How to Create a Contingency Table in Python

Authored by

stats writer

December 6, 2025

Creating a contingency table in Python is a fundamental skill for data analysts seeking to understand the relationships between two or more categorical variables. This process leverages the powerful data manipulation capabilities provided by the Pandas library. Specifically, we utilize the `crosstab` function, which efficiently summarizes the joint distribution of these variables. This guide will walk you through the entire process, from setting up your data using a DataFrame object to interpreting the final, insightful results.

The core objective of generating these tables is to transition raw, disorganized data into a structured format that facilitates statistical analysis and decision-making. By structuring the data with variables of interest assigned to the index (rows) and columns, we can quickly identify patterns, frequencies, and dependencies within the dataset. Mastering this technique is essential for anyone working with survey data, experimental results, or any dataset involving nominal or ordinal measurements.

Understanding Contingency Tables and Their Purpose

A contingency table, also known as a cross-tabulation or cross-tab, is a matrix format used in statistics to display the frequency distribution of the variables. Its primary role is to showcase the interdependence between two or more variables, allowing researchers to determine if there is a relationship between different categories. For instance, we might want to know if the type of product purchased is dependent on the geographic region of the order. The table structure makes this comparison visually and numerically immediate.

Statistically, contingency tables form the basis for tests like the Chi-squared test of independence, which helps quantify whether the observed differences in frequencies across categories are statistically significant or merely due to random chance. When analyzing survey results, quality control data, or customer demographics, the ability to quickly summarize the joint occurrences of different characteristics is invaluable. The table provides not just counts, but a clear snapshot of how categories interact within the dataset.

To effectively utilize this tool, it is paramount that the variables being analyzed are categorical variables--meaning they fall into distinct groups or categories (e.g., gender, country, product type). While numerical data can sometimes be grouped or binned to create categories, the native function of the contingency table shines when applied directly to predefined categorical fields.

Prerequisite: The Pandas **crosstab** Function

The Pandas library in Python provides the specialized `crosstab` function, which is optimized for calculating the cross-tabulation of two or more factors. Unlike using `groupby()` and `count()` manually, `crosstab` offers a streamlined and highly readable approach to creating these frequency

matrices. It handles missing data gracefully and provides built-in options for normalization and margin totals.

The basic syntax for the function is remarkably simple, requiring only the two arrays or series that represent the categorical variables you wish to compare. This simplicity belies the powerful statistical foundation upon which it operates. Understanding the input parameters is key to structuring the output accurately:

pandas.crosstab(index, columns)

Where the arguments specify how the variables map onto the final table structure:

index: This parameter specifies the array, Series, or list of variables whose unique values will define the **rows** of the resulting contingency table.
columns: This parameter specifies the array, Series, or list of variables whose unique values will define the **columns** of the resulting contingency table.

The subsequent steps will demonstrate how to apply this function practically to a real-world product order dataset, illustrating its efficiency and clarity in data analysis.

Step 1: Preparing the Data and Importing Libraries

Before generating the table, we must ensure our data is loaded into a suitable format, typically a Pandas DataFrame. For this example, we will simulate a dataset representing 20 product orders, tracking two key categorical variables: the specific type of product purchased (TV, Computer, or Radio) and the country (A, B, or C) from which the order originated. This setup allows us to easily investigate the relationship between product preference and country.

The first step involves importing the necessary libraries--in this case, only Pandas is required--and then constructing the DataFrame. Pay close attention to how the data is structured, as the 'Product' and 'Country' columns will serve as our input variables for the cross-tabulation.

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, `df`, now holds the raw order information. The data is ready for the cross-tabulation analysis. We will use 'Country' as the row variable (index) and 'Product' as the column variable (columns) in the next step to count the occurrences of each unique country-product combination.

Step 2: Generating the Basic Contingency Table

With the data prepared, we can now invoke the `crosstab` function from Pandas. By passing the Series representing 'Country' to the `index` parameter and the Series representing 'Product' to the `columns` parameter, we instruct Pandas to count every intersection between the unique values of these two fields. The resulting matrix provides the absolute frequency counts for every combination.

