Polynomial Regression in R (Step-by-Step)

Authored by

Mohammed looti

November 9, 2025

When analyzing relationships between variables in statistics, we often rely on linear models. However, real-world data frequently exhibits curvature, necessitating the use of more flexible techniques. Polynomial regression is a powerful extension of standard multiple linear regression designed specifically for modeling these nonlinear relationships. It allows us to capture complex curves by adding polynomial terms (powers) of the predictor variable to the model equation.

This method is particularly valuable when we observe that the effect of a predictor variable on the response variable changes direction or magnitude as the predictor increases. While it maintains the characteristics of a linear regression model in terms of the coefficients (since the relationship is linear in the parameters, not the variables), it produces a curve in the relationship between the predictor and the response.

The generalized form of a polynomial regression model of degree *h* is mathematically expressed as:

$$Y = \beta 0 + \beta 1X + \beta 2X2 + ... + \beta hXh + \varepsilon$$

In this tutorial, we provide a detailed, step-by-step guide on how to implement and evaluate polynomial regression using the statistical computing environment, R. We will focus on the crucial process of selecting the optimal polynomial degree using cross-validation techniques.

Step 1: Generating the Synthetic Dataset in R

To effectively demonstrate the application of polynomial regression, we will first create a synthetic dataset. This approach allows us to control the underlying relationship between the predictor variable (hours studied) and the response variable (final exam score), ensuring that a known nonlinear pattern exists. Our dataset will simulate the performance of 50 students, where the relationship between study time and score is expected to follow a curve, suggesting that increasing study hours yields diminishing returns, or perhaps an accelerated rate of improvement only after a certain threshold.

We use the `set.seed()` function to ensure the **reproducibility** of our results, which is standard practice in statistical programming. The data frame `df` is constructed using the `runif()` function in R to randomly generate study hours between 5 and 15. Crucially, the final scores are calculated using a formula that includes a cubic term (`df$hours^3/150`) plus some random noise, thereby embedding a true nonlinear relationship that the regression model must discover.

Executing the following code block creates the necessary variables and displays the initial structure of our simulated data:

```
#make this example reproducible
set.seed(1)
```

```
#create dataset
df <- data.frame(hours = runif(50, 5, 15), score=50)
df$score = df$score + df$hours^3/150 + df$hours*runif(50, 1, 2)

#view first six rows of data
head(data)
```

```
hours score
1 7.655087 64.30191
2 8.721239 70.65430
3 10.728534 73.66114
4 14.082078 86.14630
5 7.016819 59.81595
6 13.983897 83.60510
```

Step 2: Visualizing the Data Relationship (Exploratory Analysis)

