How to Easily Create Categorical Variables in Pandas Using cut and qcut

Authored by

stats writer

November 24, 2025

RECOMMENDED CITATION

stats writer (2025). *How to Easily Create Categorical Variables in Pandas Using cut and qcut*. PSYCHOLOGICAL SCALES. Retrieved from https://scales.arabpsychology.com/?p=100359

ARABPSYCHOLOGY.COM

Creating categorical variables in Pandas is a fundamental skill in data preparation and feature engineering. This process involves transforming raw data--often continuous or textual--into a structured set of distinct categories or levels. Effective categorization is essential because many statistical models and machine learning algorithms perform better when input features are represented as discrete categories rather than continuous numerical scales.

When working with the Pandas library in Python, we primarily use built-in functions like `cut` and `qcut` to manage this transformation efficiently. These functions are designed to segment or 'bin' a column of numerical data into specified intervals, subsequently assigning a human-readable label or category to each interval. This transformation significantly aids in subsequent data analysis and visualization efforts, allowing analysts to quickly grasp distributions and relationships within the data.

Understanding the nuances of these methods is crucial for any data scientist. Whether you are defining categories based on predefined business logic or attempting to optimize model performance, the ability to generate clean and valid categorical features using Pandas ensures that your DataFrame is structured for maximum analytical utility. In this comprehensive guide, we will explore both manual creation and automated methods using practical examples.

ARABPSYCHOLOGY.COM

Overview of Categorization Techniques in Pandas

Within the Pandas ecosystem, there are two primary approaches to creating categorical variables. The first method involves explicitly defining the categories, often utilized when the data is already discrete or textual. The second, more common method involves converting a continuous numerical variable into distinct classes or groups using binning techniques. Both approaches are essential depending on the source data's nature and the requirements of the subsequent analysis.

The choice between these methods depends heavily on whether your source data is naturally categorical (e.g., product names, colors) or continuous (e.g., scores, ages). For naturally categorical data, simple assignment is sufficient. However, for continuous data, we must define the boundaries of the categories, a process known as binning. Pandas provides highly efficient, vectorized operations to handle both scenarios, ensuring fast execution even with large datasets.

Below we outline the foundational syntax for both manual creation and the conversion of existing numerical columns using the powerful `pd.cut()` function, setting the stage for detailed examples that follow.

You can use one of the following methods to create a categorical variable in Pandas:

Method 1: Creating a Categorical Variable from Scratch

This method is straightforward and involves initializing a new column directly within the DataFrame with a list or series of predefined category values. This is typically done when creating flags, grouping indices, or when manually inputting known qualitative data into the dataset.

When you assign a list of strings directly to a new column name, Pandas automatically infers the data type, usually setting it to `object`, which is Python's general type for strings and mixed types, and serves as the foundation for categorical variables before explicit type conversion. While simple string assignment works, for memory optimization and improved performance in statistical operations, it is highly recommended to explicitly convert this column to the official `category` dtype using `.astype('category')` after initialization.

Here is the fundamental syntax for creating a new categorical column by assigning predefined textual labels:

df =

Method 2: Converting Numerical Data to a Categorical Variable via Binning

Perhaps the most common need in data preparation is converting a continuous numerical variable into discrete categories. This process, known as discretization or binning, is achieved in Pandas primarily through the `pd.cut()` function. The `cut()` function divides the range of values in the input array into specified intervals, or bins, and assigns a label to each resulting bin. This is invaluable when creating classes like 'Low,' 'Medium,' or 'High' based on scores or measurements.

The `pd.cut()` function requires two key parameters: the series to be binned and the definition of the bin edges (`bins`). Optionally, but highly recommended for clarity, are the `labels`, which provide meaningful names to the created categories. Defining appropriate bin boundaries is a critical step that often relies on domain knowledge or statistical analysis of the distribution of the numerical variable.

