How to Easily Describe Categorical Variables with Python

Authored by

stats writer

November 20, 2025

The describe() function in pandas is an indispensable tool within the Python data analysis ecosystem, providing swift and informative summaries of data distributions. While it is commonly utilized to calculate measures of central tendency, dispersion, and shape for numerical columns, its capability extends powerfully to handle categorical variables as well. Understanding how to properly invoke `describe()` for non-numeric data is crucial for any data analyst, as it allows for immediate insight into the composition and quality of nominal or ordinal features within a dataset. When applied to categorical data, the function delivers essential metrics like the count of entries, the number of unique categories, the most frequent category (the mode), and its corresponding frequency, making it an invaluable starting point for exploratory data analysis.

The initial analysis of categorical variables often dictates subsequent steps in data cleaning and feature engineering. By using the specialized parameters available in the describe() method, analysts can bypass the default behavior of focusing exclusively on numerical summaries (like mean, standard deviation, and quartiles). The resultant summary statistics--specifically the metrics like uniqueness and frequency--help identify potential issues such as low cardinality, high cardinality, or class imbalances, which are fundamental concerns when preparing data for machine learning models or detailed reporting. Mastering these methods ensures that all components of a DataFrame, regardless of their underlying data type, receive a robust statistical overview.

Understanding the Default Behavior of describe() in Pandas

By default, when invoked on a DataFrame without any specific arguments, the **describe()** function is designed to calculate descriptive statistics exclusively for columns that possess a numeric data type, such as integers or floats. This is a logical starting point for many datasets where numerical features dominate, providing the standard five-number summary (minimum, maximum, quartiles), along with the count, mean, and standard deviation. The function intentionally excludes columns typed as 'object' (which often represent strings or mixed types, including categorical data) or dedicated categorical types unless explicitly told otherwise. This default exclusion is necessary because mathematical summaries like the mean or standard deviation are nonsensical when applied to nominal categories like 'color' or 'country,' highlighting the need for specialized parameters when dealing with non-numeric fields.

The automatic filtering mechanism inherent in **describe()** is efficient but sometimes masks critical information contained within the categorical features. When a dataset contains a mix of numeric and string columns, running the default `df.describe()` command will silently ignore all textual or categorical attributes, focusing solely on quantitative analysis. To gain a complete picture of the dataset's structure, the analyst must actively intervene using the `include` parameter. This parameter allows the user to specify which data types should be considered for the summary calculation, thus overriding the standard numeric-only restriction and allowing us to focus specifically on the discrete nature of categorical data, characterized by counts and frequencies

rather than continuous measures.

As mentioned, the standard behavior in pandas calculates descriptive statistics for all numeric variables in a DataFrame. However, to access the corresponding summary statistics for categorical variables, two primary methods are employed, both leveraging the flexibility of the **describe()** function. These methods are essential for performing a comprehensive preliminary analysis, ensuring no feature type is overlooked during the critical exploratory phase of a project. We will explore each technique in detail, starting with the most precise method tailored specifically for object-type columns.

Method 1: Utilizing the include Parameter for Object Data Types

The most direct and recommended approach for summarizing categorical variables is by explicitly passing the `include='object'` argument to the **describe()** function. In pandas, string columns and mixed data type columns are typically stored using the NumPy 'object' data type. By setting `include='object'`, we instruct pandas to bypass the numeric columns and instead focus its descriptive calculation solely on these non-numeric columns. This approach yields a focused summary tailored for discrete data, providing metrics that are meaningful for nominal or ordinal scales, such as counts and mode frequency, instead of inapplicable statistical averages.

This method drastically simplifies the process of data auditing when the focus is purely on the qualitative aspects of the dataset. For columns that are identified as 'object' types, the function outputs four key statistical metrics that collectively describe the category structure: **count**, **unique**, **top**, and **freq**. These metrics are fundamental for assessing data integrity, identifying potential issues like spelling variations (indicated by a high unique count relative to the total count), and establishing the most common category, which is crucial for handling missing values or performing baseline comparisons. The clarity and specificity of this method make it the preferred technique when you need to quickly isolate and summarize only the string or categorical components of a large DataFrame.

The corresponding code syntax for this method is remarkably concise, demonstrating the efficiency inherent in the pandas library:

df.describe(include='object')

When this code is executed, the output table will only include the columns identified as categorical or object types, presenting the four specialized statistics: **count**, representing the number of non-null entries; **unique**, indicating the number of distinct categories; **top**, which is the modal category; and **freq**, the count of observations belonging to that modal category. This targeted analysis is highly beneficial for datasets with numerous columns where separating numeric from categorical

summaries enhances readability and interpretation.