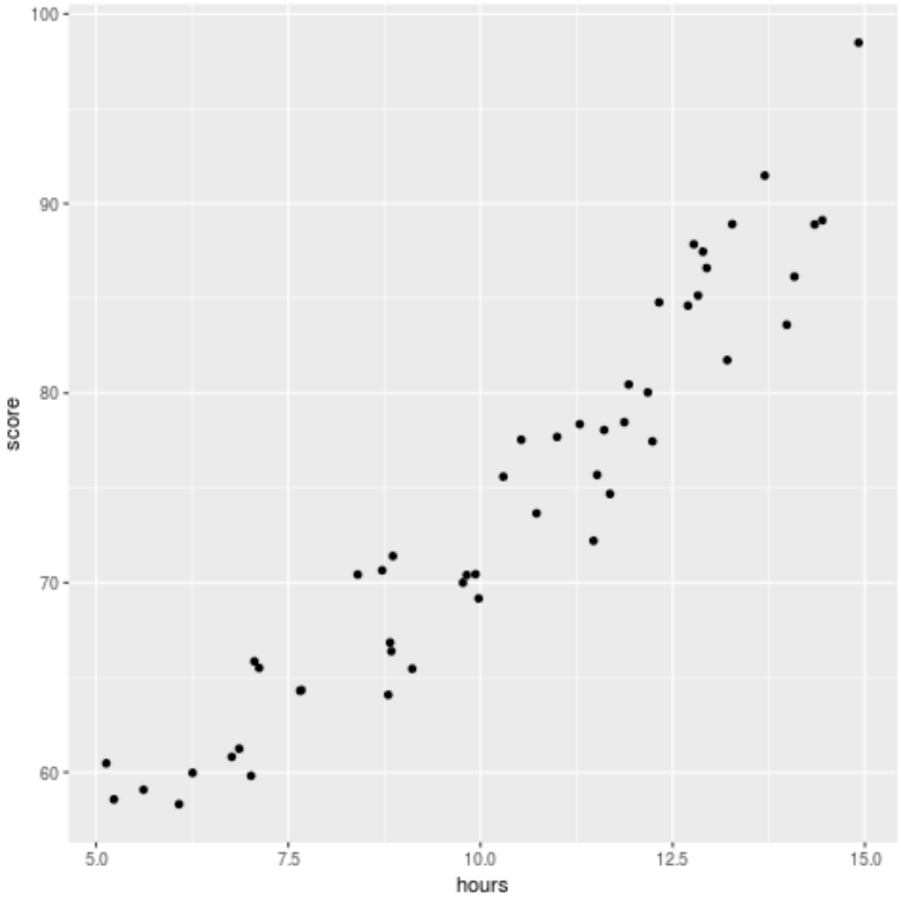
Before proceeding with model fitting, a fundamental step in any statistical analysis is performing exploratory data analysis (EDA). Visualization provides immediate insight into the nature of the relationship between the predictor variable (hours studied) and the response variable (score). If the data points roughly align along a straight line, simple linear regression would suffice. However, if a discernible curve is present, a polynomial model is likely necessary.

We utilize the powerful `ggplot2` package in R to generate a scatterplot. This plot visually confirms the hypothesis derived from our data generation process. By mapping `hours` to the x-axis and `score` to the y-axis, we can assess the trend and determine the presence of nonlinearity.

The resulting scatterplot clearly demonstrates a nonlinear trend. Specifically, the data points exhibit a noticeable upward curve, which is not adequately captured by a straight line. This characteristic curvature strongly suggests that a model incorporating a **quadratic term** (degree 2) or potentially higher-order terms would provide a significantly better fit than a standard linear model. This visual confirmation is crucial for justifying the subsequent use of polynomial regression methods.

library(ggplot2)

```
ggplot(df, aes(x=hours, y=score)) +
geom_point()
```



Step 3: Model Selection via K-Fold Cross-Validation

A critical challenge in polynomial regression is determining the optimal degree (*h*) for the polynomial. Selecting too low a degree (underfitting) fails to capture the complexity of the data, while selecting too high a degree (overfitting) captures noise and reduces the model's ability to generalize to new, unseen data. To rigorously select the best model, we employ k-fold cross-validation.

In this process, we fit five distinct polynomial models, ranging from degree *h* = 1 (simple linear regression) up to degree *h* = 5. We use **k-fold cross-validation** with K=10 folds. The dataset is randomly partitioned into 10 equal subsets. The model is trained on 9 folds (training data) and tested on the remaining fold (testing data). This rotation is repeated 10 times, ensuring every data point serves exactly once as part of the test set. The primary metric used for comparison is the Mean Squared Error (MSE), which quantifies the average squared difference between the predicted and actual response values on the test set. A lower test MSE indicates superior predictive performance and better generalization.

The R code below automates this selection process. It shuffles the data, defines the folds, and

