Python Quantile Regression

Introduction to Quantile Regression in Python

When you first learn about regression, ordinary least squares (OLS) is usually the starting point. OLS models the conditional mean of a target variable given predictors. But what if you care about the median, the 10th percentile, or the 90th percentile? That is precisely where **Python quantile regression** shines. Instead of focusing only on the average relationship, quantile regression lets you estimate different conditional quantiles of your response variable. This technique is especially valuable when your data contains outliers, when the effect of predictors changes across the distribution (heteroscedasticity), or when you need a more complete picture of the relationship than a single slope can provide.

Python, with its rich ecosystem of scientific libraries, makes implementing quantile regression straightforward. Libraries like `statsmodels` provide dedicated functionality through the `QuantReg` class, while `scikit-learn` offers limited support via its `QuantileRegressor` in recent versions. This article will guide you through the theory, practical implementation, interpretation, and real-world applications of quantile regression in Python.

Why Use Quantile Regression Instead of Ordinary Least Squares?

Ordinary least squares regression assumes that the mean of the response variable is a linear function of the predictors, and that the error terms are normally distributed with constant variance. However, real-world data often violates these assumptions. **Quantile regression** relaxes many of these constraints and offers several key advantages:

- **Robustness to outliers:** Because quantile regression minimizes the sum of absolute residuals (for median regression) or weighted absolute residuals for other quantiles, it is far less sensitive to extreme values than OLS, which minimizes squared residuals.
- **Heteroscedasticity handling:** When the variability of the response changes with the predictors, quantile regression can model how different parts of the distribution respond differently.
- **Complete distributional view:** You can estimate multiple quantiles (e.g., 0.1, 0.5, 0.9) to understand the full conditional

distribution, not just the mean.
- **No assumption of normality:** Quantile regression is a non-parametric technique; it does not require the error distribution to follow any particular shape.

For example, in economics, the effect of education on income might be stronger at the top of the income distribution than at the bottom. Quantile regression captures this disparity elegantly. In biomedical studies, researchers often want to estimate growth charts (e.g., the 5th percentile of height for a given age) – a natural application of quantile regression.

Core Concepts: Conditional Quantiles and Loss Functions

In standard linear regression, we model the conditional expectation: $E(Y|X) = X\beta$. In quantile regression, we model the **conditional τ-th quantile**: $Q_\tau(Y|X) = X\beta(\tau)$, where τ is a quantile between 0 and 1. For example, $\tau = 0.5$ corresponds to the conditional median.

The coefficients $\beta(\tau)$ are estimated by minimizing the following loss function:

$$min \sum \rho_\tau(y_i - x_i^T\beta)$$

where $\rho_\tau(u) = u \cdot (\tau - I(u < 0))$. This is known as the **pinball loss** or tilted absolute error. When $\tau = 0.5$, the loss becomes $\sum |y_i - x_i^T\beta|$, which is the sum of absolute deviations – equivalent to median regression. For other quantiles, the loss is asymmetric, giving different weights to positive and negative errors.

The pinball loss is convex, which ensures that a unique solution exists (though no closed-form exists, so numerical optimization is used). Python's `statsmodels` handles this via linear programming (interior point methods).

Setting Up Your Python Environment

Before diving into code, ensure you have the necessary libraries. The primary package for quantile regression is `statsmodels`. You can install it along with other helpful libraries using pip:

```
pip install statsmodels pandas numpy matplotlib seaborn
```

Optional but recommended: `scikit-learn` (version 1.0+) includes a `QuantileRegressor`, but `statsmodels` is more mature and flexible for statistical inference (confidence intervals, hypothesis testing). We will focus on `statsmodels` in this article.

Implementing Quantile Regression in Python with statsmodels

Step 1: Load and Explore the Data

We will use a classic dataset from `statsmodels`: the *Engel* dataset, which contains income and food expenditure data for Belgian households. This dataset is well-known for exhibiting heteroscedasticity – the variability of food expenditure increases with income.

```
import statsmodels.api as sm
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load the Engel dataset
data = sm.datasets.engel.load()
df = pd.DataFrame(data.data, columns=['income', 'foodexp'])
print(df.head())
```

A quick scatter plot reveals the relationship and the increasing spread:

```
sns.scatterplot(data=df, x='income', y='foodexp')
plt.title('Food Expenditure vs. Income')
plt.show()
```

Step 2: Fit a Quantile Regression Model for the Median

We start by estimating the conditional median ($\tau=0.5$). This is often the first quantile to examine because it provides a robust alternative to the mean.

```
import statsmodels.formula.api as smf

model_median = smf.quantreg('foodexp ~ income', df)
result_median = model_median.fit(q=0.5)
print(result_median.summary())
```

The summary output includes coefficients, standard errors (computed via bootstrap), pseudo-R-squared, and quantile regression diagnostics. The intercept and slope for income at the median are both statistically significant.

