# Learning Polynomial Regression in R with stat_poly_eq()

Authored by

**Mohammed looti**

November 13, 2025

RECOMMENDED CITATION

Mohammed looti (2025). *Learning Polynomial Regression in R with stat_poly_eq()*. PSYCHOLOGICAL STATISTICS. Retrieved from https://statistics.arabpsychology.com/?p=24081

## Understanding Polynomial Regression

When analyzing datasets, we often find that the relationship between a predictor variable and a response variable is not strictly linear. In such cases, standard linear regression models fail to capture the underlying structure accurately. This is where Polynomial regression becomes an essential statistical technique. It allows us to model a nonlinear relationship by fitting an *n*th degree polynomial to the data.

The core concept involves introducing powers of the predictor variable into the regression equation. For instance, a polynomial regression model takes the general form shown below:

$$Y = \beta0 + \beta1X + \beta2X2 + ... + \betahXh + \varepsilon$$

Here, *h* represents the **degree** of the polynomial. Choosing the correct degree is critical; too low a degree (underfitting) misses the curvature, while too high a degree (overfitting) can model the noise rather than the signal. Once the appropriate model is chosen and fitted, data scientists frequently need to visualize the results, plotting the fitted curve alongside a crucial metric, the **R-squared value**, to assess the goodness of fit directly on the graph.

## Introducing stat_poly_eq() and the ggpmisc Package

While the R environment provides robust tools for statistical modeling and visualization, combining statistical output directly onto graphical representations sometimes requires manual steps or complex code. The primary goal for many researchers is to fit a polynomial regression model, plot the resulting curve, and display the model's summary statistics--specifically the equation and the R-squared value--on the same plot for immediate interpretation.

The most efficient method to achieve this level of integrated visualization is by utilizing the **stat_poly_eq()** function. This function is an integral part of the ggpmisc package, an extension designed to work seamlessly with **ggplot2**. The **ggpmisc** package specializes in automating the display of equations and statistics for various types of regression lines within the ggplot framework, greatly simplifying the workflow for data presentation.

The utility of **stat_poly_eq()** lies in its ability to calculate the regression equation and the associated R-squared value dynamically and render them as text annotations on the plot itself. This eliminates the need for manual text placement or external calculations, ensuring that the displayed statistics are always consistent with the regression line being plotted. We will now proceed through a detailed example demonstrating how to implement this function effectively in practice.

## Step-by-Step Example: Preparing the Dataset in R

To illustrate the application of **stat_poly_eq()**, we will first generate a synthetic dataset. This dataset simulates a common scenario in educational statistics, where we examine the relationship between the number of hours a student studies and their final exam score. By creating synthetic data, we ensure the example is reproducible and clearly demonstrates the nonlinear relationship we intend to model.

We begin by setting a seed for reproducibility and then constructing a data frame containing 50 observations. The relationship is deliberately structured to exhibit a cubic tendency, mimicking a scenario where scores increase disproportionately as study hours grow longer, before applying random noise to simulate real-world variability.

The following R code chunk details the creation of the sample data frame, `df`, and displays the first six rows, allowing us to inspect the structure of the `hours` (predictor) and `score` (response) variables.

**#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(df)

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

## Fitting and Interpreting the Polynomial Model

With our dataset prepared, the next step is to fit a polynomial regression model. Given the observed curvature in the data generation process, we hypothesize that a quadratic (degree 2) polynomial will provide an excellent fit. We use the fundamental lm() function (linear model) in R, specifying the polynomial term using the `poly()` function.

Crucially, within the `lm()` call, we set `raw=T` inside `poly()`. This argument ensures that the terms are entered into the model as raw powers (X, $X^2$, $X^3$), rather than using orthogonal polynomials, which simplifies the interpretation of the resulting equation when we display it on the plot later. The output generated by the `summary()` function provides a comprehensive overview of the model's performance and significance.

The following code fits the quadratic model and displays its summary statistics, which will be essential for validating the output produced by **stat_poly_eq()** in the subsequent visualization step.

**#fit polynomial regression model with degree of 2**
**fit = lm(score ~ poly(hours, 2, raw=T), data=df)**

#view summary of model
summary(fit)

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

Residual standard error: 2.8 on 47 degrees of freedom
Multiple R-squared: 0.93, Adjusted R-squared: 0.927
F-statistic: 312.1 on 2 and 47 DF, p-value: < 2.2e-16

A careful inspection of the output reveals several key metrics. The Residuals summary indicates how closely the predicted values align with the actual data points. Most importantly for visualization purposes, the **Multiple R-squared** value is calculated as **0.93**. This high value suggests that 93% of the variability in the exam scores is explained by the quadratic relationship with study hours. Our objective now is to display this exact statistic directly on the scatterplot of the data.

## Visualizing the Model with stat_poly_eq()

The final step in our analysis is to create a compelling visualization that communicates both the raw data points and the fitted model, along with its statistical summary. We achieve this by leveraging the **ggplot2** package for plotting and integrating **stat_poly_eq()** to automatically annotate the results.

Before plotting, we must load the necessary libraries: `ggplot2` for the visualization framework and `ggpmisc`, which contains our target function. We also define the polynomial formula structure, ensuring it matches the formula used in the `lm()` call. This formula is passed to both `geom_smooth()` (to draw the curve) and `stat_poly_eq()` (to print the statistics).

