# Week 06: Quantitative Scales, Curvilinear Relationships, and Transformations

Polynomials, transformations, and model choice

Bartosz Maćkiewicz

# From Diagnostics to Response

Up to this point quantitative predictors have mostly been treated as if they bore only a straight-line relation to the criterion. The next question is what to do when the data suggest that the relation is **curved rather than straight**.

The aim is to detect, represent, and study **nonlinear relationships** while staying as close as possible to the ordinary multiple-regression framework, using multiple and sometimes nonlinear aspects of the same research variable.

# Quantitative Scales and Curvature

Quantitative scales can take on a continuous range of values. Ordinal scales provide order. Interval scales add equal units. Ratio scales add equal units and equal ratios, hence a true zero.

Despite these differences in the amount of information they yield, all three can appear as quantitative predictors in regression. Some research problems require nonlinear aspects in addition to, or in place of, the linear aspect.

# What Do We Mean by Linear Regression?

The key distinction is between an equation that is linear in the **parameters** and one that is linear in the **variables**.

$$\hat{Y} = B_0 + B_1 X_1 + B_2 X_2 + \cdots + B_k X_k$$

Any relationship that can be written in this form can be handled in linear multiple regression. Each predictor is multiplied by a coefficient and the products are added to produce the predicted score. The relationship between a predictor and the criterion need not itself be a straight line.

# Linear in Parameters Is Not the Same as Linear in Variables

A straight-line relation between $X$ and $Y$ is only one special case. Linearity in the variables means that a one-unit increase in $X$ is associated with a constant increase in $Y$, regardless of where on the $X$ scale it occurs. If the slope changes over the range of $X$, the relation is no longer linear in the variables.

But that does not force us to abandon linear regression. We can write

$$\hat{Y} = B_0 + B_1 X + B_2 X^2$$

and the model is still linear in the coefficients $B_0$, $B_1$, and $B_2$. So long as an equation is linear in the parameters, it can be analyzed within the ordinary multiple-regression system.

# Four Approaches to Nonlinear Relationships

There are four broad approaches to nonlinear relationships in multiple regression. **Polynomial regression** represents one variable by several powers such as $X$, $X^2$, and $X^3$. **Monotonic nonlinear transformations** shrink or stretch parts of the scale so that transformed variables bear a closer-to-linear relation to one another.

When transformation cannot reduce the model to a linear form, **nonlinear regression** is used. **Nonparametric regression** is more modest still: a smooth curve such as lowess is allowed to follow the data first, so that the data can suggest what kind of parametric equation should come next.

# Synthetic Example A: Training and Retention

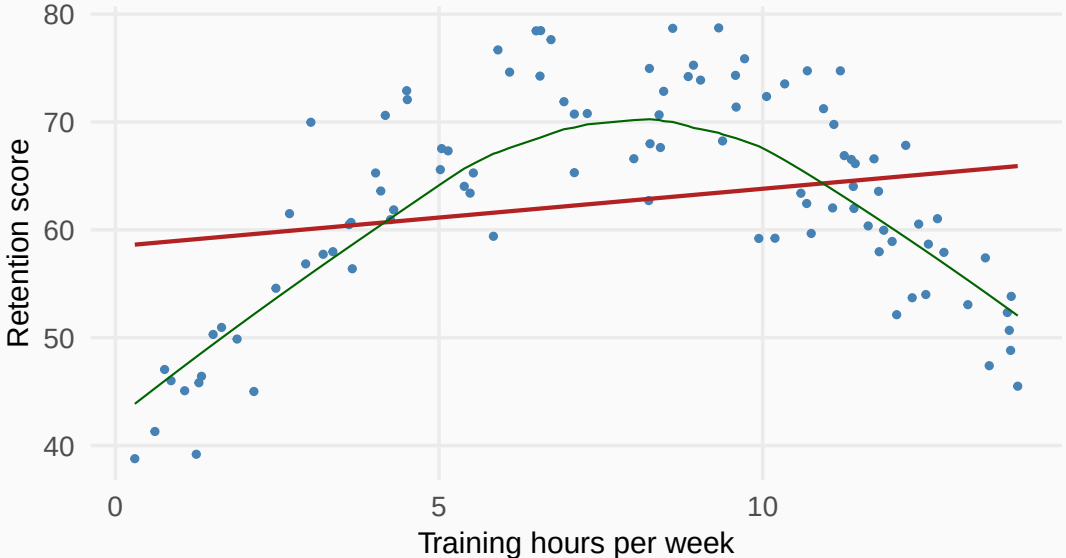
Suppose weekly training hours predict a later retention score on a skill task. At first, more training is helpful. After a point, however, fatigue and diminishing returns appear, so retention no longer rises indefinitely.