Step 3: Fit Multiple Quantiles to Understand the Full Distribution

The real power of **Python quantile regression** comes from fitting many quantiles simultaneously. Let's estimate the 0.1, 0.25, 0.5, 0.75, and 0.9 quantiles.

```
quantiles = [0.1, 0.25, 0.5, 0.75, 0.9]
models = {}
results = {}

for q in quantiles:
    models[q] = smf.quantreg('foodexp ~ income', df)
    results[q] = models[q].fit(q=q)

# Store coefficients in a DataFrame
coef_df = pd.DataFrame({
    'quantile': quantiles,
    'intercept': [results[q].params['Intercept'] for q in quantiles],
    'income_coef': [results[q].params['income'] for q in quantiles]
})
print(coef_df)
```

Notice how the income coefficient increases at higher quantiles. For low-income households ($\tau=0.1$), a unit increase in income leads to

a smaller increase in food expenditure compared to high-income households (τ=0.9). This is a classic sign of **heteroscedasticity** and varying marginal effects across the distribution.

Step 4: Visualizing the Fitted Quantile Lines

A plot of the estimated quantile lines overlaid on the scatter plot provides an intuitive understanding.

```
# Generate x values for prediction
x_pred = pd.DataFrame({'income': np.linspace(df['income'].min(), df['income'].max(), 100)})
y_pred = pd.DataFrame({
    '0.1': results[0.1].predict(x_pred),
    '0.25': results[0.25].predict(x_pred),
    '0.5': results[0.5].predict(x_pred),
    '0.75': results[0.75].predict(x_pred),
    '0.9': results[0.9].predict(x_pred)
})

plt.figure(figsize=(10,6))
sns.scatterplot(data=df, x='income', y='foodexp', alpha=0.6)
for col in y_pred.columns:
    plt.plot(x_pred['income'], y_pred[col], label=f'τ={col}')
plt.legend()
plt.title('Quantile Regression Lines for Food Expenditure')
plt.xlabel('Income')
plt.ylabel('Food Expenditure')
plt.show()
```

The lines fan out as income increases, clearly showing that the effect of income on food expenditure is larger at the top of the distribution. An OLS regression line would lie somewhere near the median but would not capture this widening dispersion.

Interpreting Quantile Regression Results

Interpretation of coefficients in quantile regression is similar to OLS, but with a key difference: the coefficient $\beta(\tau)$ measures the change in the conditional τ-th quantile of Y for a one-unit change in X, holding other variables constant. For example, in the Engel dataset, the income coefficient at $\tau=0.9$ is approximately 0.694, meaning that a one-unit increase in income is associated with a 0.694 increase in the 90th percentile of food expenditure.

When you compare coefficients across quantiles, you can test whether the effect is constant. A simple way is to look at the overlapping confidence intervals. More formally, `statsmodels` provides a method to compute **variance-covariance matrices** via bootstrap, allowing hypothesis tests such as the equality of slopes across quantiles.

Computing Confidence Intervals and Hypothesis Testing

Quantile regression standard errors can be computed analytically (assuming i.i.d. errors) but are more commonly obtained via **bootstrap** because it is robust to heteroscedasticity. In `statsmodels`, you can use the `fit(q=..., cov_type='iid')` or `'robust'` options. For bootstrap, you can manually resample.

```
# Bootstrap example for median
def bootstrap_quantile(data, q=0.5, n_boot=1000):
    coefs = []
    n = len(data)
    for _ in range(n_boot):
        sample = data.sample(n=n, replace=True)
        model = smf.quantreg('foodexp ~ income', sample)
        res = model.fit(q=q)
        coefs.append(res.params)
    coef_df_boot = pd.DataFrame(coefs)
    ci = coef_df_boot.quantile([0.025, 0.975])
    return ci
```

```
ci_median = bootstrap_quantile(df, q=0.5)
print(ci_median)
```

This gives a 95% confidence interval for the intercept and slope based on the percentile bootstrap.

Comparing Quantile Regression with OLS

It is illuminating to overlay the OLS regression line on the same plot. OLS estimates the mean conditional on X, which, under heteroscedasticity, may be a poor representation of the data.

```
ols_model = smf.ols('foodexp ~ income', df)
ols_result = ols_model.fit()
ols_pred = ols_result.predict(x_pred)

plt.figure(figsize=(10,6))
sns.scatterplot(data=df, x='income', y='foodexp', alpha=0.4)
for col in y_pred.columns:
    plt.plot(x_pred['income'], y_pred[col], label=f'Quantile τ={col}', linestyle='--')
plt.plot(x_pred['income'], ols_pred, label='OLS mean', color='black', linewidth=2)
plt.legend()
plt.title('OLS vs. Quantile Regression')
plt.show()
```

You will see that the OLS line cuts through the middle but provides no information about the spread. Quantile regression, in contrast, reveals that the relationship is not uniform across the distribution.

Handling Large Datasets and Optimization

For very large datasets, the computational cost of quantile regression (which requires solving a linear programming problem) can be higher than OLS. However, `statsmodels` is reasonably efficient for datasets with tens of thousands of observations. If you need

speed, consider using `scikit-learn's`