

# How to perform Quantile Regression in Python?

Authored by  
**stats writer**

December 14, 2025

## RECOMMENDED CITATION

stats writer (2025). *How to perform Quantile Regression in Python?*. PSYCHOLOGICAL SCALES. Retrieved from <https://scales.arabpsychology.com/?p=107427>

Quantile regression is a powerful statistical technique used for modeling the relationship between a set of predictor variables and specific conditional quantiles of the response variable. In the realm of data science, performing quantile regression in Python is most efficiently accomplished using the statsmodels library. This specialized library provides robust tools necessary to fit a regression model to empirical data while specifically targeting a desired quantile, such as the median or the 75th percentile, enabling the precise calculation of regression parameters for that specific point in the distribution.

Beyond the core modeling capabilities, the statsmodels package offers utilities essential for validating the model's accuracy, assessing significance, and visualizing the resulting fits. Furthermore, the widely used scikit-learn library also incorporates a dedicated estimator for linear quantile regression, offering alternative implementation pathways for analysts. This comprehensive approach ensures that practitioners have access to versatile tools for exploring distributional relationships that simple mean regression might overlook.

## Understanding Quantile Regression: Beyond the Mean

Linear regression, the most commonly employed statistical modeling technique, focuses primarily on establishing the relationship between one or more predictor variables and the conditional mean of the response variable. When using standard Ordinary Least Squares (OLS) estimation, the goal is to find the line of best fit that minimizes the sum of squared residuals, thereby targeting the average outcome across the dataset. While effective for symmetrical distributions and homogeneous variance, this approach often fails to capture the full picture when the data exhibits high variability or heteroscedasticity.

In contrast, quantile regression allows the estimation of the relationship between predictors and various quantiles (or percentile values) of the response variable distribution. This means we are not restricted to modeling just the mean; we can estimate the 10th percentile, the median (50th percentile), or, as often relevant in risk analysis, extreme quantiles like the 90th or 95th percentile. This capability is exceptionally useful when the effect of predictors varies significantly across the distribution of the outcome.

This tutorial focuses specifically on the practical implementation of this method within the Python environment. We will walk through a robust, step-by-step example using the capabilities provided by the statsmodels library, demonstrating how to fit the model, interpret its parameters, and visualize the resulting quantile fit line relative to the underlying data points.

## Why Choose Quantile Regression?

The selection of quantile regression over traditional linear regression is often motivated by specific data characteristics or analytical goals. One primary advantage is its robustness to outliers in the

response variable, as the objective function minimizes the sum of absolute errors weighted by the quantile level, rather than the squared errors used in OLS, which heavily penalizes large errors and can distort the mean estimate significantly.

Furthermore, quantile regression provides invaluable insight into heterogeneous effects. For instance, if studying income based on education, the impact of an extra year of schooling might be small for individuals in the lowest income percentiles but substantially larger for those in the highest percentiles. Standard mean regression would only provide an average effect, potentially masking these crucial distributional differences that are vital for targeted policy design.

By modeling different parts of the conditional distribution--not just the center--analysts gain a richer and more complete understanding of how variables interact across the entire spectrum of possible outcomes. This feature is particularly important in fields like finance, epidemiology, and environmental sciences where modeling extremes, such as minimum performance thresholds or maximum observed concentrations, is often critical for decision-making and risk assessment.

## Prerequisites: Essential Python Libraries

To successfully execute advanced statistical modeling like quantile regression in Python, several fundamental libraries are required for numerical computation, data manipulation, and statistical processing. Establishing this technical foundation is a prerequisite for generating reliable and reproducible results.

The core computational backbone is provided by NumPy, which handles high-performance array operations, and Pandas, which is essential for efficient data structuring and manipulation, particularly for managing dataframes. For the statistical heavy lifting, the statsmodels library provides the dedicated implementation of the quantile regression algorithm (`quantreg`).

Finally, effective communication of the analytical results necessitates powerful visualization tools. We utilize Matplotlib's `pyplot` module to create clear scatter plots and overlay the fitted quantile lines, allowing for immediate visual interpretation of how the model results align with the structure of the raw data.