This is exactly the kind of situation in which a curvilinear relation is substantively plausible.

# Code: Scatterplot, Line, and Lowess

```r
curve <- lowess(training$training_hours, training$retention_score) |>
  as.data.frame()

p_scatter <- ggplot(training, aes(x = training_hours, y = retention_score)) +
  geom_point(color = "steelblue") +
  geom_smooth(method = lm, se = FALSE, color = "firebrick") +
  geom_line(
    data = curve,
    aes(x = x, y = y),
    inherit.aes = FALSE,
    color = "darkgreen"
  ) +
  labs(x = "Training hours per week", y = "Retention score")
```

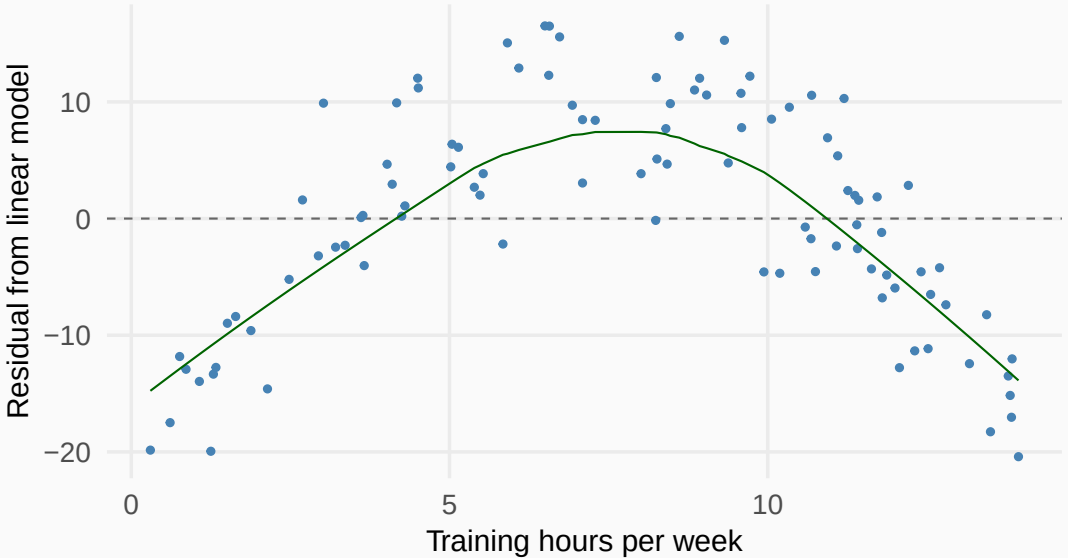
# The Scatterplot Already Suggests Curvature



# Code: Residual Plot and Lowess

```r
resid_curve <- lowess(
  training$training_hours,
  resid(fit_train_lin)
) |>
  as.data.frame()

p_resid <- ggplot(training, aes(x = training_hours, y = resid(fit_train_lin))) +
  geom_hline(yintercept = 0, linetype = 2, color = "grey40") +
  geom_point(color = "steelblue") +
  geom_line(
    data = resid_curve,
    aes(x = x, y = y),
    inherit.aes = FALSE,
    color = "darkgreen"
  ) +
  labs(x = "Training hours per week", y = "Residual from linear model")
```

# Removing the Linear Trend Reveals the Remaining Shape



# Power Polynomials

Polynomial regression expands one research variable into several aspects of itself. The equation includes stand-in variables such as $X^2$ and $X^3$ that possess a **known nonlinear relation** to the original variable.

