# Perform Quantile Regression in Python

Authored by

**Mohammed looti**

November 6, 2025

The vast landscape of statistical modeling is frequently dominated by linear regression, a widely adopted and powerful technique designed to quantify the relationship between one or more predictor variables and a corresponding response variable. The conventional approach, Standard Linear Regression--typically executed using the Ordinary Least Squares (OLS) method--is fundamentally focused on estimating the conditional **mean** of the outcome distribution.

While OLS offers robustness and broad applicability, its reliance on assumptions such as homoscedasticity (constant error variance across all predictor levels) often limits its utility when dealing with complex, real-world data. When these assumptions are violated, or when the goal is to understand outcomes other than the average, OLS results can become inefficient or misleading. Many analytical scenarios demand insight into the distribution's extremities, specific thresholds, or overall variability, rather than just the central tendency.

This is precisely the domain where **quantile regression** (QR) provides a superior and highly flexible analytical framework. Unlike OLS, QR shifts the focus from the conditional mean to estimating any specific **quantile** (or percentile) of the response variable's distribution. This allows researchers to model critical thresholds such as the 10th, 50th (the conditional median), or 90th percentile independently, providing a much richer perspective on the data relationship.

This comprehensive tutorial serves as an expert guide to performing quantile regression using Python, utilizing the industry-standard **Statsmodels** library. We will detail the implementation steps, demonstrate how to interpret the resulting model coefficients, and visualize the non-central relationships captured by this advanced technique.

## Understanding the Advantages of Quantile Regression

The core utility of quantile regression stems from its enhanced flexibility and resilience compared to traditional OLS. Standard OLS models are notoriously sensitive to outliers and heavy-tailed error distributions, which can significantly skew the estimated mean relationship. Conversely, QR achieves robustness by minimizing the sum of asymmetric absolute errors, making it particularly resistant to extreme values, especially when modeling the median (the 50th quantile).

A key practical necessity for QR arises when the influence of a predictor variable is not uniform across the response distribution--a phenomenon common in fields ranging from finance to biological sciences. For example, analyzing the correlation between employee training hours and job performance might show that extra training has a dramatically stronger positive effect on employees who already rank in the top 90th percentile of performance than it does on those in the 10th percentile. QR is essential for uncovering these nuanced, varying effects.

Crucially, QR naturally handles data afflicted by **heteroscedasticity**--a condition where the variance of the residuals changes as the predictor value increases. Since QR models distinct

quantiles independently, it does not require the complex data transformations necessary to satisfy OLS assumptions under heteroscedastic conditions. By fitting separate regression lines for different quantiles, QR offers a complete, distribution-aware picture of the data generating process that a single mean estimate simply cannot provide.

## Step 1: Loading Essential Python Packages

To successfully execute our quantile regression analysis, we must first ensure that all necessary libraries are imported into the Python environment. Our workflow relies on standard tools for numerical processing and data management, alongside specialized packages for statistical modeling and visualization.

**NumPy**: The foundational library required for efficient numerical array operations and mathematical functions.
**Pandas**: Indispensable for structured data manipulation, cleaning, and handling the DataFrame objects used throughout the analysis.
**Statsmodels**: The primary library containing the dedicated `quantreg` function, which implements the core statistical modeling capabilities.
**Matplotlib**: Used specifically for plotting the resulting regression lines and visualizing the relationship between variables.

The following code block should be executed to import these packages using their conventional aliases, preparing the environment for data generation and model fitting:

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

## Step 2: Creating and Inspecting the Sample Data

To effectively demonstrate the utility of quantile regression, we will construct a synthetic dataset simulating the relationship between the number of hours studied and the resulting exam score for 100 hypothetical students. The key feature of this synthetic data is the intentional incorporation of **heteroscedastic noise**. This means the variability (or spread) of the exam scores increases as the number of hours studied increases, mirroring many real-world phenomena where outcomes are more unpredictable at higher input levels.

The data generation process sets a random seed to ensure reproducibility, creates 100

observations of 'hours studied' (drawn from a uniform distribution), and then calculates 'score' using a base linear function plus a normally distributed error term. Crucially, the standard deviation of this error term is made proportional to the 'hours studied'. This structure makes the dataset ideal for contrasting QR against standard linear regression.

The resulting **Pandas DataFrame**, which holds the 'hours' (predictor) and 'score' (response) variables, is then inspected using the `head()` method to confirm the structure and initial values:

**#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

## Step 3: Performing the Quantile Regression Analysis

The core of the analysis involves fitting the quantile regression model. Unlike OLS, which optimizes by minimizing squared errors, QR minimizes a weighted sum of absolute residuals, where the weighting is determined by the specific quantile ($tau$) of interest. This quantile is represented by the parameter `q`, which must be a value between 0 and 1.

For this demonstration, we are specifically interested in modeling the expected **90th percentile** of exam scores as a function of hours studied. This threshold represents the trajectory of high-achieving students--those who perform better than 90% of their peers for any given study time. We achieve this by setting the quantile parameter `q=0.9` within the `fit()` method of the `smf.quantreg()` function. This function allows for convenient model specification using R-like

formula syntax (`'score ~ hours'`).

