How to Easily Find and Sort Unique Values in a Pandas Column

Authored by

stats writer

November 20, 2025

ARABPSYCHOLOGY.COM

RECOMMENDED CITATION

stats writer (2025). *How to Easily Find and Sort Unique Values in a Pandas Column*. PSYCHOLOGICAL SCALES. Retrieved from https://scales.arabpsychology.com/?p=98487

Welcome to this detailed guide on leveraging the power of Pandas and Python to efficiently identify and organize data within a pandas DataFrame. Data analysis frequently requires isolating the distinct values within a feature column, followed by arranging those values in a meaningful order. This process is fundamental for data cleaning, aggregation, and insightful reporting.

While the original approach might suggest using the `unique()` function, a more robust and idiomatic solution for achieving sorted unique values that maintains the structure needed for subsequent Pandas operations involves chaining the `drop_duplicates()` and `sort_values()` functions. This methodology ensures that the output is a properly structured pandas Series, which is inherently designed to handle sorting operations.

The Necessity of Finding Unique Values in Data Science

In modern data analysis, particularly when dealing with large datasets, identifying the set of unique elements within a column is a critical first step. This task helps analysts understand the cardinality of a variable, diagnose data entry errors, and prepare for categorical encoding or grouping operations. For instance, if you have a column listing customer locations, finding the unique values immediately tells you how many distinct geographic areas your customers inhabit.

The need to sort these unique values arises when clarity and presentation are paramount. Whether you are generating a report for stakeholders or preparing data for visualization, presenting unique elements in a logical sequence--such as alphabetical order (ascending) or numerical magnitude (descending)--makes the data far easier to interpret. Pandas, built on the solid foundation of Python, provides straightforward methods to execute this chained operation seamlessly.

Choosing the correct functions for this task is key. While `.unique()` is fast and simple, it returns a NumPy array, which means you lose the convenient Series methods like `.sort_values()`. By contrast, combining `.drop_duplicates()` with `.sort_values()` preserves the pandas Series structure, allowing for effortless sorting and subsequent manipulation.

Understanding the Core Syntax for Sorting Unique Values

The standard methodology for extracting sorted unique values involves two chained functions applied directly to the target column (which is itself a pandas Series). The first step, `drop_duplicates()`, acts like an initial filter, removing all redundant entries and leaving only one instance of each value present in the column. The second step, `sort_values()`, takes the resultant unique set and orders it according to specified criteria.

The fundamental syntax is simple and highly readable, reflecting the "chaining" philosophy common in Pandas operations. This syntax is applied directly to the column of your DataFrame,

ARABPSYCHOLOGY.COM

identified using standard dictionary-style bracket notation.

You can use the following basic syntax to find the unique values in a column of a pandas DataFrame and then sort them:

df.drop_duplicates().sort_values()

This command returns a pandas Series containing every unique value found in the specified column, sorted automatically in **ascending order** (from smallest to largest, or A to Z). The indices of the original DataFrame are preserved in the Series output, though they are often irrelevant once the duplicates have been removed and the values sorted.

Controlling Sort Order: Ascending vs. Descending

To gain full control over the output order, the `sort_values()` function accepts an argument called `ascending`. By default, `ascending` is set to `True`. To reverse the order--sorting from largest to smallest, or Z to A--we simply pass the Boolean value `False` to this parameter.

To instead sort the unique values in **descending order**, use **`ascending=False`**:

df.drop_duplicates().sort_values(ascending=False)

Understanding these parameters is vital for tailoring data presentation to specific analytical needs. For numerical data, descending order is often useful when focusing on top performers or highest values, while ascending order provides a baseline perspective.

Practical Demonstration: Setting up the DataFrame

To illustrate this process clearly, we will work through an explicit example. We first need to define a sample pandas DataFrame that contains duplicate entries in one or more columns. This allows us to observe the effects of both the `drop_duplicates()` and `sort_values()` functions.

Suppose we are tracking team scores and have the following sample pandas DataFrame:

import pandas as pd

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

#view DataFrame

ARABPSYCHOLOGY.COM

print(df)

team points
0 A 5
1 A 5
2 A 9
3 A 12
4 A 12
5 B 5
6 B 10
7 B 13
8 B 13
9 B 19

In this DataFrame, we can clearly observe duplicate entries in the `points` column (5 appears three times, 12 appears twice, and 13 appears twice). Our goal is to extract the unique scores (5, 9, 10, 12, 13, 19) and present them in a clean, sorted list.