Yet the regression equation remains **linear in the parameters** and can still be analyzed by ordinary least squares. A quadratic term allows **one bend**, a cubic term **two bends**, and increasingly higher-order terms can fit increasingly complex curves.

# Fit the Linear Model in R

```
m1 <- lm(retention_score ~ training_hours, data = training)
parameters::model_parameters(m1, ci = NULL, pretty_names = FALSE, verbose = FALSE)
```

```
Parameter      | Coefficient |   SE | t(98) |      p
-----------------------------------------------------
(Intercept)    |       58.47 | 2.15 | 27.25 | < .001
training_hours |        0.53 | 0.24 |  2.19 | 0.031
```

```
performance::model_performance(m1, metrics = "R2")
```

```
# Indices of model performance

R2
-----
0.046
```

# Add the Quadratic Term in R

```r
m2 <- lm(retention_score ~ training_hours + I(training_hours^2), data = training)
parameters::model_parameters(m2, ci = NULL, pretty_names = FALSE, verbose = FALSE)
```

```
Parameter           | Coefficient |   SE |  t(97) |      p
-----------------------------------------------------------
(Intercept)         |       35.52 | 1.67 |  21.23 | < .001
training_hours      |        9.49 | 0.52 |  18.19 | < .001
I(training_hours^2) |       -0.61 | 0.03 | -17.64 | < .001
```

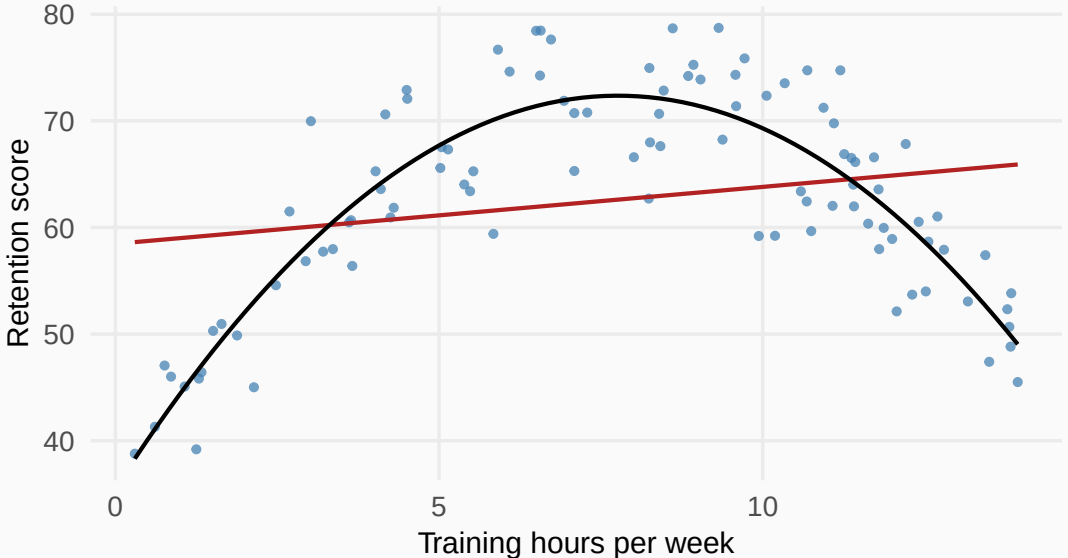
```r
performance::model_performance(m2, metrics = "R2")
```

```
# Indices of model performance

R2
-----
0.773
```

# Code: Linear and Quadratic Fits

```
p_quad <- ggplot(training, aes(x = training_hours, y = retention_score)) +
  geom_point(color = "steelblue", alpha = 0.75, size = 1.7) +
  geom_smooth(
    method = lm, formula = y ~ x, se = FALSE,
    color = "firebrick"
  ) +
  geom_smooth(
    method = lm, formula = y ~ x + I(x^2), se = FALSE,
    color = "black"
  ) +
  labs(x = "Training hours per week", y = "Retention score")
```

