Learning Guide: Extracting P-Values from Linear Regression Models using Statsmodels in Python

Authored by

Mohammed looti

October 28, 2025

When conducting **linear regression** analysis in **Python**, particularly using the robust **Statsmodels** library, the ability to accurately understand and extract the **p-values** associated with your model's **coefficients** is paramount. These values are the cornerstone of hypothesis testing, determining the **statistical significance** of each **predictor variable** in explaining the variation observed in the response. This comprehensive guide details various programmatic methods to precisely extract these **p-values** from a fitted **linear regression model** generated via the **Statsmodels** library, ensuring you gain access to the full numerical precision required for rigorous data analysis.

The **Statsmodels** library provides a convenient interface to retrieve **p-values** directly from the results object following model fitting. You can easily access the entire collection of **p-values** for all **predictor variables**, including the **intercept**, or opt to target a specific **coefficient** based on its label (name) or its sequential index within the model structure. The following code demonstrates the basic access patterns before we dive into a practical implementation:

 # Extract p-values for all predictor variables using a loop
for x in range (0, 3): # Assuming 3 coefficients (including intercept)
print(model.pvalues)

Extract p-value for a specific predictor variable by its name
model.pvalues.loc

Extract p-value for a specific predictor variable by its positional index
model.pvalues # Typically the intercept

Understanding P-Values: The Core Concept

The **p-value** represents a core concept within inferential statistics and **hypothesis testing**. In the context of **linear regression**, the p-value associated with a **coefficient** quantifies the probability of observing sample data results (or more extreme results) if the **null hypothesis** were actually true. The null hypothesis for any specific **predictor** states that there is no linear relationship between that variable and the response variable, meaning the true population **coefficient** is zero.

A critical step in interpreting regression output is comparing the calculated **p-value** against a predetermined significance level, typically denoted as alpha ($alpha$), which is most commonly set at 0.05. If the calculated p-value is less than this alpha level, we reject the **null hypothesis**. This rejection implies that the evidence is strong enough to conclude that the **predictor variable** is **statistically significant** in influencing the response.

Conversely, if the **p-value** exceeds the alpha threshold (e.g., $p > 0.05$), we fail to reject the **null hypothesis**. In this case, we lack sufficient statistical evidence to conclude that a meaningful linear relationship exists between the **predictor** and the response, given the structure of the model and

the available data. Therefore, understanding and accurately extracting these **p-values** is indispensable for robust model interpretation, effective feature selection, and drawing reliable, defensible conclusions from any **regression** analysis.

Preparing Data and Fitting the Model in Statsmodels

To effectively demonstrate the process of p-value extraction, we must first establish a working environment by preparing a sample dataset and fitting a **linear regression** model using **Statsmodels**. We will utilize a hypothetical student dataset encapsulated within a **pandas DataFrame**, tracking variables such as 'hours studied', 'exams taken', and the resulting 'score'. This structure allows us to simulate a typical data analysis workflow where we seek to model the final score based on the effort variables.

import pandas as pd

# Create a sample DataFrame for demonstration
df = pd.DataFrame({'hours': ,
'exams': ,
'score': })

# Display the first few rows of the DataFrame
df.head()

hours exams score
0 1 1 76
1 2 3 78
2 2 3 85
3 4 5 88
4 2 2 72

Once the data is loaded, we define our **predictor variables** ('hours' and 'exams') and the **response variable** ('score'). The standard approach for fitting linear models in **Statsmodels** is using the **OLS** (Ordinary Least Squares) function from the `statsmodels.api` module. However, a crucial preliminary step for OLS is ensuring the model includes an **intercept** term.

Unlike some statistical packages, **OLS** in **Python** does not automatically include the **intercept**. We must explicitly add this **constant term** to our **predictor variables** matrix using **sm.add_constant()**. This constant represents the expected value of the response when all predictors are zero, and its inclusion is vital for a correctly specified **regression equation**. We then fit the model to generate the results object from which we will extract the **p-values**.

import statsmodels.api as sm

```
# Define response and predictor variables
y = df
x = df]

# Add a constant (intercept) to the predictor variables
x = sm.add_constant(x)

# Fit the Ordinary Least Squares (OLS) linear regression model
model = sm.OLS(y, x).fit()

# Display the comprehensive model summary
print(model.summary())
```

OLS Regression Results

==========

===

Dep. Variable: score R-squared: 0.718

Model: OLS Adj. R-squared: 0.661

Method: Least Squares F-statistic: 12.70

Date: Fri, 05 Aug 2022 Prob (F-statistic): 0.00180

Time: 09:24:38 Log-Likelihood: -38.618

No. Observations: 13 AIC: 83.24

Df Residuals: 10 BIC: 84.93

Df Model: 2

Covariance Type: nonrobust

==========

===

coef std err t P>|t|

---

const 71.4048 4.001 17.847 0.000 62.490 80.319

hours 5.1275 1.018 5.038 0.001 2.860 7.395

exams -1.2121 1.147 -1.057 0.315 -3.768 1.344

==========

===

Omnibus: 1.103 Durbin-Watson: 1.248

Prob(Omnibus): 0.576 Jarque-Bera (JB): 0.803

Skew: -0.289 Prob(JB): 0.669

Kurtosis: 1.928 Cond. No. 11.7

==========

===

Analyzing the Initial OLS Summary

The output generated by the **summary()** function provides a wealth of statistical details, organized in a clear tabular format. Within the main table of **coefficients**, we find the calculated coefficients, their standard errors, the corresponding **t-statistics**, and, most importantly for this discussion, the **p-values** in the column labeled `P>|t|`. This summary offers an immediate assessment of the **statistical significance** for the **intercept** and each predictor variable.