The following example demonstrates how to apply the `cut` function to segment a numerical column into three distinct categories: 'Bad', 'OK', and 'Good', using specified numerical breakpoints.

df = pd.cut(df,
bins=,
labels=)

The examples below showcase how to use each method in practice, ensuring a clear

ARABPSYCHOLOGY.COM

understanding of the resulting data structure and types.

Example 1: Creating a Categorical Variable from Scratch in a DataFrame

To illustrate the first method, we will construct a basic DataFrame containing team names and their corresponding scores. The 'team' column, which consists of strings, is inherently categorical. By analyzing the data types after creation, we confirm how Pandas initially interprets such textual data.

This approach is useful for building mock datasets, adding identifier columns, or incorporating external categorical information that isn't derived from existing numerical data. The subsequent code initializes the data structure and then inspects the data types using the `.dtypes` attribute, a critical step in verifying data integrity and type interpretation by the Python environment.

The following code shows how to create a Pandas DataFrame with one inherently categorical variable called `team` and one numerical variable called `points`:

import pandas as pd

#create DataFrame with one categorical variable and one numeric variable
df = pd.DataFrame({'team': ,
'points': })

#view DataFrame
print(df)

team points
0 A 12
1 B 15
2 C 19
3 D 22
4 E 24
5 F 25
6 G 26
7 H 30

#view data type of each column in DataFrame
print(df.dtypes)

team object
points int64
dtype: object

ARABPSYCHOLOGY.COM

By using **df.dtypes**, we can see the resulting data types in the DataFrame. This output is critical for confirming that the data structure is as expected before proceeding with advanced statistical analysis or model training.

Specifically, observing the output reveals the following data types:

The variable `team` is designated as an **object**.
The variable `points` is designated as an **int64**.

In the context of Pandas, an **object** data type is the default storage mechanism for string data. Although it effectively holds categorical information, it is generally treated as equivalent to a character or nominal categorical variable when dealing with non-numeric data, such as the team variable. For optimized operations, converting this **object** type explicitly to `category` is the preferred best practice.

Example 2: Discretizing Numerical Data Using pd.cut()

The second example demonstrates the practical application of discretizing a continuous column--the `points` score--into a new set of discrete, labeled categories representing performance status. This is a crucial step in transforming raw measurements into meaningful features suitable for modeling, where numerical magnitude might be less important than the category it belongs to (e.g., separating "failing" scores from "passing" scores).

We use the `cut` function to define custom bin edges, allowing us to control the exact size and boundaries of each category. By defining the bins explicitly, we ensure that the categorization aligns precisely with predefined business rules or analytical requirements. The use of `float('Inf')` allows the final bin to capture all values greater than the last defined boundary, ensuring no data points are missed.

The following code shows how to create a new categorical variable called `status` from the existing numerical variable called `points` within the DataFrame:

import pandas as pd

#create DataFrame with one categorical variable and one numeric variable
df = pd.DataFrame({'team': ,
'points': })

#create categorical variable 'status' based on existing numerical 'points' variable
df = pd.cut(df,
bins=,
labels=)

ARABPSYCHOLOGY.COM

#view updated DataFrame
print(df)

team points status
0 A 12 Bad
1 B 15 Bad
2 C 19 OK
3 D 22 OK
4 E 24 OK
5 F 25 OK
6 G 26 Good
7 H 30 Good

Understanding Binning Parameters in pd.cut()

The successful application of the `cut` function hinges on correctly defining the boundary conditions using the `bins` parameter and assigning appropriate `labels`. Using the `cut()` function, we successfully created the new categorical variable called `status` that assigns categories based on numerical thresholds defined by the boundaries 0, 15, 25, and infinity.

Reviewing the resulting DataFrame, we can observe the logical mapping defined by the binning:

The category **'Bad'** is assigned if the value in the `points` column is greater than 0 and less than or equal to 15.
The category **'OK'** is assigned if the value in the `points` column is greater than 15 and less than or equal to 25.
The category **'Good'** is assigned otherwise (i.e., if the value is greater than 25, extending up to positive infinity).

