Learning Quadratic Regression with Python: A Comprehensive Guide

Authored by

Mohammed looti

November 7, 2025

The Fundamentals of Quadratic Regression

Quadratic regression represents a powerful and specialized technique within the realm of polynomial regression. It is primarily employed in statistical analysis when the relationship between a single predictor variable (often denoted as X) and a corresponding response variable (the outcome Y) is distinctly non-linear and exhibits a parabolic curve. This mathematical model is defined by a second-degree polynomial equation: $$ Y = beta_0 + beta_1 X + beta_2 X^2 + epsilon $$ where $beta_0$, $beta_1$, and $beta_2$ are the coefficients determined through the regression process, and $epsilon$ represents the error term.

Unlike simple linear regression, which mandates that data points follow a straight line, quadratic models are essential for capturing curvature. This curvature typically manifests as a "U" shape or, conversely, an inverted "U" shape when the data is plotted on a coordinate plane. The need for this technique arises specifically when the rate of change in the response variable is not constant. Instead, the effect of the predictor variable increases up to an inflection point and then begins to decrease (or vice-versa).

Failing to account for this inherent curvature by forcing a linear model onto parabolic data results in significant statistical shortcomings, including biased coefficient estimates and large residuals, leading to poor predictive accuracy. Consequently, researchers rely on quadratic regression whenever their data suggests an optimal level or a point of diminishing returns. This methodology allows for the accurate identification of the vertex of the parabola, which corresponds to the maximizing or minimizing value of the response variable relative to the predictor.

This comprehensive tutorial focuses on leveraging the robust data science toolkit available in Python, specifically utilizing the specialized capabilities of the **NumPy** library, to efficiently perform the necessary calculations for fitting this critical polynomial curve.

A Practical Example: Optimizing Happiness vs. Hours Worked

To solidify the conceptual understanding of quadratic regression, we will analyze a highly relatable, albeit hypothetical, dataset. This example investigates the relationship between the number of hours an individual devotes to work each week and their self-reported happiness level. The response variable, happiness, is quantified on a standardized scale ranging from 0 (minimum happiness) to 100 (maximum happiness). Our dataset comprises observations from 16 distinct subjects.

The core hypothesis driving this analysis is based on established psychological principles: extreme conditions often lead to undesirable outcomes. We posit that working very few hours (e.g., due to unemployment or financial instability) might correlate with lower happiness levels. Conversely, working an excessively long schedule (resulting in burnout, stress, and lack of personal time) will

also lead to a decline in reported happiness.

This scenario inherently implies the existence of an optimal workload--a "sweet spot" of hours per week that maximizes happiness before the negative effects of overwork set in. This theoretical relationship perfectly maps onto an inverted "U" shape, confirming that **quadratic regression** is the most suitable statistical technique to accurately model and identify this crucial optimal point. Our goal is to derive a predictive model that quantifies this complex, non-linear trade-off.

Preparing the Python Environment and Data Arrays

The initial stage of any statistical analysis in Python requires the proper setup of the environment, primarily by importing essential libraries. We rely heavily on NumPy, the fundamental package for numerical computing in Python, which provides powerful array objects and sophisticated mathematical functions necessary for statistical calculations. We also import `scipy.stats`, although NumPy's core functionalities will handle the primary fitting process.

The two critical data structures, defined below as `hours` and `happ`, represent our observed data points. The `hours` array serves as our predictor variable (X), and `happ` serves as our response variable (Y). It is imperative to structure this data into NumPy arrays or standard Python lists before proceeding with the fitting process.

import numpy as np
import scipy.stats as stats

#define variables for hours worked and happiness scores
hours =
happ =

This step ensures the data is correctly structured for immediate statistical processing. While the subsequent steps will automate the mathematical fitting, a fundamental best practice in data analysis is to always visually inspect the raw data first. This preliminary visual check, often via a scatterplot, is essential for confirming that the hypothesized curvilinear relationship is actually present in the sample data before committing to the quadratic model fitting.

