How to Calculate Multinomial Distribution Probabilities in Python

Authored by

stats writer

December 2, 2025

ARABPSYCHOLOGY.COM

RECOMMENDED CITATION

stats writer (2025). *How to Calculate Multinomial Distribution Probabilities in Python*. PSYCHOLOGICAL SCALES. Retrieved from https://scales.arabpsychology.com/?p=103575

Understanding and applying the Multinomial Distribution is essential for analyzing experiments where outcomes fall into more than two categories. Unlike the Binomial distribution, which models binary success/failure trials, the Multinomial distribution allows us to calculate the probability of obtaining specific counts across multiple possible outcomes in a fixed number of independent trials. This powerful statistical tool is implemented efficiently within Python, primarily through the scipy.stats module.

The `scipy.stats.multinomial` class provides a robust framework for handling these multi-category probability calculations. It requires defining three key parameters: the total number of trials (n), a list of probabilities associated with each distinct event (p), and the specific combination of observed outcomes (x). By utilizing its built-in methods, data scientists and analysts can accurately model scenarios ranging from predicting election results and sports performance to complex business inventory decisions.

The Mathematical Definition

Formally, the Multinomial distribution describes the probability mass function (PMF) of obtaining a specific configuration of counts (x1, x2, ..., xk) for *k* different, mutually exclusive outcomes. This calculation is valid provided that the probability (pi) of each outcome remains fixed across all independent trials, and the sum of all probabilities equals one (Σpi = 1).

If a random vector *X* follows a multinomial distribution, the probability of observing exactly x1 occurrences of outcome 1, exactly x2 occurrences of outcome 2, up to xk occurrences of outcome *k*, given *n* total trials, is determined by the following formula:

Probability = n! * (p1x1 * p2x2 * ... * pkxk) / (x1! * x2! ... * xk!)

This formula incorporates the multinomial coefficient (the first part of the expression) multiplied by the product of the individual probabilities raised to their respective observed counts. Understanding these variables is key to successful implementation:

n: The **total number of trials or events** in the experiment.
xi: The specific **number of times outcome *i* occurs** (where i ranges from 1 to k).
pi: The fixed **probability that outcome *i* occurs** in a single, given trial.

Implementing the Multinomial PMF in Python

To practically solve problems involving multi-category probabilities, we leverage the statistical capabilities of the SciPy library. Specifically, we import the `multinomial` object from `scipy.stats`. This object provides the `pmf` method, which calculates the Probability Mass Function directly, saving us from manually implementing the complex factorial computations shown in the theoretical

ARABPSYCHOLOGY.COM

formula.

The `multinomial.pmf()` function requires three arguments: the specific counts of outcomes (`x`), the total number of trials (`n`), and the corresponding list of category probabilities (`p`). Ensuring the order of probabilities in the `p` vector aligns correctly with the order of counts in the `x` vector is essential for accurate calculation.

The following detailed examples illustrate how to structure your input parameters--the counts vector (x), the total trials (n), and the probability vector (p)--to use the `multinomial.pmf()` function effectively in Python.

Example 1: Election Forecasting with Three Candidates

Consider a hypothetical election scenario involving three mayoral candidates: Candidate A, Candidate B, and Candidate C. Based on polling data, the known underlying probabilities of a voter selecting each candidate are: A receives 10% (pA=0.10), B receives 40% (pB=0.40), and C receives 50% (pC=0.50). Since the probabilities sum to 1.0, this situation perfectly fits the criteria for the **Multinomial distribution**.

We are interested in calculating the exact probability of a specific outcome when we sample a small group of voters. Specifically, if we take a random sample of 10 voters (n=10), what is the probability that we observe 2 votes for Candidate A (xA=2), 4 votes for Candidate B (xB=4), and 4 votes for Candidate C (xC=4)?