A critical rule to remember when utilizing the `cut` function is the relationship between the number of bins and the number of labels. The number of **labels** provided must always be exactly one less than the number of **bins** defined. This is because N bins define N-1 intervals, and each interval requires a unique label.

In our example, we used four values for **bins** to define three distinct bin edges (0 to 15, 15 to 25, 25 to Inf), and correspondingly, we used three values for **labels** ('Bad', 'OK', 'Good') to specify the names for the resulting categories. Failing to maintain this N vs. N-1 relationship will result in a ValueError from Pandas.

ARABPSYCHOLOGY.COM

Advanced Binning: Using pd.qcut() for Quantile-Based Segmentation

While `pd.cut()` is ideal for defining bins based on fixed, predetermined numerical intervals (e.g., grading scales), the `pd.qcut()` function offers an alternative, distribution-aware approach. `qcut()` ensures that each resulting bin contains an approximately equal number of observations based on the quantiles of the data. This technique is particularly useful in data analysis when the underlying data distribution is skewed, or when the goal is to create equally sized groups (e.g., tertiles, quartiles, or quintiles).

If we were to use `qcut()` on the `points` column and specify `q=4`, the function would attempt to divide the data into four quartiles, ensuring that roughly 25% of the data falls into the 'Q1,' 'Q2,' 'Q3,' and 'Q4' bins, regardless of the numerical span of those bins. This adaptive approach to binning is often superior for creating ranks or performance tiers where relative position within the population matters more than absolute scores.

Although the input sample is small, demonstrating the syntax of `qcut()` helps illustrate this powerful function. Instead of specifying the exact bin edges, we specify the number of quantiles (q) or the quantile probabilities (e.g.,).

 # Example using qcut to create 4 equal-sized groups (Quartiles)
df = pd.qcut(df, q=4,
labels=)
print(df)

Best Practices and Data Type Conversion

Regardless of whether you use manual assignment, `cut`, or `pd.qcut` to create your categorical variables, the final and most important step in data preparation is often ensuring the column is stored using the explicit Pandas `category` data type. While string-based object types function correctly for basic operations, using the `category` dtype offers substantial memory savings, especially for columns with high redundancy (many repeated values), and allows for specialized statistical functions designed specifically for categorical data.

To convert any column (including those created via assignment or binning) to the optimized type, use the `.astype()` method. For instance, `df = df.astype('category')` explicitly tells Pandas to use its optimized categorical representation. This conversion is crucial before passing data to libraries like Scikit-learn, which often require explicit handling of categorical features.

Furthermore, when performing any data analysis involving categorical variables, always check the category order. If the categories represent an ordinal scale (like 'Bad', 'OK', 'Good'), ensure that the

ARABPSYCHOLOGY.COM

order is correctly defined in the `labels` array or adjusted afterward using `.cat.reorder_categories()`. This maintains the logical structure of the data for visualizations and ordered statistical comparisons.

Summary of Categorization Benefits in Data Analysis

The transformation of raw numerical data into binned categorical variables is a powerful technique in feature engineering. It helps smooth out noise in highly variable continuous data, simplifies models by reducing complexity, and facilitates non-parametric statistical testing. By grouping continuous values into meaningful bins, we often uncover patterns and relationships that might be obscured when viewing the raw data alone.

Mastering the use of `pd.cut()` and `pd.qcut` provides analysts with the flexibility to define categories based on either fixed interval logic or distribution-based equal representation. Both functions are cornerstones of efficient data manipulation within the Pandas environment, allowing for cleaner preprocessing steps and more effective modeling outcomes.

Remember that careful selection of bin boundaries (for `cut`) or the number of quantiles (for `qcut`) directly impacts the resulting analysis. Always review the data distribution and consult domain expertise to define the most analytically sound binning strategy. This diligence ensures the categorical variables accurately reflect the underlying phenomenon being studied.

ARABPSYCHOLOGY.COM