Examining the summary table for our fitted **model**, we observe the following rounded p-values: for the constant ('const'), the p-value is 0.000; for 'hours', it is 0.001; and for 'exams', it is 0.315. While this truncation is perfectly adequate for a quick overview and initial decision-making (e.g., rejecting the null hypothesis for 'const' and 'hours'), it masks the true precision of the underlying calculation.

In situations demanding extreme numerical accuracy, such as publishing academic results or performing complex simulations where threshold comparisons must be exact, relying solely on the rounded values from the summary table may be insufficient. The **Statsmodels** results object stores the full, unrounded floating-point values, and accessing these is necessary to ensure the highest fidelity in your analysis and subsequent reporting.

Detailed Extraction: Accessing Full Precision P-Values

The primary mechanism for accessing the full numerical precision of the **p-values** in **Statsmodels** is through the **`.pvalues`** attribute of the fitted model object. This attribute returns a dedicated **pandas Series**, where the index corresponds to the names of the **coefficients** (e.g., 'const', 'hours', 'exams') and the values hold the exact, unrounded p-values.

To display these values with their complete decimal representation, you can iterate directly over the **`.pvalues` Series** using a standard **`range()`** loop, as demonstrated below. This method is especially useful when performing automated checks or exporting results to specialized statistical software that requires maximum precision.

 # Extract p-values for all predictor variables with full precision
for x in range (0, 3): # Loop through the indices of coefficients (0 for const, 1 for hours, 2 for exams)
print(model.pvalues)

6.514115622692573e-09
0.0005077783375870773
0.3154807854805659

By executing this loop, we reveal the true precision hidden behind the summary table's rounding. For instance, the p-value for the **intercept** is actually $6.514 times 10^{-9}$, confirming its overwhelming **statistical significance**. Note that the value `3` used in our **`range()`** function corresponds exactly to the total number of **coefficients** in our model: the **intercept** plus the two independent variables ('hours' and 'exams'). You must always ensure the loop range matches the dimensions of your specific model.

Targeted Extraction: Using Names and Indices

While extracting all **p-values** is useful for a complete overview, analytical workflows often require retrieving the p-value for only a single, specific **variable's name**. **Statsmodels** provides flexible access mechanisms to handle this targeted extraction, using either the coefficient's label or its position. This capability is invaluable for building streamlined data pipelines or conditional statements that depend on a single variable's significance.

The most robust way to extract a specific p-value is by using its label (name). Since `model.pvalues` is a **pandas Series**, we can utilize the **`.loc`** accessor, passing the variable name as a string. This method is preferred because it is immune to changes in the order of **variables**, making your code more resilient and readable.

 # Extract p-value for the 'hours' variable specifically by its name
model.pvalues.loc

0.0005077783375870773

Alternatively, if you are certain of the variable ordering, you may access the **p-value** using its **positional index**, which follows standard **Python indexing** conventions. Remember that the **intercept** is always at index `0`, followed sequentially by the predictors in the order they were provided to the model.

 # Extract p-value for the coefficient at index position 0 (the intercept)
model.pvalues

6.514115622692573e-09

Interpreting P-Values for Model Significance

After successfully extracting the high-precision **p-values**, the essential final step is interpreting their meaning within the context of our **linear regression model**. We utilize the conventional significance level of $alpha = 0.05$ for decision-making. Reviewing the extracted precise values

from our student score model:

P-value for **intercept** ('const'): $6.514 times 10^{-9}$
P-value for 'hours': 0.0005077
P-value for 'exams': 0.3154807

Both the **intercept** and the 'hours' **predictor variable** exhibit p-values far below 0.05. Consequently, we confidently reject the **null hypothesis** for these **coefficients**. This statistical finding indicates that the intercept is **statistically significant** (the baseline score is not zero) and that 'hours studied' is a highly **statistically significant** predictor of the final score. Furthermore, the positive coefficient (5.1275) confirms the intuitive relationship: more hours studied are strongly associated with a higher final score.

In contrast, the p-value for 'exams' (0.3154807) is significantly higher than our 0.05 threshold. For this variable, we must fail to reject the **null hypothesis**. This result implies that, once the effect of 'hours studied' has been accounted for in the model, the number of 'exams taken' does not demonstrate a **statistically significant** linear relationship with the student's final score in this particular sample. It is crucial to remember that failing to reject the null hypothesis does not prove that the effect is zero; it simply means that the data provides insufficient evidence to claim a significant effect, given the model's structure.

Conclusion and Further Exploration

Mastering the extraction of **p-values** from **Statsmodels** **linear regression models** is a foundational skill for accurate quantitative analysis in **Python**. Whether your objective is rapid interpretation via the model summary or deep numerical analysis requiring full precision, **Statsmodels** provides straightforward, flexible methods using the `.pvalues` attribute.

The ability to programmatically access precise **p-values** enables powerful automation, custom visualization, and the ability to integrate rigorous hypothesis testing directly into automated scripts. Always ensure that the interpretation of these values is grounded in the context of your data, the underlying assumptions of the **linear regression** method, and a cautious awareness of the limitations inherent in statistical inference.

 # Additional snippet for completeness: getting all p-values as a pandas Series
model.pvalues

const 6.514116e-09
hours 5.077783e-04
exams 3.154808e-01
dtype: float64

Additional Resources

To further refine your skills in **statistical modeling** using **Python**, it is recommended to explore concepts related to overall model quality and assumption checks:

Understanding **Confidence Intervals** for **Regression Coefficients**
Evaluating the Model Fit using **R-squared** and **F-statistics**
Checking **Regression Assumptions** (e.g., linearity and normality of residuals)
Performing **Multicollinearity** Diagnostics to ensure reliable coefficient estimates

These topics will help ensure that the **statistical significance** results derived from your model are robust and reliable for drawing practical conclusions.