Learning to Analyze Categorical Data: A Step-by-Step Guide to Creating Contingency Tables in Python

Authored by

Mohammed looti

November 5, 2025

In the expansive field of data analysis and statistical research, establishing clear relationships between qualitative variables is fundamentally important. When dealing with discrete, descriptive data, the tool of choice for summarizing frequency distributions is the contingency table. Often referred to interchangeably as a cross-tabulation or a crosstab, this structured visualization is indispensable for helping analysts rapidly identify underlying patterns, dependencies, and potential associations within complex datasets.

This comprehensive guide is designed to demonstrate the efficient generation and precise interpretation of contingency tables utilizing the formidable data manipulation capabilities of the pandas library within the Python programming environment. Our primary focus will be on mastering the `pandas.crosstab` function, detailing its syntax, core parameters, and practical application through a step-by-step example. By the end of this tutorial, you will possess the expertise to transform raw categorical data into meaningful statistical summaries.

The Statistical Foundation: Contingency Tables and Categorical Data

A contingency table systematically organizes raw data by displaying the count, or frequency distribution, of observations that fall into every possible combination of categories defined by two or more chosen variables. Consider a scenario involving market research: an analyst might cross-tabulate 'Customer Satisfaction Level' against 'Product Version' to visualize exactly how many users of each product version fall into 'Highly Satisfied', 'Neutral', or 'Dissatisfied' groups. This joint frequency display is the core utility of the crosstab.

Crucially, the variables employed in a cross-tabulation must be categorical variables. These are variables whose values are restricted to a fixed, finite number of labels or groups. These can be nominal categories, such as 'Male' or 'Female', or ordinal categories, which have an inherent ordering, such as 'Low', 'Medium', or 'High'. The structure of the resulting table--specifically the labels defining the rows and columns--is determined entirely by the unique levels present within these selected categories.

The successful generation of this basic table serves as the vital first step for conducting sophisticated statistical inference. For instance, it is the prerequisite output for performing the Chi-Squared test for independence, a rigorous statistical procedure used to formally determine whether the observed relationship between the two variables is statistically significant or merely due to random chance. Understanding the raw frequencies provided by the contingency table is key to understanding the subsequent statistical conclusions.

Leveraging the Efficiency of pandas.crosstab() in Python

While the pandas library offers numerous powerful functions for data aggregation and summarization, `pandas.crosstab` stands out as the specialized and most streamlined function

specifically designed for computing cross-tabulations. It abstracts away the complexity of manually chaining together grouping and counting operations, offering a clean, one-line solution for frequency analysis.

The effective use of this function primarily hinges on defining two essential parameters that dictate the structural layout of the resultant table:

pandas.crosstab(index, columns)

These arguments structure the table output:

index: This parameter specifies the array, Series, or column name whose unique values will be assigned to define the **rows** of the resulting contingency table.
columns: This parameter specifies the array, Series, or column name whose unique values will be assigned to define the **columns** of the resulting contingency table.

The output is a new DataFrame where the data points within the cells represent the observed joint frequency counts--the number of times an intersection of a specific row category and a specific column category occurs in the dataset. This immediate count provides the foundational quantitative summary required for further exploratory data analysis (EDA).

Step 1: Preparing and Inspecting the Sample Dataset

Before any meaningful analysis can commence, a suitable dataset must be prepared. For this demonstration, we will construct a mock pandas DataFrame representing sales information for 20 unique orders. This dataset is engineered to contain two key categorical variables: `Product` (which can be 'TV', 'Computer', or 'Radio') and `Country` (which can be 'A', 'B', or 'C'), allowing us to examine the relationship between geographic location and product preference.

We initiate the process by importing the necessary pandas library, followed by the explicit construction of our sample DataFrame:

import pandas as pd

#create data
df = pd.DataFrame({'Order': ,
'Product': ,
'Country': })

#view data
df

Order Product Country
0 1 TV A
1 2 TV A
2 3 Comp A
3 4 TV A
4 5 TV B
5 6 Comp B
6 7 Comp B
7 8 Comp B
8 9 TV B
9 10 Radio B
10 11 TV B
11 12 Radio B
12 13 Radio C
13 14 Radio C
14 15 Comp C
15 16 Comp C
16 17 TV C
17 18 TV C
18 19 Radio C
19 20 TV C

The resulting DataFrame, named `df`, is successfully created. A quick inspection confirms that we are working with 20 total observations, and the critical categorical columns--`Product` and `Country`--each contain three unique levels, which will ultimately define the 3x3 structure of our contingency table.

Step 2: Generating and Displaying the Basic Contingency Table

With the data prepared, our next objective is to execute the cross-tabulation. To effectively analyze the joint distribution of products sold per country, we invoke the powerful `pandas.crosstab` function. We strategically define the 'Country' variable as the `index` parameter, ensuring that the countries populate the rows, and the 'Product' variable as the `columns` parameter, organizing the product types across the top.