iterates through each fold (outer loop) and each polynomial degree (inner loop). Inside the loops, it fits the model using `lm()` with the `poly()` function, predicts scores on the held-out test data, and calculates the MSE for that specific fold and degree. Finally, it averages the MSE across all 10 folds for each degree, giving us a robust estimate of the true prediction error.

```
#randomly shuffle data
df.shuffled <- df

#define number of folds to use for k-fold cross-validation
K <- 10

#define degree of polynomials to fit
degree <- 5

#create k equal-sized folds
folds <- cut(seq(1,nrow(df.shuffled)),breaks=K,labels=FALSE)

#create object to hold MSE's of models
mse = matrix(data=NA,nrow=K,ncol=degree)

#Perform K-fold cross validation
for(i in 1:K){

#define training and testing data
testIndexes <- which(folds==i,arr.ind=TRUE)
testData <- df.shuffled
trainData <- df.shuffled

#use k-fold cv to evaluate models
for (j in 1:degree){
fit.train = lm(score ~ poly(hours,j), data=trainData)
fit.test = predict(fit.train, newdata=testData)
mse = mean((fit.test-testData$score)^2)
}
}

#find MSE for each degree
colMeans(mse)

9.802397 8.748666 9.601865 10.592569 13.545547
```

Reviewing the results of the cross-validation process, we obtain the following average test MSE

values for each model degree:

Test MSE with degree h = 1 (Linear Model): **9.80**
Test MSE with degree h = 2 (Quadratic Model): **8.75**
Test MSE with degree h = 3 (Cubic Model): **9.60**
Test MSE with degree h = 4: **10.59**
Test MSE with degree h = 5: **13.55**

Based on the principle of minimizing prediction error, the model with the lowest average test Mean Squared Error is the polynomial regression model of degree *h* = 2 (the quadratic model). This outcome aligns perfectly with the initial visual assessment performed in Step 2, confirming that a quadratic relationship best describes the connection between study hours and exam scores without introducing unnecessary complexity or overfitting the data.

Step 4: Analyzing and Interpreting the Optimal Model Coefficients

Having identified the quadratic model (degree 2) as the best fit based on k-fold cross-validation, the next logical step is to finalize this model using the entire dataset and analyze its coefficients. This allows us to derive the explicit mathematical equation that can be used for prediction and inference. When fitting the final model, we must use the `raw=T` argument within the `poly()` function in R. This ensures that the polynomial terms are returned as standard powers of the predictor (X, X^2) rather than orthogonal polynomials, which simplifies the interpretation of the resulting coefficients significantly.

We fit the final model using `lm()` and then inspect the detailed summary output. The summary provides vital statistical information, including the estimated coefficients (β values), their standard errors, T-statistics, and associated P-values. These metrics help us determine which components of the model are statistically significant contributors to predicting the response variable.

Upon reviewing the coefficients section of the summary, we can construct the final fitted equation. For our best-performing quadratic model, the equation is determined by the estimates for the intercept (β?), the linear term (β?), and the quadratic term (β?):

#fit best model
best = lm(score ~ poly(hours,2, raw=T), data=df)

#view summary of best model
summary(best)

Call:
lm(formula = score ~ poly(hours, 2, raw = T), data = df)

Residuals:
Min 1Q Median 3Q Max
-5.6589 -2.0770 -0.4599 2.5923 4.5122

Coefficients:
Estimate Std. Error t value Pr(>|t|)
(Intercept) 54.00526 5.52855 9.768 6.78e-13 ***
poly(hours, 2, raw = T)1 -0.07904 1.15413 -0.068 0.94569
poly(hours, 2, raw = T)2 0.18596 0.05724 3.249 0.00214 **

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

From the output, we extract the coefficient estimates to define the model explicitly. The final fitted polynomial regression model equation is:

Score = 54.00526 - 0.07904*(hours) + 0.18596*(hours)2

This equation is now the predictive tool derived from our analysis. We can use it to estimate the expected exam score for any student, given their study hours. For instance, if a student studies for **10 hours**, the estimated score is calculated as follows:

Score = 54.00526 - 0.07904*(10) + 0.18596*(10)2 = 54.00526 - 0.7904 + 18.596 ≈ **71.81**. This demonstrates the practical application of the derived coefficients in making actionable predictions.

