How to Calculate Partial Correlation in Python with Statsmodels

Authored by

stats writer

March 15, 2026

How can partial correlation be calculated in Python?

The field of **statistics** offers a variety of tools to uncover the hidden relationships between variables. One of the most common metrics used in **data science** is the **Pearson correlation coefficient**, which quantifies the linear association between two factors. However, in complex real-world scenarios, a simple correlation can be misleading. Often, a third variable--known as a **confounding variable**--influences both variables of interest, creating a "spurious" relationship. To address this, researchers utilize **partial correlation**, a sophisticated method that isolates the unique relationship between two variables while controlling for the effects of one or more additional covariates.

In the **Python** programming ecosystem, calculating these values has become remarkably efficient thanks to specialized libraries like **Pingouin** and **Statsmodels**. These libraries provide robust functions to strip away the "noise" introduced by outside factors, allowing for a clearer view of the underlying data structure. Whether you are analyzing economic trends, psychological behaviors, or biological processes, the ability to control for secondary variables is essential for producing accurate and actionable insights. This guide provides a comprehensive walkthrough of implementing partial correlation using Python, ensuring your statistical conclusions are grounded in rigorous methodology.

By using partial correlation, you essentially perform a **linear regression** where you predict both your primary variables from the controlling variable. You then calculate the correlation between the **residuals** of those regressions. This process ensures that the resulting coefficient represents only the variance shared exclusively by the two main variables, independent of the influence exerted by the **covariate**. This level of detail is crucial for establishing **causality** and understanding the direct impact of interventions in experimental and observational studies.

The Conceptual Foundation of Partial Correlation

Understanding the difference between zero-order correlation and partial correlation is fundamental for any **analytical** framework. Zero-order correlation refers to the standard correlation between two variables without adjusting for any other factors. While useful, it can be deceptive if a third variable drives the movement in both. For instance, there is a high correlation between ice cream sales and drowning incidents; however, this is not a causal link but rather a result of the temperature. In this classic example, temperature is the confounding variable. By applying partial correlation, we can control for temperature, which would likely reveal that the direct relationship between ice cream and drowning is near zero.

The mathematical logic behind this involves removing the "shared variance." Imagine three overlapping circles in a **Venn diagram** representing variables X, Y, and Z. The standard

correlation between X and Y includes the area where X and Y overlap, including any area where Z also overlaps with them. Partial correlation effectively removes the portion of the overlap that is attributable to Z. This ensures that the remaining correlation coefficient reflects the pure association between X and Y. This technique is widely used in **multivariate statistics** to build more accurate predictive models and to refine **hypotheses** during the exploratory phase of research.

Furthermore, partial correlation is an extension of the **general linear model**. It assumes that the relationships between the variables are linear and that the data follows a **normal distribution**. If these assumptions are violated, the results may be biased. Therefore, it is standard practice to perform **exploratory data analysis** (EDA) to check for **outliers** and non-linear patterns before proceeding with the calculation. In Python, tools like **Matplotlib** and **Seaborn** are invaluable for visualizing these relationships and ensuring the data is suitable for partial correlation analysis.

Setting Up the Python Environment for Statistical Analysis

To begin calculating partial correlation, you must first ensure that your **Python** environment is equipped with the necessary **libraries**. The primary library we will use is **Pandas**, which is the industry standard for data manipulation and analysis. Pandas allows you to store your data in a **DataFrame**, a powerful two-dimensional table-like structure that makes it easy to reference columns and perform complex operations. Additionally, we will need **NumPy** for numerical computations and the **Pingouin** package, which is specifically designed for simple yet exhaustive statistical tests.

Installation of these packages can be handled easily via the **pip** package manager. By running simple commands in your terminal or command prompt, you can fetch the latest versions of these tools from the **Python Package Index**. Once installed, these libraries work in harmony to provide a seamless workflow. Pandas handles the data ingestion and cleaning, NumPy provides the underlying mathematical engine, and Pingouin offers high-level functions that return detailed statistical tables, including **p-values** and **confidence intervals**, which are often missing from other basic libraries.