Executing the function generates the initial frequency distribution, where the cell values represent the precise count for every unique intersection of the row and column categories:

#create contingency table
pd.crosstab(index=df, columns=df)

Product Comp Radio TV
Country
A 1 0 3
B 3 2 3
C 2 3 3

Interpreting the Joint Frequency Distribution

The resulting cross-tabulation table provides an immediate, quantitative display of the joint frequency distribution of the two variables. Each value housed within a cell signifies the number of orders where that specific country (row) purchased that specific product (column). This visualization offers crucial, actionable insights into the dataset's composition and highlights preliminary patterns of association between the variables.

We can systematically interpret these counts by analyzing the data in two primary dimensions:

Row-wise Interpretation (Conditional on Country): Reading across the rows reveals the product preference distribution specific to each country:

For **Country A**, the orders consisted of **1** Computer, **0** Radios, and **3** TVs.
For **Country B**, the orders were distributed as **3** Computers, **2** Radios, and **3** TVs.
For **Country C**, the orders included **2** Computers, **3** Radios, and **3** TVs.

Column-wise Interpretation (Total Product Popularity): Reading down the columns reveals the overall popularity of each product across all geographic regions combined:

The total count for **Computers (Comp)** is 1 (from A) + 3 (from B) + 2 (from C), totaling **6** computers purchased globally.
The total count for **Radios** is 0 (from A) + 2 (from B) + 3 (from C), totaling **5** radios purchased globally.
The total count for **TVs** is 3 (from A) + 3 (from B) + 3 (from C), totaling **9** TVs purchased globally.

This basic cross-tabulation not only confirms the exact number of product sales conditional on the geographic region but also immediately highlights notable patterns, such as the complete absence of radio sales in Country A during this specific observation period.

Step 3: Enhancing the Table with Marginal Totals (Margins)

While the joint frequency table is invaluable, analysts frequently require marginal totals--the sums of the rows and columns--to gain a complete understanding of the univariate distribution of each variable independent of the other. Marginal totals provide context for the cell counts. We can easily

incorporate these summations into the cross-tabulation by setting the optional argument `margins=True` within the `pandas.crosstab` function call.

#add margins to contingency table
pd.crosstab(index=df, columns=df, margins=True)

Product Comp Radio TV All
Country
A 1 0 3 4
B 3 2 3 8
C 2 3 3 8
All 6 5 9 20

The enhanced table now features an 'All' row and an 'All' column, which present the aggregated totals for both dimensions:

Interpretation of Marginal Totals:

Row Totals (The 'All' Column): This column represents the total order volume originating from each respective country, summarizing the univariate distribution of the `Country` variable:

Country A placed a total of **4** orders (4/20 = 20% of total orders).
Country B placed a total of **8** orders (8/20 = 40% of total orders).
Country C placed a total of **8** orders (8/20 = 40% of total orders).

Column Totals (The 'All' Row): This row represents the total number of orders placed for each respective product type across all countries, summarizing the univariate distribution of the `Product` variable:

A total of **6** computers were purchased globally.
A total of **5** radios were purchased globally.
A total of **9** TVs were purchased globally.

Crucially, the value situated in the bottom right corner of the table, at the intersection of the 'All' row and 'All' column (labeled **20**), represents the grand total number of observations in the entire dataset. This grand total serves as a fundamental verification point, ensuring that all data points have been correctly accounted for in the cross-tabulation.

Further Applications: Calculating Proportions and Percentages

While raw frequency counts are the default output, the true analytical power of `pandas.crosstab`

often lies in its ability to quickly display results as proportions or percentages, rather than absolute numbers. This transformation is controlled by the flexible `normalize` parameter, which allows analysts to shift focus from counts to relative distributions:

Setting `normalize='index'` transforms the cell values into **row percentages**. This answers questions like, "Conditional on being Country A, what percentage of those orders were TVs?"
Setting `normalize='columns'` transforms the cell values into **column percentages**. This answers questions like, "Conditional on a product being a TV, what percentage of those TV sales originated from Country C?"
Setting `normalize='all'` transforms the cell values into proportions of the **grand total**. This answers questions like, "What percentage of all 20 orders were specifically TVs purchased in Country B?"

Mastering the `crosstab` function in Python is a cornerstone skill for performing robust Exploratory Data Analysis (EDA). It provides a clean, highly effective method for summarizing and visualizing the intrinsic relationships that exist between categorical variables, preparing the data for deeper statistical modeling and decision-making processes.

Additional Resources for Advanced Analysis

For those interested in performing advanced statistical tests, such as generating expected frequency tables for the Chi-Squared test, or customizing the output structure further, we highly recommend consulting the official documentation for the pandas library. These resources provide exhaustive detail on every parameter available within the `crosstab` function.