close attention to rows 0 and 1, and rows 6 and 7, as these contain exact matches.

We import the necessary Pandas library and define the data dictionary, creating columns for 'team', 'points', and 'assists'.

ARABPSYCHOLOGY.COM

import pandas as pd

#create DataFrame
df = pd.DataFrame({'team': ,
'points': ,
'assists': })

#view DataFrame
print(df)

team points assists
0 A 10 5
1 A 10 5
2 A 12 7
3 A 12 9
4 B 15 12
5 B 17 9
6 B 20 6
7 B 20 6

Example 1: Finding Duplicates Across All Columns

When you need to identify rows that are entirely redundant, you utilize the duplicated() method without specifying the `subset` parameter. This approach checks for complete matches across every column in the DataFrame. By default, the first instance of a repeating row is considered the original (marked as `False` in the underlying Boolean series), and any subsequent matching rows are identified as duplicates (marked as `True`).

#identify duplicate rows
duplicateRows = df

#view duplicate rows
duplicateRows

team points assists
1 A 10 5
7 B 20 6

As confirmed by the output, rows with index 1 and 7 are identified as duplicates. Row 1 matches row 0 (A, 10, 5), and row 7 matches row 6 (B, 20, 6). The method successfully filtered out the

ARABPSYCHOLOGY.COM

unique rows and the first occurrences of the duplicate sets, leaving us with exactly two rows that are exact duplicates of other rows in the DataFrame.

Controlling Which Duplicates to Keep using the `keep` Parameter

The behavior of the duplicated() method regarding which instances are marked as duplicates can be managed using the optional keep parameter. This parameter dictates which occurrence of a set of duplicate values should be treated as the non-duplicate (`False`) and which should be marked as the duplicate (`True`).

To illustrate the effect of changing this parameter, we apply `keep='last'` to identify the first instances of the duplicate rows, marking them as the records to be identified for potential removal or inspection.

#identify duplicate rows
duplicateRows = df

#view duplicate rows
print(duplicateRows)

team points assists
0 A 10 5
6 B 20 6

Example 2: Finding Duplicates Based on a Subset of Columns

Often, you only want to define uniqueness based on a select set of columns, ignoring potential variations in other fields. This is useful when performing a specific business logic check, such as ensuring no player on the same team has the exact same point total. This functionality is handled by the `subset` parameter of the duplicated() method.

Here, we identify duplicates considering only the 'team' and 'points' columns, ignoring the 'assists' column entirely:

#identify duplicate rows across 'team' and 'points' columns
duplicateRows = df)]

#view duplicate rows
print(duplicateRows)

team points assists

ARABPSYCHOLOGY.COM

1 A 10 5
3 A 12 9
7 B 20 6

We now see three results, indicating three instances where the combination of 'team' and 'points' was a duplicate of a previously encountered row. This demonstrates how the `subset` parameter allows for highly targeted duplicate identification, independent of the rest of the dataset's columns.

Example 3: Identifying Duplicates Within a Single Column

A common data validation requirement is checking for duplicate entries within a single key field, such as ensuring uniqueness for user IDs or transaction numbers. By passing a list containing only one column name to the `subset` parameter, we instruct Pandas to treat every row whose value in that column matches a previous value as a duplicate.

#identify duplicate rows in 'team' column
duplicateRows = df)]

#view duplicate rows
print(duplicateRows)

team points assists
1 A 10 5
2 A 12 7
3 A 12 9
5 B 17 9
6 B 20 6
7 B 20 6

The resulting DataFrame shows six rows. Since the first occurrence of 'A' (index 0) and the first occurrence of 'B' (index 4) are marked as originals, all subsequent rows belonging to Team A and Team B are flagged as duplicates based on the 'team' column alone. This high number of results accurately reflects the structure of our sample data.

Summary of Key Parameters and Best Practices

Effectively managing redundant data relies on a precise understanding of the available tools within the Pandas ecosystem. The `.duplicated()` method, coupled with its powerful parameters, provides the necessary control for sophisticated data quality checks. Remembering the fundamental differences between the parameters is crucial for accurate data processing.

ARABPSYCHOLOGY.COM

Essential Parameters for `.duplicated()`

`subset`: This parameter defines the boundary of the uniqueness check. Use it when you need to ignore specific columns or focus the check exclusively on a handful of key fields. It takes a list of column labels.
`keep`: This parameter determines which instance of a duplicate set is marked as the original (`False`) and which are marked as duplicates (`True`). It is indispensable for defining the retention strategy, whether you prefer to keep the first, the last, or none of the duplicate entries.

Finally, remember that `.duplicated()` is purely for identification, returning a Boolean series mask. If the goal is data reduction, the next logical step is applying the drop_duplicates() method. This method accepts the exact same `subset` and keep parameter arguments, allowing you to directly remove the rows identified as duplicates based on your specified criteria. Whether you are performing quick verification or large-scale cleaning, mastering these Pandas functions is fundamental to robust data preparation.

ARABPSYCHOLOGY.COM