## Step 1: Loading Dependencies

The first practical step in any Python analysis workflow is loading the required packages. We import the necessary libraries under conventional aliases, which ensures that the code remains concise, readable, and easily maintainable throughout the subsequent steps of data preparation and model execution.

```
import numpy as np
```

```
import pandas as pd
import statsmodels.api as sm
import statsmodels.formula.api as smf
import matplotlib.pyplot as plt
```

We rely on [NumPy](#) for numerical operations and [Pandas](#) for handling the structured dataset. We import both the standard API (`sm`) and the formula API (`smf`) from [statsmodels](#), as the formula API simplifies model specification using familiar R-style syntax, which is particularly intuitive for regression analysis. Matplotlib is included to facilitate the creation of the final visualization.

## Step 2: Generating Synthetic Dataset

To provide a concrete illustration of [quantile regression](#), we generate a synthetic dataset. This dataset mimics the relationship between the number of hours studied and the resulting exam score for 100 students. Crucially, the data generation process is intentionally designed to produce heteroscedasticity, meaning the variance of the scores increases as the hours studied increase. This structure highlights the specific advantage of using quantile methods over standard [linear regression](#).

We use [NumPy](#)'s random functions to generate 100 observations. The study hours are uniformly distributed, and the scores are calculated with a linear trend, adding noise that scales proportionally with the study hours. By setting a random seed using `np.random.seed(0)`, we ensure that this exact dataset is reproducible every time the code is executed, guaranteeing consistency in the analysis example.

```
#make this example reproducible
```

```
np.random.seed(0)
```

```
#create dataset
```

```
obs = 100
```

```
hours = np.random.uniform(1, 10, obs)
```

```
score = 60 + 2*hours + np.random.normal(loc=0, scale=.45*hours, size=100)
```

```
df = pd.DataFrame({'hours': hours, 'score': score})
```

```
#view first five rows
```

```
df.head()
```

```
hours score
```

```
0 5.939322 68.764553
```

```

1 7.436704 77.888040
2 6.424870 74.196060
3 5.903949 67.726441
4 4.812893 72.849046

```

The resulting Pandas DataFrame contains the two variables necessary for our analysis: `hours` serving as the predictor variable, and `score` as the response variable. This data structure is now formatted correctly for input into the statsmodels quantile regression function.

### Step 3: Implementing Quantile Regression with Statsmodels

We now proceed to fit the quantile regression model. Our objective is to determine how the number of hours studied influences the conditional 90th percentile of the exam scores. Modeling this high quantile allows us to specifically analyze the relationship among high-performing students.

The core implementation relies on the `smf.quantreg()` function from statsmodels. We define the model using the formula `'score ~ hours'` and then apply the `.fit()` method, explicitly specifying the quantile level using the parameter `q=0.9`. This parameter directs the optimization algorithm to minimize the weighted absolute residuals specifically for the 90th quantile, thereby estimating the upper boundary of the expected scores.

**#fit the model**

```
model = smf.quantreg('score ~ hours', df).fit(q=0.9)
```

```
#view model summary
```

```
print(model.summary())
```

```
QuantReg Regression Results
```

```
=====
```

```
===
```

```
Dep. Variable: score Pseudo R-squared: 0.6057
```

```
Model: QuantReg Bandwidth: 3.822
```

```
Method: Least Squares Sparsity: 10.85
```

```
Date: Tue, 29 Dec 2020 No. Observations: 100
```

```
Time: 15:41:44 Df Residuals: 98
```

```
Df Model: 1
```

```
=====
```

```
===
```

```
coef std err t P>|t|
```

```
-----
```

```
Intercept 59.6104 0.748 79.702 0.000 58.126 61.095
hours 2.8495 0.128 22.303 0.000 2.596 3.103
```

```
=====
===
```

## Interpreting the Quantile Regression Results

The detailed summary table provides the estimated coefficients and statistical metrics specific to the fitted 90th quantile model. These coefficients define the impact of the hours studied solely on the location of the 90th percentile of the score distribution, offering an interpretation distinct from the mean-based coefficients produced by linear regression.

Based on the coefficient estimates shown in the output, the estimated regression equation for the 90th percentile of exam score is: **90th percentile of exam score = 59.6104 + 2.8495 \* (hours studied)**. The intercept (59.6104) is the predicted 90th percentile score when hours studied is zero, and the slope (2.8495) indicates that for every additional hour studied, the 90th percentile score is expected to increase by approximately 2.85 points.