# A Quadratic Fit



# Compare Linear and Quadratic Models

```
anova(m1, m2)
```

```
Analysis of Variance Table

Model 1: retention_score ~ training_hours
Model 2: retention_score ~ training_hours + I(training_hours^2)
  Res.Df    RSS Df Sum of Sq      F    Pr(>F)
1     98 9293.6
2     97 2209.2  1    7084.4 311.06 < 2.2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

# Add the Cubic Term in R

```r
m3 <- lm(
  retention_score ~ training_hours + I(training_hours^2) + I(training_hours^3),
  data = training
)
parameters::model_parameters(m3, ci = NULL, pretty_names = FALSE, verbose = FALSE)
```

```
Parameter           | Coefficient |       SE | t(96) |      p
-----------------------------------------------------------------
(Intercept)         |       35.89 |     2.37 | 15.17 | < .001
training_hours      |        9.21 |     1.34 |  6.86 | < .001
I(training_hours^2) |       -0.57 |     0.21 | -2.70 | 0.008
I(training_hours^3) |   -2.11e-03 | 9.45e-03 | -0.22 | 0.824
```

```r
performance::model_performance(m3, metrics = "R2")
```

```
# Indices of model performance

R2
-----
0.773
```

# Compare Quadratic and Cubic Models

```
m3 <- lm(
  retention_score ~ training_hours + I(training_hours^2) + I(training_hours^3),
  data = training
)
anova(m2, m3)
```

```
Analysis of Variance Table

Model 1: retention_score ~ training_hours + I(training_hours^2)
Model 2: retention_score ~ training_hours + I(training_hours^2) + I(training_hours^3)
  Res.Df    RSS Df Sum of Sq      F Pr(>F)
1     97 2209.2
2     96 2208.0  1    1.1479 0.0499 0.8237
```

# The Higher-Order Term Tests Curvature

In this dataset, the linear model explains very little of the variation, with $R^2 \approx 0.05$. Adding the quadratic term raises that to about $R^2 \approx 0.77$.

The gain in prediction from adding the **quadratic term** is substantial, $F(1, 97) = 311.06$, $p < .001$. By contrast, adding a **cubic term** on top of the quadratic model contributes essentially nothing, $F(1, 96) = 0.05$, $p = 0.824$. The higher-order term tests whether that additional aspect of the curve contributes **over and above** all lower-order terms already in the equation.

# Higher-Order Terms Require Lower-Order Terms

One practical rule is **nonnegotiable**: if a model contains $X^2$, it should also contain $X$. If it contains $X^3$, it should also contain $X$ and $X^2$.

The reason is interpretive. In order that higher-order terms have meaning, **all lower-order terms must be included**. Otherwise the higher-order coefficient confounds simpler and more complex parts of the relationship.

# What Order Should We Estimate?

From a mathematical standpoint, one can fit increasingly high-order polynomials and eventually capture every random jiggle in the data. From a research standpoint, that is seldom useful.

Theory should guide the order of the equation. In most behavioral applications, **quadratic and at most cubic relations** are the practical focus. Higher-order coefficients are difficult to interpret, and social-science data rarely support explorations above the cubic level.

# Centering Predictors in Polynomial Equations

Lower-order coefficients in higher-order regression equations have a useful interpretation only when the variable has a **meaningful zero**. When a polynomial equation includes both $X$ and $X^2$, the meaning of the linear coefficient depends on the value of $X = 0$. In many applications that value is arbitrary or even outside the observed range.

**Centering** solves that problem. By subtracting the mean of $X$ from every score, zero becomes the **sample mean**. The first-order coefficient is then interpreted at a meaningful location on the scale rather than at an artificial point.

# Fit the Centered Quadratic Model in R