It is also worth noting that while **Statsmodels** is a more comprehensive library for econometric modeling, Pingouin is often preferred for partial correlation due to its concise syntax and informative output. Pingouin's `partial_corr` function is particularly user-friendly, as it accepts a Pandas DataFrame directly and allows you to specify the columns of interest with simple string identifiers. This reduces the amount of **boilerplate code** you need to write, allowing you to focus on the interpretation of your results rather than the intricacies of the programming logic.

Example: Partial Correlation in Python

To illustrate the practical application of this method, let us consider a scenario involving student performance. Suppose we have a **DataFrame** containing three variables: the student's current grade in a class, the total number of hours they studied for the final exam, and the score they ultimately received on that exam. We might find a high correlation between hours studied and the exam score. However, we also suspect that students who already have a high current grade might study differently or have a different baseline of knowledge. Therefore, we want to measure the relationship between study hours and exam scores while **controlling** for the current grade.

The first step in our Python implementation is to define the dataset. We use a dictionary to represent our columns and then convert it into a Pandas object. This structured approach mimics how real-world data is often imported from **CSV** files or **SQL** databases. Once the data is in a DataFrame, we can easily inspect its contents, calculate summary statistics, and visualize the distributions of each variable to identify any potential issues before running our main analysis.

import numpy as np
import pandas 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

As seen in the data above, there are variations across all three metrics. Without a partial correlation analysis, we might mistakenly attribute the exam score solely to the number of hours

studied, neglecting the fact that a student's prior performance (current grade) is a significant **predictor** of their future success. By using Python to isolate these effects, we can gain a more nuanced understanding of educational outcomes and the efficiency of study habits.

Implementing the partial_corr() Function

To calculate the specific partial correlation between **hours** and **examScore** while controlling for **currentGrade**, we utilize the `partial_corr()` function from the Pingouin library. This function is designed to be highly intuitive. It requires four primary arguments: the name of the DataFrame, the two variables you are comparing (x and y), and the covariate (covar) that you wish to hold constant. This allows the mathematical model to account for the shared variance between the covariate and the other variables before calculating the final **correlation** coefficient.

The syntax for this function is as follows: **partial_corr(data, x, y, covar)**. The beauty of this implementation is that it returns a complete statistical summary in the form of a Pandas DataFrame. This summary includes the number of samples (n), the correlation coefficient (r), the **95% confidence interval** (CI95%), the coefficient of determination (r2), and the **p-value**. Having all these metrics in one table is incredibly useful for validating the **statistical significance** of your findings.

#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

In this specific example, the output reveals a partial correlation coefficient of **0.191**. This suggests a weak positive relationship between study hours and exam scores when current grades are accounted for. Interestingly, the high **p-value** (0.598) indicates that this relationship is not statistically significant at the standard 0.05 level. This insight is vital; it suggests that for this specific group of students, simply increasing study hours may not lead to a predictable increase in exam scores once their existing academic standing is taken into consideration.

Deep Dive into the Statistical Output

When analyzing the results of a partial correlation, it is important to look beyond the "r" value. The **Bayes Factor** (BF10) provided in the Pingouin output offers an alternative to traditional p-values

for **hypothesis testing**. A BF10 value less than 1, as seen in our result (0.438), provides evidence in favor of the **null hypothesis**, suggesting that there is no meaningful relationship between the variables after controlling for the covariate. This Bayesian approach is increasingly popular in modern **data analysis** because it quantifies the strength of evidence for both the null and alternative hypotheses.

Another critical metric is **statistical power**. In our output, the power is quite low (0.082), which is largely due to the very small **sample size** of only 10 students. Low power means that even if a true relationship existed in the broader population, our specific test was unlikely to detect it. This highlights the importance of having an adequate sample size when conducting **correlation analysis**. Researchers must be cautious not to over-interpret results from small datasets, even when using advanced methods like partial correlation.

The **coefficient of determination** (r2) is also provided, which in this case is 0.036. This means that only about 3.6% of the variance in exam scores can be uniquely explained by study hours after the current grade has been accounted for. The remaining variance is likely due to other factors not included in our model, such as student motivation, the difficulty of the exam, or even random chance. By examining the adjusted r2 and the confidence intervals, analysts can build a more realistic picture of the **effect size** and the reliability of their measurements.