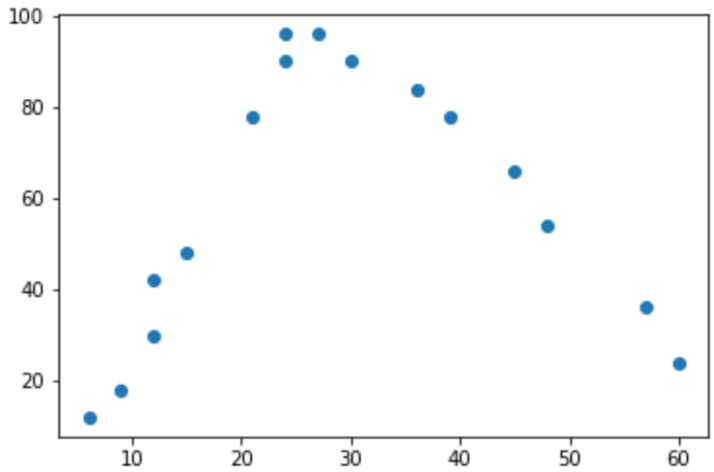
Visual Confirmation: Plotting the Curvilinear Relationship

Before calculating the regression coefficients, visualizing the dataset is the most critical step for validating the choice of model. We must confirm that the relationship between hours worked and happiness levels indeed follows a parabolic trajectory, justifying the use of quadratic over linear methods. This visualization is achieved using the Matplotlib library, Python's standard library for generating static, interactive, and animated visualizations.

The scatterplot serves as a powerful diagnostic tool. By plotting the raw data points, we can immediately observe the trend. If the points form a distinct curve--specifically an inverted "U" in our case--it provides strong empirical support for proceeding with a second-degree polynomial fit. If the relationship appeared linear or exponential, a different regression technique would be necessary.

import matplotlib.pyplot as plt

#create scatterplot of raw data
plt.scatter(hours, happ)



The generated scatterplot provides clear visual evidence supporting our hypothesis. Happiness scores initially rise sharply as the number of hours worked increases, peaking around 30 to 40 hours per week. Critically, past this threshold, the trend reverses, and further increases in working hours lead to a noticeable decline in reported happiness. This classic inverted "U" pattern confirms that the relationship is non-linear and that **quadratic regression** is the statistically correct approach for accurately modeling this data.

Executing the Quadratic Fit using NumPy's polyfit

The core of performing quadratic regression in Python lies within the `numpy.polyfit()` function. This highly versatile function is designed to find the coefficients of a best-fit polynomial of a specified degree for a given set of X and Y coordinates. Because quadratic regression corresponds to a second-degree polynomial, we must explicitly set the degree parameter to `2`.

The output of `numpy.polyfit(x, y, 2)` is an array containing the three required coefficients ($beta_2$, $beta_1$, $beta_0$) that define the parabola. These raw coefficients are then typically passed to `np.poly1d()`. The `np.poly1d()` function is extremely useful as it encapsulates these

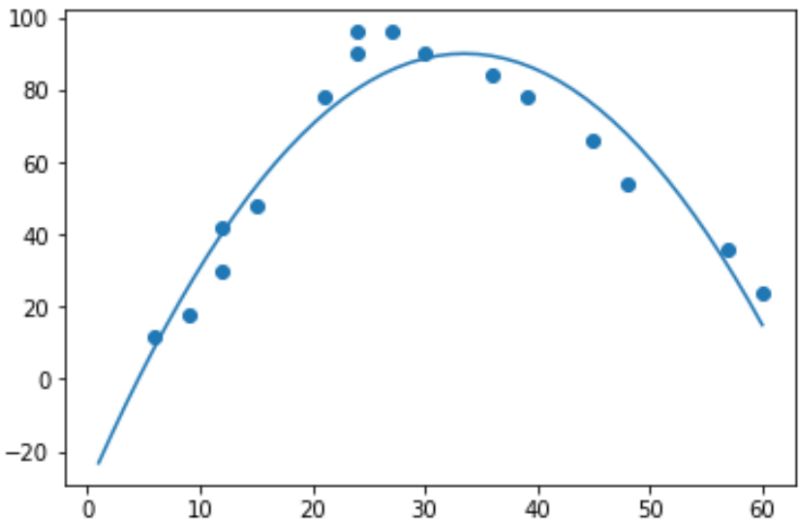
coefficients into a one-dimensional polynomial object that behaves like a callable function. This makes subsequent tasks, such as calculating predicted Y values for new X inputs or plotting the smooth curve, exceptionally straightforward.