Method 2: Coercing All Variables to Object Type for Universal Categorical Summary

An alternative, though more aggressive, method involves temporarily casting the DataFrame or specific columns to the 'object' data type before calling **describe()**. By chaining the `astype('object')` method before invoking describe(), we force pandas to treat every single column--including those that were originally numeric--as if they contained categorical data. This provides a unified categorical summary across the entire DataFrame, which can be particularly useful when comparing the cardinality (uniqueness) of both seemingly numeric and truly categorical features side-by-side.

While this technique provides the same four categorical statistics (**count**, **unique**, **top**, and **freq**) for every column, it must be used with caution. The primary drawback is that it fundamentally changes the context for numeric variables. For example, if a column contains continuous numerical scores, treating them as categorical might obscure the statistical distribution. However, seeing the number of unique values for a numeric column can be insightful--if a numeric column that should contain continuous data shows very low uniqueness, it might indicate that it is actually an encoded categorical feature (e.g., ratings on a scale of 1 to 5) masquerading as a float or integer. Thus, this method serves as an excellent tool for data quality checks and identifying misclassified feature types.

The syntax for implementing this universal categorical summary is achieved through method chaining in Python, where the temporary type conversion is immediately followed by the describe() function call:

df.astype('object').describe()

Executing this command will yield a summary table encompassing every variable in the DataFrame. Crucially, the statistics generated are always those appropriate for categorical data, even for the numeric columns, meaning you will see the **top** value (the mode) and its **freq**, rather than the mean or standard deviation. This provides a uniform view across the entire dataset regarding category representation and frequency distribution.

Setting Up the Illustrative Example Dataset

To demonstrate the practical application of both descriptive methods for categorical variables, we will utilize a small, simulated DataFrame created using the pandas library in Python. This dataset contains information related to basketball player statistics, featuring a mix of categorical and

numerical variables. Specifically, we have 'team' as the categorical identifier (string/object type), and 'points', 'assists', and 'rebounds' as numeric integer types, allowing us to clearly differentiate the output when applying the different **describe()** techniques.

The construction of this sample dataset involves importing the pandas library, creating a dictionary of lists to define the columns and data, and then instantiating the DataFrame. Having a clean, representative dataset is paramount for illustrating how the `include='object'` parameter successfully isolates the categorical column ('team') and how the `astype('object')` coercion affects the summary of the numerical columns. The code below initializes the eight-row dataset, which will be the basis for our subsequent analysis:

import pandas as pd

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

#view DataFrame
print(df)

team points assists rebounds
0 A 18 5 11
1 B 22 7 8
2 C 19 7 10
3 D 14 9 6
4 E 14 12 6
5 F 11 9 5
6 G 20 9 9
7 H 28 4 12

Upon reviewing the output, we confirm that 'team' is indeed an object (string) column, while 'points', 'assists', and 'rebounds' are numeric (integer) columns. This structured dataset allows us to proceed confidently to the first example, where we apply the targeted describe() function solely to the categorical component of this data, demonstrating Method 1 in practice and showcasing its ability to isolate specific data types for analysis.

Example 1: Isolating Categorical Variables using include='object'

Our first example focuses on using the `include='object'` parameter to calculate descriptive

statistics exclusively for the categorical columns within our basketball player dataset. Since 'team' is the only non-numeric column, the output will provide a highly focused summary of its structure, cardinality, and frequency distribution. This technique is especially useful when dealing with wide datasets containing dozens of columns, allowing analysts to quickly verify the quality and distribution of features intended for use in grouping or classification tasks without being distracted by the numerical metrics.

We execute the following concise Python command to generate the summary for 'team':

#calculate descriptive statistics for categorical variables only
df.describe(include='object')

team
count 8
unique 8
top A
freq 1

The resulting output clearly demonstrates the power of targeted analysis. Only the 'team' column is returned, along with the four specialized statistics for categorical variables. Interpreting this summary is straightforward and provides immediate insights into the structure of this identifier column. We can see immediately that there are 8 non-null entries, and critically, all 8 entries are unique, indicating that 'team' acts as a primary identifier for the observations--a key piece of information for subsequent data modeling steps.