For practical context, if a student studies for 8 hours, the expected 90th percentile of scores for students in that group would be calculated as  $59.6104 + 2.8495 * 8$ , resulting in approximately 82.39 points. The summary also includes standard errors, t-statistics, and confidence intervals, which confirm the statistical significance of the predictor variable hours in determining the upper boundary of performance.

## Step 4: Visualizing the Quantile Fit

Visualization is essential for understanding the model's performance, especially in the context of quantile regression where the fitted line does not necessarily pass through the center of the data cloud. We utilize Matplotlib to create a scatter plot of the raw data and overlay the estimated 90th quantile line, providing a clear graphical representation of the model.

First, we calculate the predicted values ( $\hat{y}$ ) using the coefficients derived from our statsmodels fit. We then plot these predicted values as a solid line alongside the scatter plot of the observed hours and scores. This process allows us to visually verify that the model correctly captures the intended percentile boundary.

```
#define figure and axis
```

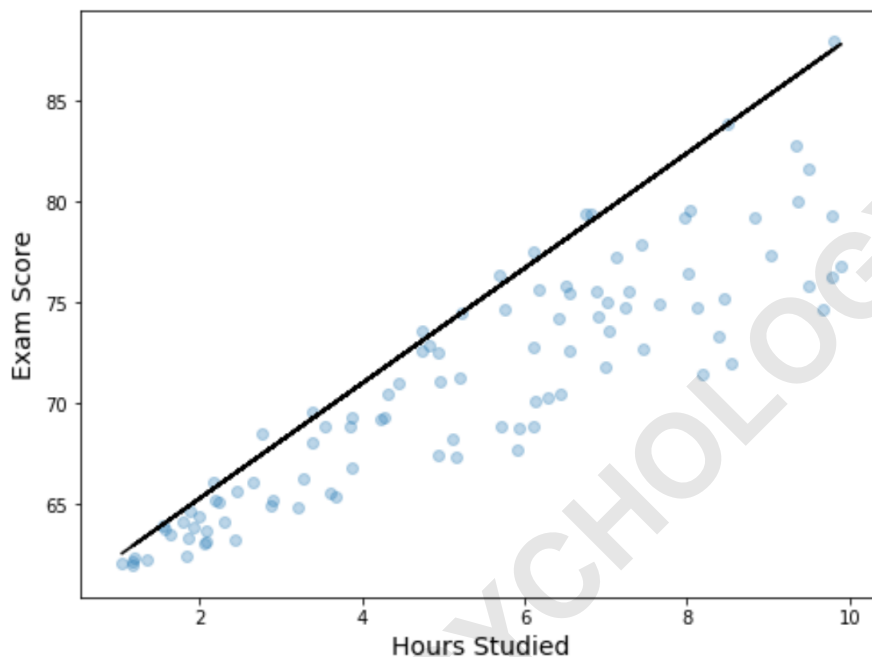
```
fig, ax = plt.subplots(figsize=(8, 6))
```

```
#get y values
```

```
get_y = lambda a, b: a + b * hours
```

```
y = get_y(model.params, model.params)

#plot data points with quantile regression equation overlaid
ax.plot(hours, y, color='black')
ax.scatter(hours, score, alpha=.3)
ax.set_xlabel('Hours Studied', fontsize=14)
ax.set_ylabel('Exam Score', fontsize=14)
```



Observe that the fitted black line does not behave like a standard linear regression line; it clearly sits above the majority of the data points. This line represents the estimated 90th percentile of scores at each level of the predictor variable, effectively modeling the upper boundary of the conditional distribution, demonstrating the robustness of quantile regression in the presence of increasing variance.

## Conclusion: Practical Applications of Quantile Regression

The ability to model distinct parts of the conditional distribution makes quantile regression an exceptionally versatile and powerful analytical tool. While our example focused on a simple educational scenario, the technique's utility spans numerous complex domains where understanding extreme outcomes is crucial.

For instance, economists use it to study how tax policy affects income distribution at the lowest and highest income brackets. Clinically, it helps model factors influencing the lowest (or highest)

recorded physiological metrics. By providing a comprehensive view beyond the average, quantile regression offers superior robustness and interpretability when data is heteroscedastic, heavily skewed, or affected by non-constant predictor effects.

Harnessing Python libraries like statsmodels and scikit-learn ensures that data scientists can efficiently implement and deploy these advanced statistical methods, moving beyond the limitations of standard mean-based modeling to achieve deeper and more nuanced data insights.

ARABPSYCHOLOGY.COM