# How to Easily Perform White’s Test for Heteroscedasticity in Python

Authored by

**stats writer**

December 2, 2025

The evaluation of model assumptions is critical for reliable statistical inference in linear regression. One of the primary tools used to diagnose issues with the variance of errors is White's test.

You can perform White's test efficiently in Python primarily utilizing the statsmodels library. The process involves several key analytical steps: first, establishing a linear model using the OLS method; second, fitting this model to your dataset and extracting the residuals; and finally, executing the White test function, which calculates relevant statistics (like the F-statistic or the Lagrange Multiplier statistic) to determine if the assumption of constant variance is met.

## Understanding Heteroscedasticity and White's Test

**White's test** is a robust diagnostic tool specifically designed to determine if heteroscedasticity is present within the error terms of a regression model. Identifying this condition is vital because heteroscedasticity violates the core assumption of ordinary least squares (OLS) regression that the variance of the residuals is constant across all levels of the predictor variables.

**Heteroscedasticity** refers to the unequal scatter (non-constant variance) of residuals at different levels of the independent variables. If this condition exists, the standard errors calculated for the regression coefficients become unreliable, potentially leading to incorrect conclusions about the statistical significance of the predictors.

The following step-by-step example demonstrates how to implement White's test in Python. We will use the powerful functionality of the `statsmodels` package to rigorously assess whether heteroscedasticity poses a problem in a sample regression analysis.

## Step 1: Load Necessary Libraries and Data

To begin, we must load the necessary Python libraries, including `statsmodels` for the test itself, `sklearn` for general linear modeling tools, and `pandas` for efficient data handling. In this demonstration, we will fit a model using the well-known **mtcars** dataset, which is commonly used for statistical examples.

The code below illustrates how to import the required modules and load the **mtcars** dataset into a pandas DataFrame directly from a hosted URL. We then display the summary information to confirm the data structure and types.

```
from sklearn.linear_model import LinearRegression
from statsmodels.stats.diagnostic import het_white
import statsmodels.api as sm
import pandas as pd

#define URL where dataset is located
```

```
url = "https://raw.githubusercontent.com/arabpsychology/Python-Guides/main/mtcars.csv"

#read in data
data = pd.read_csv(url)

#view summary of data
data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32 entries, 0 to 31
Data columns (total 12 columns):
# Column Non-Null Count Dtype
--- ------ -------------- -----
0 model 32 non-null object
1 mpg 32 non-null float64
2 cyl 32 non-null int64
3 disp 32 non-null float64
4 hp 32 non-null int64
5 drat 32 non-null float64
6 wt 32 non-null float64
7 qsec 32 non-null float64
8 vs 32 non-null int64
9 am 32 non-null int64
10 gear 32 non-null int64
11 carb 32 non-null int64
dtypes: float64(5), int64(6), object(1)
```

## Step 2: Fit the OLS Regression Model

Once the data is prepared, the next step involves defining the model structure and performing the regression analysis. We will use the Miles Per Gallon (**mpg**) as our response variable (Y) and Displacement (**disp**) and Horsepower (**hp**) as the two predictor variables (X).

We use the `OLS` class from the `statsmodels.api` module. It is standard practice in OLS regression using this library to explicitly include a constant (intercept) term, which we achieve using `sm.add_constant(x)` before fitting the model.

The resulting `model` object contains all the necessary components, including the calculated residuals, which will be input directly into the White's test function in the next step.

**#define response variable**

**y = data**

#define predictor variables
x = data]

#add constant to predictor variables
x = sm.add_constant(x)

#fit regression model
model = sm.OLS(y, x).fit()

## Step 3: Executing White's Test in Python

With the OLS model fitted, we can now execute the diagnostic test. We utilize the `het_white` function, imported from `statsmodels.stats.diagnostic`, to perform White's test. This function requires the model residuals (`model.resid`) and the original predictor matrix (`model.model.exog`) as inputs.

The function returns a tuple of results, which we then map to descriptive labels for clarity. These results include the Test Statistic (based on the Lagrange Multiplier, distributed as Chi-squared), its corresponding p-value, the F-Statistic, and its p-value. The output provides the critical metrics needed to make a statistical decision regarding the presence of heteroscedasticity.

**#perform White's test**
**white_test = het_white(model.resid, model.model.exog)**

#define labels to use for output of White's test
labels =

#print results of White's test
print(dict(zip(labels, white_test)))

{'Test Statistic': 7.076620330416624, 'Test Statistic p-value': 0.21500404394263936,
'F-Statistic': 1.4764621093131864, 'F-Test p-value': 0.23147065943879694}

## Interpreting the White's Test Results

The output provides two sets of test results: one based on the Chi-squared distribution (Test Statistic) and one based on the F-distribution (F-Statistic). In practice, the interpretation based on the p-value associated with the Test Statistic is most common.

Based on the execution above, the key results are:

The Lagrange Multiplier Test Statistic (X2) is **7.0766**.
The corresponding p-value is **0.215**.

To interpret these values, we must formally state the hypotheses used in White's test:

**Null Hypothesis (H0)**: Homoscedasticity is present (the variance of residuals is constant).
**Alternative Hypothesis (HA):** Heteroscedasticity is present (the variance of residuals is not constant).

Using a standard significance level ($alpha$) of 0.05, we compare the p-value (0.215) to this threshold. Since the p-value is greater than 0.05, we fail to reject the null hypothesis (H0). This indicates that we do not have sufficient statistical evidence to conclude that heteroscedasticity is a significant issue in this specific linear regression model.

## Remediation Strategies for Heteroscedasticity

If the White's test results lead you to fail to reject the null hypothesis (as in our example), you can proceed confidently with interpreting the output of your original OLS method regression, as the assumption of homoscedasticity is met.

However, if the test yields a p-value less than $alpha$ (e.g., $p < 0.05$), you must reject the null hypothesis, confirming the presence of heteroscedasticity. In this scenario, the standard errors of the regression coefficients are biased and unreliable, potentially invalidating your hypothesis testing results. Two common strategies exist to address this statistical issue:

### 1. Transforming the Response Variable

One effective solution is to apply a mathematical transformation to the response variable. For example, taking the natural logarithm of the response variable often stabilizes the variance of the residuals, thereby causing heteroscedasticity to dissipate. Other transformations, such as the square root or inverse, may also be appropriate depending on the nature of the data.

### 2. Using Weighted Regression (WLS)

Another robust method is employing Weighted Least Squares (WLS) regression. WLS assigns a specific weight to each data point inversely proportional to the variance of its corresponding fitted value. Essentially, observations associated with higher variance (the source of the heteroscedasticity) are given smaller weights in the estimation process. When the proper weights are successfully determined and applied, WLS provides unbiased and efficient standard error estimates, effectively mitigating the problem caused by unequal residual variances.

## Further Resources on Python Regression

For those looking to deepen their understanding of regression analysis and diagnostics in Python, the following tutorials offer additional valuable information: