How to Calculate Partial Correlation in Python?

Authored by

stats writer

December 25, 2025

Calculating Partial correlation in Python involves a multi-step process. Initially, one typically prepares the data using a Pandas DataFrame. For calculating partial correlation values between two specific variables while rigorously controlling for the influence of other variables, specialized libraries such as Pingouin or Statsmodels are used. Specifically, functions like `partial_corr()` allow for precise statistical analysis that isolates the relationship of interest, yielding a far more nuanced understanding than simple zero-order correlation.

In the field of statistics, researchers frequently rely on standard correlation techniques to quantify the linear association between two distinct variables. This standard measure, often the Pearson correlation coefficient, is fundamental. However, real-world data is complex, and observed relationships are often confounded or masked by external factors. Consequently, there are many scenarios where the primary objective is to isolate the true relationship between variables A and B, **while carefully controlling for the confounding influence of a third variable, C**. This is precisely where the concept of partial correlation becomes indispensable.

Consider a practical example in educational research: we wish to determine the association between the total number of hours a student dedicates to studying and their final examination score. If we simply calculate the standard correlation, the result might be heavily influenced by the student's inherent academic ability, which is reflected in their current grade status. To obtain a cleaner, more accurate measure of the relationship between study time and exam performance, independent of prior academic performance, we must use a **partial correlation**. This technique mathematically removes the linear effect of the current grade, allowing us to assess the isolated link between hours studied and the final score.

This comprehensive tutorial is designed to guide you through the process of calculating partial correlation efficiently and correctly within the Python programming environment, utilizing modern statistical libraries optimized for this task.

Understanding Partial Correlation: The Need for Control

Simple or zero-order correlation calculates the overall relationship between two variables without accounting for any other factors in the environment. While useful for initial exploration, it often suffers from spurious correlation--relationships that appear strong but are actually driven by an unmeasured or unaddressed third variable. The primary goal of partial correlation is to refine this measurement by calculating the correlation between two residuals: the residual of Y regressed on Z, and the residual of X regressed on Z. By modeling out the effect of Z (the covariate), the remaining relationship is interpreted as the pure, isolated association.

The mathematical formulation for partial correlation involves matrix algebra, often relying on the inversion of a relevant correlation matrix or using specific regression techniques. When interpreted

correctly, the resulting partial correlation coefficient (often denoted as $r_{xy.z}$) provides the degree of linear association between X and Y that is independent of Z. This allows researchers to distinguish direct effects from indirect or mediated effects, leading to robust conclusions in areas like social sciences, economics, and machine learning feature selection.

In the context of data analysis using Python, we rely on established packages that encapsulate these complex calculations into straightforward functions. While libraries like Statsmodels offer capabilities for controlling variables through regression, the Pingouin library, specifically designed for statistical analyses, provides exceptionally clean and focused functions for calculating partial correlation coefficients directly, which we will use in our examples below.

Setting Up the Sample Dataset in Python

To illustrate the computation of partial correlation, we will work with a hypothetical dataset representing student performance. We assume we have tracked three key metrics for a small group of students: their academic standing before the examination, the time they spent preparing, and their final results. This data is best structured using the fundamental data structure of the Pandas library, the Pandas DataFrame, which facilitates easy data manipulation and statistical processing.

The three variables we will analyze are: `currentGrade` (the baseline academic measure), `hours` (total hours studied), and `examScore` (the dependent variable). We must first import the necessary libraries, **NumPy** and **Pandas**, and then construct the DataFrame as shown in the code block below. This step ensures that our data is properly organized and ready for statistical analysis using Pingouin.

Suppose we have the following Pandas DataFrame that displays the current grade, total hours studied, and final exam score for 10 students:

import numpy as np
import panda as pd

data = {'currentGrade': ,
'hours': ,
'examScore': ,
}

df = pd.DataFrame(data, columns =)
df

currentGrade hours examScore
0 82 4 88

1 88 3 85
2 75 6 76
3 74 5 70
4 93 4 92
5 97 5 94
6 83 8 89
7 90 7 85
8 90 4 90
9 80 6 93

Calculating Bivariate Partial Correlation Using Pingouin

Our primary analytical goal is to find the isolated relationship between `hours` studied and the `examScore`, after removing the effect of `currentGrade`. This requires calculating a bivariate partial correlation. The Pingouin package is highly recommended for this task due to its statistical rigor and ease of use. If you have not yet installed it, the installation command is included in the subsequent code block. Once installed and imported, the `pg.partial_corr()` function is the tool of choice.

The syntax of the `partial_corr()` function is intuitive and requires specifying the dataset, the two variables of interest (X and Y), and the variable(s) acting as controls or covariates (`covar`). It is critical to ensure the variable names passed into the function exactly match the column names in the Pandas DataFrame.

To calculate the partial correlation between **hours** and **examScore** while controlling for **currentGrade**, we use the **partial_corr()** function. This function adheres to the following general syntax structure, making it clear how data roles are assigned:

partial_corr(data, x, y, covar)

The parameters utilized in this function are defined as follows:

data: This represents the name of the input DataFrame containing all necessary variables.
x, y: These are the names of the two primary columns whose relationship we intend to measure.
covar: This specifies the name of the covariate column(s) in the DataFrame--the variable(s) whose effect must be statistically controlled or removed.