Interpreting the Categorical Output Metrics

The specialized summary generated by applying describe() to categorical data provides four distinct metrics, each offering valuable information about the distribution of categories. Understanding these metrics is essential for accurate exploratory data analysis (EDA). These statistics diverge significantly from the numerical output (like mean and standard deviation) because they focus on frequency and identification rather than continuous distribution.

The interpretation of the output for the 'team' column is as follows:

count: This metric reveals the total number of non-null observations in the column. For 'team', the count is 8, confirming that all rows in our DataFrame have a value for this variable. If the count were less than the total number of rows (8 in this case), it would signal the presence of missing data, which requires attention.
unique: This metric states the number of distinct or unique categories present in the column. For 'team', this value is 8. Since the count also equals 8, we confirm that every observation has a

unique team designation, indicating that this variable functions as a unique identifier for each player entry in the dataset. A low unique count relative to the total count would suggest a variable with high repetition and low cardinality.

top: This value represents the mode of the distribution--the category that appears most frequently in the column. In this specific output, 'A' is listed as the top value. It is important to note that when all categories have the same frequency (as is the case here, where frequency is 1), describe() simply selects the first one it encounters, often based on alphabetical or internal ordering.

freq: This final metric indicates how many times the value identified as 'top' actually appears in the dataset. Since 'A' is the top category and it only appears once, the frequency is 1. If the frequency were significantly higher than 1, it would highlight a dominant category or a severe class imbalance, prompting further investigation into that particular category's influence on the overall data structure.

Example 2: Applying Categorical Summary to All Variables via Type Coercion

The second example demonstrates how to calculate the categorical descriptive statistics (count, unique, top, freq) for every single column in the DataFrame, including the originally numeric ones ('points', 'assists', 'rebounds'). This is achieved by temporarily converting the entire DataFrame to the 'object' data type using `astype('object')` before executing **describe()**. This method is instrumental in revealing the cardinality profile of variables that might otherwise be masked by their numeric designation.

Executing the conversion and description yields the following comprehensive summary:

#calculate categorical descriptive statistics for all variables
df.astype('object').describe()

team points assists rebounds
count 8 8 8 8
unique 8 7 5 7
top A 14 9 6
freq 1 2 3 2

This combined output shows **count**, **unique**, **top**, and **freq** for all four variables. While the 'team' column statistics remain the same, the output for the numerical columns (points, assists, rebounds) now provides categorical summaries. For instance, the 'points' column, which is numeric, shows that there are 7 unique values, with '14' being the most frequent score (the mode), appearing 2 times. This confirms that even though a variable holds numerical meaning, the coercion allows us to analyze its distribution in terms of discrete occurrences and category concentration.

Comparing Targeted vs. Universal Categorical Description

Choosing between Method 1 (`include='object'`) and Method 2 (`astype('object').describe()`) depends entirely on the analytical goal. Method 1 is precise and clean; it isolates the true categorical variables (those stored as strings or objects) and provides an accurate measure of their composition. This is the ideal method for standard exploratory data analysis where you wish to separate qualitative analysis from quantitative analysis. It avoids misinterpretation by focusing only on variables where categorical metrics are inherently appropriate.

Conversely, Method 2 offers a holistic view of uniqueness and frequency across the entire DataFrame, regardless of the underlying data type. While statistically cautious analysts might avoid coercing numeric types, this method is highly valuable for data quality assessment. If a numeric column exhibits very low uniqueness (e.g., unique=3 out of count=1000), Method 2 immediately flags it as a potential categorical feature that should be converted to an official pandas categorical data type to save memory and improve performance in modeling. Therefore, Method 2 serves as a rapid, high-level inspection tool for assessing variable cardinality across the board.

Summary and Best Practices for Categorical Summarization

The ability to effectively use the describe() function for categorical variables is a cornerstone of robust data analysis in Python. By understanding and applying the `include='object'` parameter (Method 1), analysts can generate relevant statistics--count, unique, top, and freq--that accurately reflect the discrete nature of nominal and ordinal data, ensuring that the initial exploratory phase is comprehensive and targeted.

As a best practice, always prioritize Method 1 when the goal is to summarize variables explicitly defined as object or string types. Use Method 2 sparingly, primarily as a diagnostic tool for cross-checking the cardinality of numeric columns that may secretly be discrete or encoded categorical features. Both methods underscore the flexibility of pandas and ensure that crucial descriptive information regarding the distribution and composition of all variable types is readily available, paving the way for more detailed modeling and statistical inference. Mastering these techniques ensures that no part of the dataset remains opaque during the crucial initial investigation.