The resulting summary output provides crucial statistical metrics and coefficient estimates that describe the relationship precisely at this high-achievement threshold:

**#fit the model for the 90th percentile (q=0.9)**
**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
==========
===

From the summary, we derive the estimated regression equation for the 90th percentile: `90th percentile of exam score = 59.6104 + 2.8495 * (hours studied)`. The **Intercept** (59.6104) indicates that a student who studies zero hours is expected to achieve a score of 59.61 at the 90th percentile. More importantly, the coefficient for **hours** (2.8495) reveals that for every additional hour studied, the 90th percentile score increases by approximately 2.85 points. This value is likely higher than the coefficient found using OLS, suggesting that study time provides a greater marginal return for top performers. For instance, a student studying 8 hours is expected to achieve a 90th percentile score of approximately 82.41 (59.6104 + 2.8495 * 8). This interpretation means 90% of students studying 8 hours are expected to score below 82.41.

## Step 4: Visualizing the Quantile Regression Line

Visualization is crucial for understanding how the fitted quantile regression line relates to the raw data distribution. Unlike the OLS line, which seeks to minimize the distance to the conditional mean of all points, our 90th percentile line is specifically positioned to trace the upper boundary of the data cloud. This positioning is particularly insightful given the heteroscedastic nature of our synthetic data.

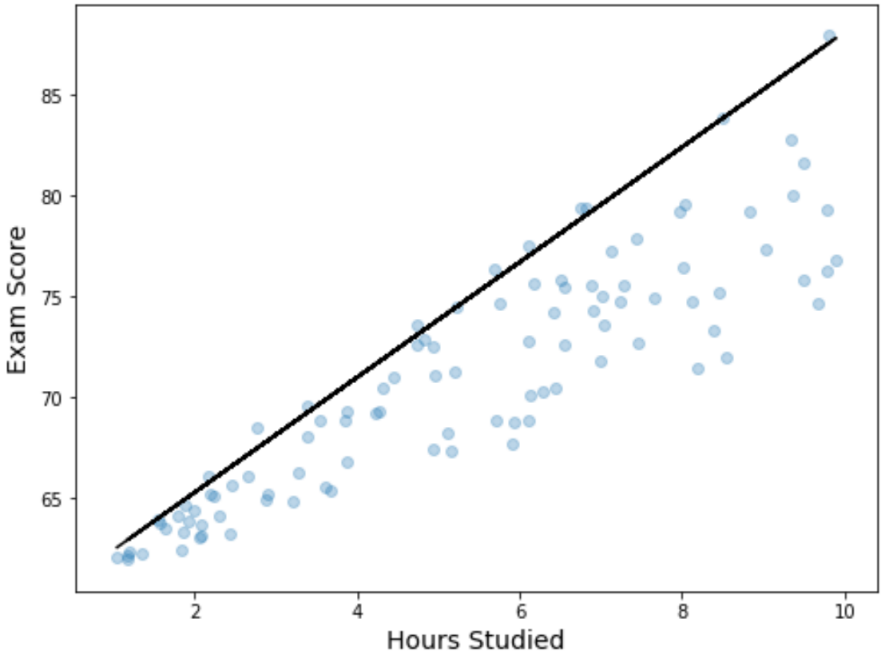
We first use the coefficients derived in Step 3 to define a function for calculating the predicted y-values, and then generate a scatter plot of the original data points overlaid with the newly calculated 90th percentile regression line. This allows for a direct visual assessment of the model's fit at the specified high quantile.

**#define figure and axis**
**fig, ax = plt.subplots(figsize=(8, 6))**

#get y values based on the fitted 90th percentile model
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)

The resulting plot visually confirms the model's accuracy:




As evident in the visualization, the fitted black line adheres closely to the upper edge of the scattered data points. Approximately 90% of the observations fall beneath this line for any given value of "Hours Studied." This visual confirmation underscores the method's ability to model non-central trends, providing valuable insight into the conditional distribution where the variance is not constant.

## Further Exploration and Practical Applications

While modeling a single high quantile (q=0.9) is useful, the full statistical power of quantile regression is unlocked by simultaneously modeling multiple quantiles. By fitting models for the 10th (`q=0.1`), 50th (`q=0.5`, the conditional median), and 90th (`q=0.9`) percentiles, a researcher can observe precisely how the effect of 'hours studied' varies across the entire distribution of exam scores.

If the coefficient for the predictor variable (hours) is significantly larger at the 90th percentile compared to the 10th percentile, this disparity highlights a **varying effect**--meaning the marginal benefit of studying is disproportionately concentrated among high-performing students. This crucial insight is fundamentally inaccessible to methods based solely on the conditional mean.

The applications of QR extend across numerous data-intensive domains where distribution extremes are more critical than the average:

**Financial Risk Management:** Employed to estimate downside risk metrics, such as Value-at-Risk (VaR), by focusing on the extreme lower quantiles of asset returns.

**Environmental and Climate Science:** Used to analyze how pollutants or temperature changes affect the lowest or highest measurements of biological indicators or crop yields.
**Healthcare and Survival Analysis:** Modeling external factors that influence patient outcomes at the extreme ends of recovery times, chronic disease progression, or treatment efficacy, allowing for targeted intervention strategies.

By moving beyond the restrictive assumptions of the conditional mean, quantile regression provides a robust, flexible, and distribution-aware framework, essential for rigorous statistical analysis in modern Python environments.