Calculating Pairwise Partial Correlations

In many research projects, you may have more than three variables and wish to see the partial correlations between all possible pairs while controlling for all other variables in the dataset. This is known as a partial correlation matrix. Python's Pandas library offers a convenient method for this called `.pcorr()`. This function calculates the pairwise partial correlation of all columns in a DataFrame, effectively treating all other columns as covariates for each specific pair calculation. This is an incredibly efficient way to identify the "pure" relationships across a complex dataset with many **variables**.

Using the `.pcorr()` function is straightforward and requires no additional arguments if you want to analyze the entire DataFrame. The output is a matrix where the diagonal values are always 1.000 (since every variable perfectly correlates with itself), and the off-diagonal values represent the partial correlations between the respective row and column variables. To make the matrix easier to read, it is often helpful to round the results to a specific number of **decimal places** using the `.round()` method.

#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

The interpretation of this matrix is vital for understanding the broader system of variables. For example, we see that the partial correlation between **currentGrade** and **examScore** is 0.736. This is a strong positive correlation, indicating that even when we control for study hours, the student's prior grade is a very strong predictor of their final exam performance. Conversely, the partial correlation between **currentGrade** and **hours** is -0.311, suggesting a slight negative relationship; perhaps students with higher grades feel less pressure to study for as many hours.

Practical Applications and Best Practices

Partial correlation is widely used in the **social sciences**, **epidemiology**, and **economics** to untangle complex webs of influence. In medical research, for instance, partial correlation can be used to determine the relationship between a specific lifestyle factor (like exercise) and a health outcome (like heart disease) while controlling for age, smoking status, and body mass index. Without these controls, the results could be significantly skewed by the fact that older people might exercise less and also have higher rates of heart disease regardless of their physical activity level.

When implementing these methods in Python, it is important to follow best practices for **data cleaning**. Ensure that your dataset does not contain missing values, as most statistical functions will either return an error or produce an **NaN** result if they encounter null entries. You can use Pandas' `df.dropna()` or `df.fillna()` methods to handle these cases. Additionally, be mindful of **multicollinearity**, which occurs when two or more of your control variables are highly correlated with each other. This can lead to unstable estimates and make it difficult to determine the unique contribution of each factor.

Finally, always remember that partial correlation, like all correlation metrics, does not prove **causation**. It only indicates that a relationship exists after accounting for specific other variables. To establish a causal link, researchers must rely on **experimental designs** or more advanced causal inference techniques. Nonetheless, Python's ability to quickly and accurately calculate partial correlations makes it an indispensable part of any modern analyst's toolkit, providing a deeper level of insight than simple correlation ever could.

Summary of Key Findings

Partial correlation is a vital statistical technique for isolating the relationship between two variables by removing the influence of **confounding factors**.
The **Pingouin** library in Python provides a user-friendly `partial_corr()` function that simplifies

complex statistical calculations.
Calculations are based on the **residuals** of linear regressions, ensuring that the resulting coefficient represents only unique shared variance.
Interpreting the **p-value** and **confidence intervals** is essential for determining if the observed relationship is statistically significant or due to chance.
The `.pcorr()` method in **Pandas** allows for the rapid generation of a partial correlation matrix, which is ideal for datasets with multiple interacting variables.
Always assess **statistical power** and **sample size** before drawing broad conclusions from your data analysis.
While partial correlation clarifies relationships, it should be used in conjunction with other **multivariate** techniques to fully understand complex data systems.

Identify the variables of interest and the potential **covariates** that may influence the results.
Import your data into a **Pandas DataFrame** and perform necessary cleaning and preprocessing steps.
Install and import the **Pingouin** library to access advanced statistical functions.
Run the `partial_corr()` function to get a detailed breakdown of the unique relationship between your primary variables.
Examine the **correlation coefficient (r)** and the **p-value** to evaluate the strength and significance of the relationship.
Consider calculating the **pairwise partial correlation matrix** for a holistic view of your entire dataset.
Document your findings, noting the specific variables controlled for to ensure **reproducibility** in your research.