The following code snippet demonstrates the fitting process, calculates the coefficients, and then generates a sequence of evenly spaced points (`polyline`) across the observed range of hours. Finally, it plots the calculated quadratic regression curve, overlaid precisely on the original scatterplot, offering an immediate visual assessment of the model's fit quality.

```
import numpy as np

#polynomial fit with degree = 2, yielding coefficients
model = np.poly1d(np.polyfit(hours, happ, 2))

#add fitted polynomial line to scatterplot for visualization
polyline = np.linspace(1, 60, 50) # creating 50 evenly spaced points for a smooth curve
plt.scatter(hours, happ)
plt.plot(polyline, model(polyline))
plt.show()
```



The resulting graph clearly shows the fitted curve passing closely through the majority of the data points. The smooth line accurately captures the initial rise and subsequent decline in happiness, validating the success of the quadratic model fitting process in Python.

Analyzing the Regression Equation and Making Predictions

Once the fitting process is complete, the polynomial object created by `np.poly1d()` provides immediate access to the derived regression equation. By simply printing the `model` object, we retrieve the standard algebraic expression, with the coefficients calculated using the principle of least squares minimization.

print(model)

-0.107x2 + 7.173x - 30.25

The resulting fitted quadratic regression equation, using conventional notation, is:

$$ text{Happiness} = -0.107 (text{Hours}^2) + 7.173 (text{Hours}) - 30.25 $$

Interpreting these coefficients provides deep insight into the relationship. The negative coefficient for the X^2 term (-0.107) is crucial; it confirms the inverted parabolic shape, meaning there is indeed a maximum point where happiness peaks before declining. The constant term (-30.25) represents the theoretical happiness level when hours worked are zero, although this value may not be practically meaningful outside the observed data range.

This mathematical equation is now a powerful predictive tool. We can use it to estimate the expected happiness score for any arbitrary number of hours worked within the range of our data. For instance, if we wish to predict the expected happiness level for an individual working exactly 30 hours per week, we substitute $X = 30$ into the derived equation:

$$ text{Happiness} = -0.107(30)^2 + 7.173(30) - 30.25 $$
$$ text{Happiness} = -0.107(900) + 215.19 - 30.25 $$
$$ text{Happiness} = -96.3 + 215.19 - 30.25 = textbf{88.64} $$

Based on our fitted model, the predicted expected happiness score for an individual working 30 hours per week is calculated to be **88.64**. This predictive capability is one of the primary benefits of successfully fitting a regression model.

Assessing Model Goodness-of-Fit: The R-squared Metric

After fitting the model and interpreting its coefficients, the final essential step is evaluating its performance. The most common metric for assessing the goodness-of-fit in regression analysis is the R-squared value, or the Coefficient of Determination. R-squared quantifies the proportion of the total variance in the response variable (Y) that is statistically predictable from the predictor variable (X).

A value of R-squared ranges from 0 to 1, where a value closer to 1 signifies that the model explains a very high percentage of the variability in the data, indicating an excellent fit. Since the

standard NumPy fitting functions do not directly output the R-squared metric, we must calculate it manually, which provides a deeper understanding of the underlying statistical principles. The R-squared is calculated as the ratio of the sum of squares of the regression (SSreg) to the total sum of squares (SStot).

The function below systematically defines the steps necessary to compute the R-squared: calculating the predicted values ($hat{Y}$), determining the mean of the observed values ($bar{Y}$), and then calculating the necessary sums of squares.

```
#define function to calculate r-squared for any polynomial degree
def polyfit(x, y, degree):
results = {}
coeffs = np.polyfit(x, y, degree)
p = np.poly1d(coeffs)
#calculate r-squared
yhat = p(x) # predicted Y values
ybar = np.sum(y)/len(y) # mean of observed Y
ssreg = np.sum((yhat-ybar)**2) # sum of squares regression (explained variance)
sstot = np.sum((y - ybar)**2) # total sum of squares (total variance)
results = ssreg / sstot
```

return results

#find r-squared of polynomial model with degree = 2
polyfit(hours, happ, 2)

{'r_squared': 0.9092114182131691}

The calculated R-squared value for our quadratic model is approximately **0.9092**. This high figure is highly encouraging, signifying that 90.92% of the variation observed in the reported happiness levels is successfully explained and accounted for by the quadratic relationship with the hours worked per week. This confirms that the derived quadratic model provides an outstanding fit for this specific non-linear dataset, offering reliable predictions and interpretation of the optimal working hours.

Further Resources for Advanced Regression Analysis

For readers interested in expanding their knowledge of modeling complex relationships beyond the quadratic form, or comparing methods across different software environments, the following resources are recommended:

How to Perform Polynomial Regression in Python
How to Perform Quadratic Regression in R
How to Perform Quadratic Regression in Excel