This simple command is the core operation for deriving insight from the joint distribution of the two variables. It aggregates the individual order records into a highly condensed and statistical format. The output immediately reveals which product categories are most popular within each country grouping.

#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

Interpretation of the Initial Contingency Table

The generated matrix is a standard contingency table where the rows represent the 'Country' categories (A, B, C) and the columns represent the 'Product' categories (Comp, Radio, TV). Each cell value denotes the joint frequency--the number of orders where the specified country and product combination occurred. Analyzing these frequencies provides direct answers regarding distribution:

A total of **1** computer was purchased from country A, while **3** TVs were purchased from country A. This suggests a strong preference for TVs over computers and radios in country A, based on this sample. Country B saw purchases of **3** computers, **2** radios, and **3** TVs, indicating a more balanced distribution across the product types compared to Country A. Country C purchased **2** computers, **3** radios, and **3** TVs. Radios and TVs appear slightly more dominant than computers in this market.

Specifically, if we examine the individual cells, we can derive highly detailed interpretations:

A total of **1** computer was purchased from country A.
A total of **3** computers were purchased from country B.
A total of **2** computers were purchased from country C.
A total of **0** radios were purchased from country A.
A total of **2** radios were purchased from country B.
A total of **3** radios were purchased from country C.
A total of **3** TV's were purchased from country A.
A total of **3** TV's were purchased from country B.
A total of **3** TV's were purchased from country C.

The total number of orders accounted for in this table is 20, which matches our original dataset size. This initial table is useful for understanding joint frequencies, but it lacks the marginal totals necessary for percentage calculations and overall distribution context.

Step 3: Enhancing the Table with Margin Totals

While the basic table shows the counts for joint occurrences, analysts often require the row totals,

column totals, and the grand total to fully contextualize the data. These totals, known as marginal frequencies, are crucial for calculating conditional and marginal probabilities. The `crosstab` function allows us to easily incorporate these totals using the `margins=True` argument.

By adding `margins=True`, Pandas automatically calculates and appends a row and a column labeled 'All'. This 'All' dimension provides the marginal sums for each category, offering a complete overview of the dataset distribution across both variables simultaneously.

#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

Interpreting Marginal Frequencies

The enhanced table provides deep insights through its marginal totals (the 'All' row and 'All' column), allowing us to analyze the total distribution of orders across both dimensions independently.

Row Totals:

The 'All' column summarizes the total number of orders originating from each country, regardless of the product type:

A total of **4** orders were placed from country A.
A total of **8** orders were placed from country B.
A total of **8** orders were placed from country C.

Column Totals:

The 'All' row summarizes the total number of times each product was ordered across all countries, regardless of origin:

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

Globally, TVs are the most frequently purchased product, followed closely by computers and then radios. The value in the bottom right corner of the table shows that a total of **20** products were ordered from all countries, representing the grand total of the DataFrame.

Advanced Analysis: Calculating Relative Frequencies

While absolute counts are informative, calculating relative frequencies (percentages or proportions) often provides a clearer picture of the relationship between variables. Relative frequencies can be calculated based on the total table count, row totals, or column totals, depending on the analytical goal.

The `crosstab` function supports this directly using the `normalize` argument. Setting `normalize='all'` calculates the percentage of the grand total; `normalize='index'` calculates row percentages (conditional on the country); and `normalize='columns'` calculates column percentages (conditional on the product).

For example, to find the distribution of products within each country (row percentages), we normalize by the index:

#calculate row percentages (conditional distribution)
pd.crosstab(index=df, columns=df, normalize='index')

Product Comp Radio TV
Country
A 0.250000 0.000000 0.750000
B 0.375000 0.250000 0.375000
C 0.250000 0.375000 0.375000

This normalized table shows that in Country A, 75% of orders were for TVs, while in Country C, the distribution was more even, with radios and TVs each accounting for 37.5% of local orders. Utilizing Python and DataFrames allows for seamless transition between raw counts and contextualized proportions, providing the analyst with a comprehensive toolkit for examining categorical variables relationships.