Step 5: Visualizing the Final Fitted Regression Curve

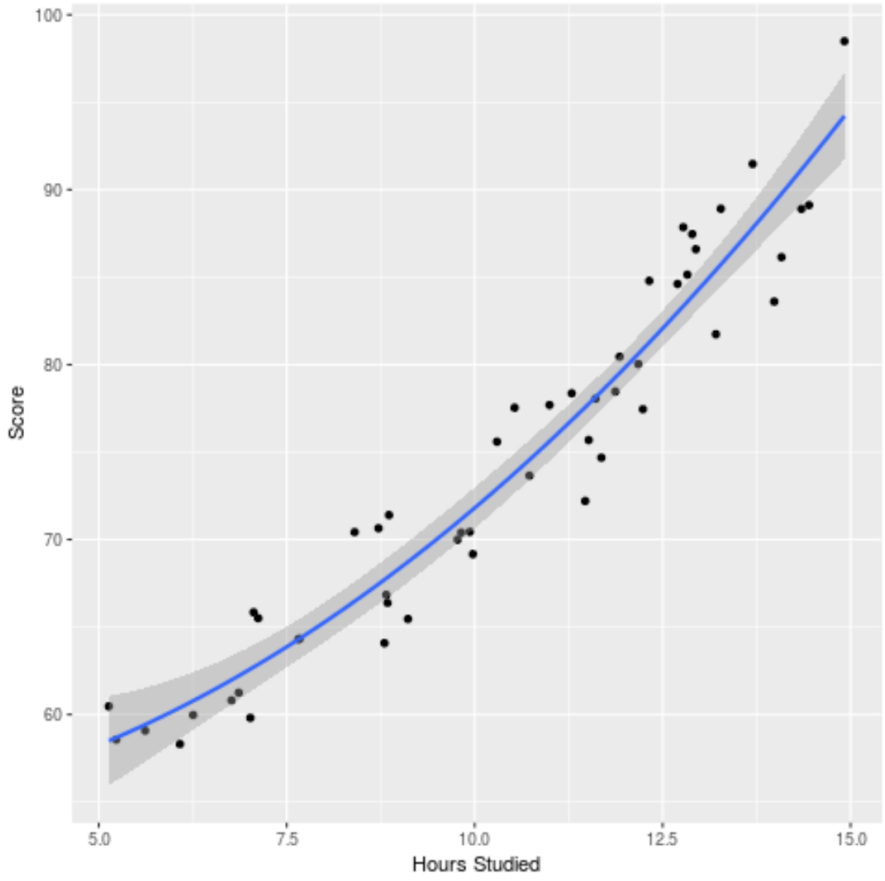
Although the statistical output confirms that the quadratic model is the optimal choice, the most intuitive way to validate its performance is by plotting the fitted curve directly onto the original scatterplot. This visualization allows us to confirm how closely the derived polynomial curve traces the path of the observed data points. If the model is robust, the curve should smoothly pass through the central tendency of the data, minimizing the vertical distance (residuals) to the majority of the observations.

We return to the `ggplot2` framework to generate this final visualization. By adding the `stat_smooth()` function, we instruct R to overlay a regression line. Crucially, we specify `method='lm'` and use the formula `y ~ poly(x, 2)` within the `stat_smooth` layer. This forces `ggplot2` to calculate and display the best-fit quadratic curve, exactly matching the model we selected in the previous steps.

The resulting plot clearly illustrates the superior fit of the quadratic model compared to what a straight line (degree 1) could achieve. The curve naturally follows the upward bending pattern of

the data, reinforcing the conclusion drawn from the MSE analysis that the degree 2 polynomial provides the best balance between complexity and predictive accuracy for this specific dataset.

```
ggplot(df, aes(x=hours, y=score)) +
geom_point() +
stat_smooth(method='lm', formula = y ~ poly(x,2), size = 1) +
xlab('Hours Studied') +
ylab('Score')
```



Conclusion and Further Resources

This tutorial demonstrated the complete workflow for applying polynomial regression in R, from data generation and initial visualization through rigorous model selection using k-fold cross-validation, and culminating in the interpretation of the final predictive model. The key takeaway is the importance of validation techniques like cross-validation to prevent overfitting when dealing with higher-order polynomial terms. By using cross-validation, we ensure the chosen model generalizes well beyond the training data.

For those interested in exploring the underlying code further or adapting it for different datasets, the complete R script used for this analysis is available at the provided external repository.

You can find the complete R code used in this example here.