Executing the Bivariate Partial Correlation Code

Having defined the variables and understood the function syntax, we now execute the code to perform the calculation. The output provided by Pingouin is a detailed table that goes beyond just

the correlation coefficient, including statistical metrics crucial for inference, such as the p-value and statistical power. We are particularly interested in the `r` column, which holds the final partial correlation value.

The following code block demonstrates the necessary installation and importation steps, followed by the specific function call that calculates the partial correlation between 'hours' and 'examScore', controlling for 'currentGrade'. Note the structured output that facilitates immediate statistical reporting.

#install and import pingouin package
pip install pingouin
import pingouin as pg

#find partial correlation between hours and exam score while controlling for grade
pg.partial_corr(data=df, x='hours', y='examScore', covar='currentGrade')

n r CI95% r2 adj_r2 p-val BF10 power
pearson 10 0.191 0.036 -0.238 0.598 0.438 0.082

Interpreting the Bivariate Partial Correlation Results

The executed code yielded a comprehensive output table. The key metric we need to focus on is the value reported under the `r` column, which is the partial correlation coefficient. In this specific case, the partial correlation between hours studied and final exam score, after factoring out the influence of current grade, is calculated as **0.191**. This value represents a small, positive linear relationship. As hours studied increases, exam score tends to increase as well, assuming current grade is held constant.

The interpretation is straightforward: assuming the student's current grade (the control variable) is held constant, there is a tendency for the final exam score to increase slightly as the number of hours studied increases. Importantly, this value (0.191) is likely significantly smaller than the zero-order correlation (which would not control for current grade), indicating that a large portion of the overall observed association between studying and high scores was actually attributable to the student's baseline ability. Furthermore, the `p-val` (0.598) indicates that this small correlation is not statistically significant given our small sample size ($n=10$), suggesting we cannot confidently reject the null hypothesis that the true partial correlation is zero.

Understanding the difference between simple and partial correlation is crucial here. If the initial simple correlation between hours and exam score was high, say 0.75, and the partial correlation dropped to 0.191, it implies that `currentGrade` is a strong intermediary variable. The high simple correlation was largely spurious, driven by the fact that students with high current grades naturally

study more (or less, depending on the dynamic) and also tend to score higher on the exam.

Calculating the Full Partial Correlation Matrix

While the `pg.partial_corr()` function is excellent for focusing on a specific bivariate relationship while controlling for one or more covariates, researchers often require the partial correlation coefficients for all possible pairs of variables within the dataset, controlling for all remaining variables simultaneously. This comprehensive overview is provided by the `.pcorr()` method, which is available directly on a Pandas DataFrame. This method uses the underlying statistical algorithms for calculation, often based on the inverse of the correlation matrix.

To calculate all pairwise partial correlations in a dataset, we simply call the `.pcorr()` method on our DataFrame `df`. Each cell (i, j) in the resulting matrix represents the partial correlation between variable i and variable j, controlling for all other variables present in the DataFrame. For clean presentation, we typically round the output to a manageable number of decimal places, such as three, as shown in the example below, ensuring the output is easy to read and report.

To calculate the partial correlation between multiple variables at once, generating a complete partial correlation matrix, we use the **.pcorr()** function:

#calculate all pairwise partial correlations, rounded to three decimal places
df.pcorr().round(3)

currentGrade hours examScore
currentGrade 1.000 -0.311 0.736
hours -0.311 1.000 0.191
examScore 0.736 0.191 1.000

Detailed Interpretation of the Full Matrix Output

The resulting partial correlation matrix provides three distinct partial correlation coefficients (excluding the diagonal 1.000 values). Each off-diagonal value must be interpreted as the correlation between the row variable and the column variable, with the effects of the third, remaining variable partialled out. Due to the symmetrical nature of correlation, the matrix values are duplicated (e.g., the correlation between Current Grade and Hours is the same as Hours and Current Grade).

The way to interpret the output is as follows:

The partial correlation between current grade and hours studied is **-0.311**. This suggests that among students who achieve the same `examScore`, those with higher baseline current grades tend

to study slightly fewer hours. This is an important negative relationship revealed only after controlling for the outcome variable.

The partial correlation between current grade and exam score is **0.736**. This represents a strong positive association. This means that even after controlling for the total number of hours studied, a student's prior academic performance (current grade) remains a highly significant and strong predictor of their final exam score.

The partial correlation between hours studied and exam score is **0.191**. This confirms the earlier bivariate calculation, showing that controlling for `currentGrade` significantly attenuates the relationship between study hours and exam score.

Conclusion: Leveraging Partial Correlation in Data Science

The ability to calculate and interpret partial correlation is a fundamental skill in advanced Python-based data analysis and statistics. Simple correlations can be misleading, especially in complex multivariate systems common in research and business intelligence. By utilizing techniques available in libraries like Pingouin, we can effectively remove confounding factors (covariates) and isolate the true, direct linear relationship between variables of interest.

Whether you are performing feature selection for a machine learning model, trying to establish causality in observational studies, or simply cleaning up spurious relationships in exploratory data analysis, the use of functions like `partial_corr()` and `.pcorr()` provides robust and transparent results. Always remember that partial correlation measures only the linear association and assumes that the relationship being controlled for is itself linear. Thorough interpretation, considering the statistical context alongside the correlation coefficient, is essential for drawing statistically sound conclusions.