Executing the Sort in Ascending Order

We will first perform the sorting operation using the default settings, which organizes the unique values from the smallest numerical value to the largest. This is the most common use case for reviewing the range and distribution of data points.

We apply the chained functions to the `points` column. The `drop_duplicates()` function first filters the 10 entries down to 6 unique points, and then `sort_values()` ensures they are arranged sequentially.

#get unique values in points column and sort them (ascending default)
df.drop_duplicates().sort_values()

0 5
2 9
6 10
3 12
7 13
9 19
Name: points, dtype: int64

ARABPSYCHOLOGY.COM

The resulting pandas Series output displays each of the unique scores from the `points` column sorted correctly in ascending numerical order. Note that the indices on the left (0, 2, 6, 3, 7, 9) correspond to the row index of the first occurrence of that value in the original DataFrame before sorting, but they should be ignored for interpretation of the result set itself.

For easier interpretation and integration into reports, we can list these unique values:

5
9
10
12
13
19

Executing the Sort in Descending Order

When analyzing performance metrics or scores, it is often more intuitive to view the results starting with the highest value. By explicitly setting the `ascending` parameter to `False` within the `sort_values()` function, we immediately reverse the output order.

This is extremely useful in business intelligence and exploratory data analysis (EDA) where quickly identifying the maximum boundary of a variable is necessary. It ensures that the most impactful or highest data point is immediately visible.

We can also get the unique values in the `points` column sorted in descending order by specifying **`ascending=False`** within the `sort_values()` function:

#get unique values in points column and sort them in descending order
df.drop_duplicates().sort_values(ascending=False)

9 19
7 13
3 12
6 10
2 9
0 5
Name: points, dtype: int64

The output now displays the distinct scores sorted from the highest (19) down to the lowest (5), providing an immediate top-down view of the available scores in the dataset.

ARABPSYCHOLOGY.COM

The output displays each of the unique values in the `points` column sorted in descending order:

```
19
13
12
10
9
5
```

Advanced Consideration: The Role of `unique()` vs. Chained Methods

While this tutorial emphasizes the combined use of `drop_duplicates()` and `sort_values()` for structured sorting, it is important to acknowledge the existence of the `unique()` method and understand why it is generally less suitable for direct, sorted output within the Pandas environment.

The `.unique()` method is exceptionally fast and optimized for extracting unique values. However, its output is a NumPy array, not a pandas Series. NumPy arrays do not possess the native `.sort_values()` method, which means that to sort the output of `.unique()`, you would need to rely on NumPy's sorting functions or convert the array back into a Series or list, adding extra steps:

Extract unique values: `df.unique()` (returns NumPy array).

Convert to list and sort: `sorted(df.unique())` (returns a sorted Python list).

Conversely, the sequence `df.drop_duplicates().sort_values()` performs the entire operation end-to-end using only Pandas methods, keeping the output as a pandas Series, which is often preferred for further data processing or calculations involving alignment with other data structures.

Handling Missing Data (NaN) During Unique Sorting

A crucial consideration in real-world data is the presence of missing values, often represented as `NaN` (Not a Number). Both `drop_duplicates()` and `sort_values()` handle these values predictably, but the behavior should be understood.

By default, `drop_duplicates()` treats `NaN` as a distinct value and includes only one instance of it in the unique set. When `sort_values()` is applied, `NaN` values are typically placed at the end of the sorted output, regardless of whether the sorting is ascending or descending. This is controlled by the `na_position` parameter in `sort_values()`, which defaults to `'last'`.

If you wanted to specifically exclude missing values from your unique list, you would need to chain a filter operation using `.dropna()` before applying the `drop_duplicates()` function. Ensuring data cleanliness regarding `NaN` is a necessary step before trusting the integrity of the unique count.

ARABPSYCHOLOGY.COM

Summary of Best Practices for Unique Sorting

When working with large DataFrames in Python, adopting the chained function approach is the recommended best practice for obtaining sorted unique values. This method prioritizes code clarity, functional completeness, and adherence to the Pandas object structure.

The sequence `.drop_duplicates().sort_values(ascending=...)` is powerful because it keeps the entire operation within the optimized Pandas ecosystem, avoiding unnecessary data type conversions that can occur when mixing Pandas with NumPy or standard Python list operations.

Remember that for categorical columns, the sorting will be lexicographical (alphabetical), while for numerical data, it will be based on magnitude. Always verify the data type of the column using `df.dtype` if unexpected sorting results occur.

The following tutorials explain how to perform other common functions in pandas:

ARABPSYCHOLOGY.COM