We define our input vectors for the `multinomial.pmf` function: the outcome counts (x), the total trials (n), and the category probabilities (p). The calculation proceeds as follows:

from scipy.stats import multinomial

# Define n (total trials) = 10
# Define p (probabilities) =
# Define x (observed counts) =
multinomial.pmf(x=, n=10, p=)

0.05040000000000001

The resulting probability mass is precisely **0.0504**. This means there is approximately a 5.04% chance of observing this exact distribution of votes (2 for A, 4 for B, 4 for C) within a random sample of 10 voters. This demonstrates the power of the PMF in pinpointing specific probabilistic outcomes.

ARABPSYCHOLOGY.COM

Example 2: Probability in Sampling with Replacement (Urn Problem)

The Multinomial distribution is particularly useful for analyzing sampling scenarios where replacement ensures the independence of trials, maintaining fixed probabilities. Consider an urn containing 10 total marbles: 6 yellow marbles, 2 red marbles, and 2 pink marbles.

First, we must determine the base probabilities (pi) for drawing a single color. Since there are 10 marbles total: P(Yellow) = 6/10 = 0.6, P(Red) = 2/10 = 0.2, and P(Pink) = 2/10 = 0.2. Our experiment consists of randomly selecting 4 balls (n=4) **with replacement**, which is critical for meeting the independence requirement of the distribution.

We aim to find the probability that all 4 balls selected are yellow. This translates to the specific outcome vector x=, where 4 yellows, 0 reds, and 0 pinks are observed. We feed these parameters into the `multinomial.pmf` function:

from scipy.stats import multinomial

# n=4 trials, p=
# x= (4 Yellow, 0 Red, 0 Pink)
multinomial.pmf(x=, n=4, p=)

0.1295999999999999

The computed probability is approximately **0.1296**. This result confirms that when modeling independent, multi-category events, the scipy.stats implementation handles the necessary combinatorics and probability calculations seamlessly, even when some outcome counts (xi) are zero.

Example 3: Modeling Multi-Outcome Sports Results

The Multinomial distribution is ideal for sports analysis, especially when outcomes involve more than just a win or loss (e.g., ties, draws, or multiple specific score combinations). Consider a chess match played between Student A and Student B, where three distinct outcomes are possible for any single game: A wins, B wins, or they tie.

We are given the fixed probabilities for these outcomes: P(A Wins) = 0.5, P(B Wins) = 0.3, and P(Tie) = 0.2. Note that 0.5 + 0.3 + 0.2 = 1.0. If the students agree to play a total of 10 games (n=10), we want to calculate the likelihood of a very specific final score distribution: A wins 4 times (xA=4), B wins 5 times (xB=5), and 1 game ends in a tie (xTie=1).

Using Python and the `multinomial.pmf` function, we input the total trials (n=10), the probability

ARABPSYCHOLOGY.COM

vector p=, and the outcome count vector x=. This immediately yields the required probability:

from scipy.stats import multinomial

# n=10 games played
# p= (Probabilities for A win, B win, Tie)
# x= (Observed counts)
multinomial.pmf(x=, n=10, p=)

0.03827249999999997

The precise probability that player A wins 4 times, player B wins 5 times, and they tie 1 time is about **0.038**. This low probability highlights how unlikely a highly specific sequence of events is, even if the individual probabilities are relatively high for certain outcomes.

Summary and Further Learning

The Multinomial Distribution is a fundamental concept in statistics that extends the simple Bernoulli/Binomial model to scenarios involving multiple categories. By mastering its implementation using the `scipy.stats.multinomial.pmf` function in Python, practitioners can accurately model complex real-world phenomena--from political surveys and quality control to biological experiments--that rely on calculating the probabilities of specific count combinations.

Remember that the key requirements for applying this distribution are a fixed number of trials (n), independence between trials (often achieved through replacement or large populations), and fixed probabilities (pi) for each of the k possible outcomes.

For those seeking to delve deeper into the mathematical derivation or explore more advanced applications of this distribution, the following resources and tutorials provide additional comprehensive information:

ARABPSYCHOLOGY.COM