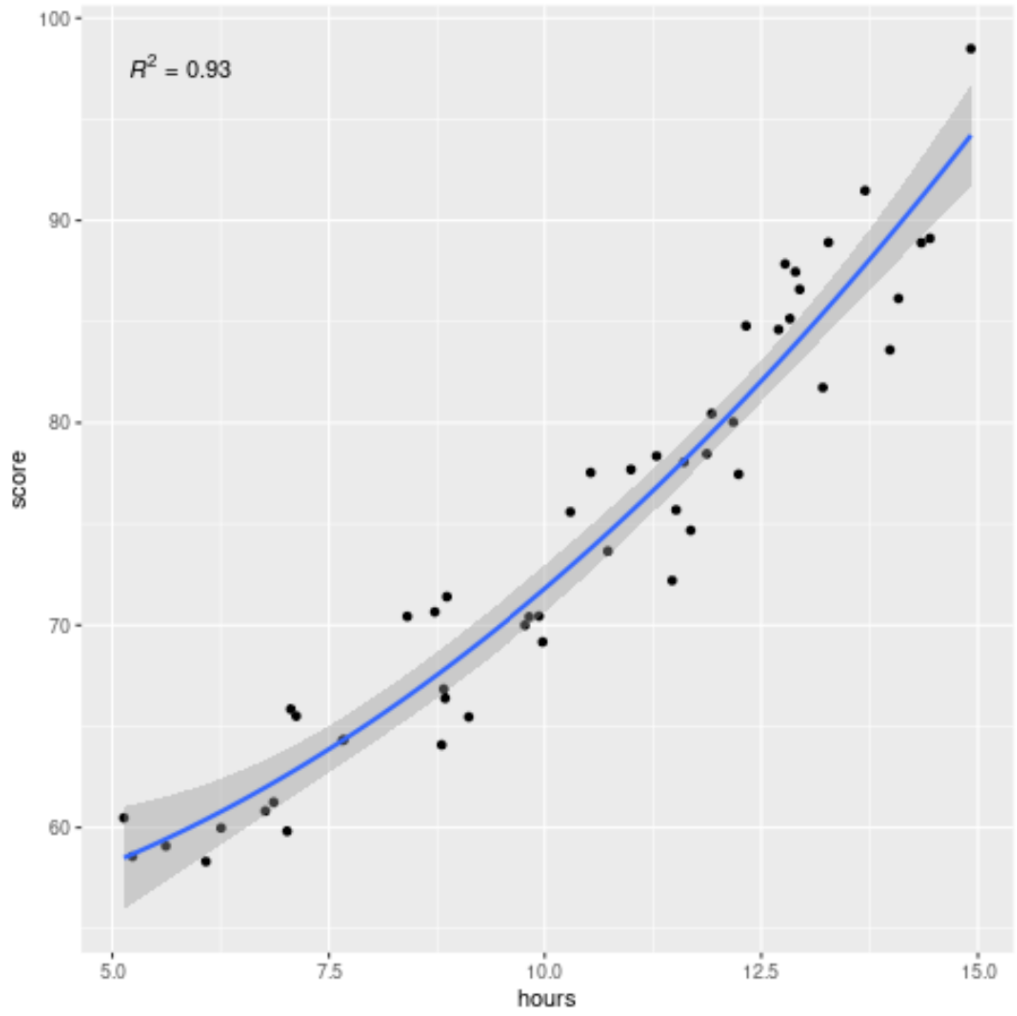
The `geom_smooth(method = "lm", formula = formula)` layer is crucial; it computes and plots the regression curve based on our linear model defined by the polynomial formula. Subsequently, the `stat_poly_eq(formula = formula, parse = TRUE)` layer calculates the equation and R-squared value for that same model and places the resulting text onto the plot area.

**library(ggplot2)**
**library(ggpmisc)**

#store fitted regression formula
formula <- y ~ poly(x, 2, raw = TRUE)

#create scatterplot with regression formula shown in plot
ggplot(df, aes(hours, score)) +
geom_point() +
geom_smooth(method = "lm", formula = formula) +
stat_poly_eq(formula = formula, parse = TRUE)

Executing this code produces the desired scatterplot visualization, complete with the fitted quadratic regression line and the model statistics displayed neatly within the plot area.




## Key Components of the Visualization Syntax

The visualization successfully maps the values from the **hours** column onto the x-axis and the **score** column onto the y-axis, represented by the scatterplot points. The most notable feature, facilitated by **stat_poly_eq()**, is the automatically generated annotation in the top left corner of the chart.

This annotation displays the full polynomial regression equation and the Multiple R-Squared value, which is confirmed to be **0.93**. This precisely matches the value we obtained earlier from the detailed `summary(fit)` output, validating the accuracy of the automated statistic generation. The inclusion of the argument `parse = TRUE` within **stat_poly_eq()** is essential, as it instructs R to render the output as mathematical notation, including superscripts and Greek letters, ensuring the equation is displayed in a professionally formatted manner.

The use of the **geom_smooth()** function, although optional for merely displaying the statistics, is

highly recommended. By including this layer, the user is provided with a visual representation of the fitted regression curve. This curve allows for an immediate visual assessment of how well the model aligns with the underlying data distribution, complementing the quantitative measure provided by the R-squared value. Both elements together provide a complete picture of the polynomial model's performance.

## Conclusion and Further Learning

The **stat_poly_eq()** function provides an unparalleled, streamlined approach to integrating complex statistical results directly into high-quality data visualizations using the **ggplot2** framework in R. This technique is invaluable for researchers and analysts who require clear, reproducible, and self-contained plots that communicate both the data distribution and the fitted model performance metrics, such as the regression equation and the goodness-of-fit statistic.

By automating the calculation and placement of these statistics, **ggpmisc** significantly enhances the efficiency and professional quality of statistical reporting. Mastering this method is key to producing dynamic and informative scatterplots when working with polynomial regression or other nonlinear models.

## Additional Resources

To continue developing your skills in R data analysis and visualization, explore related tutorials and functions:

How to Add Regression Lines to ggplot2 Plots

A Guide to Interpreting R-Squared Values

Understanding the Different Types of Residuals in Regression Analysis