```
training$hours_c <- training$training_hours - mean(training$training_hours)
m_cent <- lm(retention_score ~ hours_c + I(hours_c^2), data = training)
parameters::model_parameters(
  m_cent, ci = NULL, pretty_names = FALSE, verbose = FALSE
)
```

```
Parameter    | Coefficient |   SE |  t(97) |      p
-------------------------------------------------
(Intercept)  |       72.36 | 0.73 |  99.29 | < .001
hours_c      |       -0.08 | 0.12 |  -0.60 | 0.548
I(hours_c^2) |       -0.61 | 0.03 | -17.64 | < .001
```

# Compare Collinearity Before and After Centering

```
performance::check_collinearity(m2)
```

```
# Check for Multicollinearity

High Correlation

                Term   VIF     VIF 95% CI adj. VIF Tolerance Tolerance 95% CI
      training_hours 19.00 [13.29, 27.34]     4.36      0.05     [0.04, 0.08]
 I(training_hours^2) 19.00 [13.29, 27.34]     4.36      0.05     [0.04, 0.08]
```

```
performance::check_collinearity(m_cent)
```

```
# Check for Multicollinearity

Low Correlation

         Term  VIF   VIF 95% CI adj. VIF Tolerance Tolerance 95% CI
      hours_c 1.08 [1.01, 2.12]     1.04      0.92     [0.47, 0.99]
 I(hours_c^2) 1.08 [1.01, 2.12]     1.04      0.92     [0.47, 0.99]
```

# Uncentered and Centered Quadratic Equations

For the training example, the uncentered and centered quadratic equations are:

$$\hat{Y} = 35.52 + 9.49X - 0.61X^2$$

$$\hat{Y} = 72.36 - 0.08X_c - 0.61X_c^2$$

The **second-order coefficient is unchanged**, but the lower-order terms move because centering changes the location at which zero is defined.

# What Centering Buys Us

After centering, the linear coefficient describes the **slope of the curve at the mean level of training** rather than at an artificial zero point. In this example that coefficient is close to zero, which is exactly what one would expect near the broad top of the curve.

Centering also removes much of the **nonessential multicollinearity** that appears when $X$, $X^2$, and $X^3$ are entered together. The **shape of the fitted curve does not change**, but the lower-order coefficients become far easier to interpret.

# Strategy and Limitations

Polynomial regression is a convenient way to fit smooth curvature, but it remains a **strategy of approximation**. A **build-up strategy** examines the linear equation, then the quadratic, then the cubic. A **tear-down strategy** begins with the highest order of theoretical interest and simplifies only if that highest-order term contributes little.

No rigid rule settles the matter. Theory, effect size, plots, and residual behavior all matter, and outliers near the ends of the scale can dramatically alter the detected order. Higher-order polynomials should not be fitted merely because the software allows them.

The same general idea can also be expressed by recoding the curve components so that linear, quadratic, and cubic trends are **orthogonal** to one another. This is most natural when the predictor has a small number of **ordered, equally spaced levels** and there are equal numbers of observations at each level, as in a designed experiment.

The attraction is computational and interpretive convenience: the **separate trend components no longer overlap** in the same way they do when ordinary powers of $X$ are used.

# Synthetic Example: Ordered Dose Levels

Suppose five equally spaced dose levels are administered, with the same number of observations at each level. That is the setting in which orthogonal polynomial trends are most natural: ordered levels, equal spacing, and equal $n$.

# Raw Powers and Orthogonal Bases in R

```
round(poly(1:5, 2, raw = TRUE)[, ], 2)
```

```
     1  2
[1,] 1  1
[2,] 2  4
[3,] 3  9
[4,] 4 16
[5,] 5 25
```

```
round(poly(1:5, 2)[, ], 3)
```

```
          1      2
[1,] -0.632  0.535
[2,] -0.316 -0.267
[3,]  0.000 -0.535
[4,]  0.316 -0.267
[5,]  0.632  0.535
```

# Fit Orthogonal Polynomial Trends in R

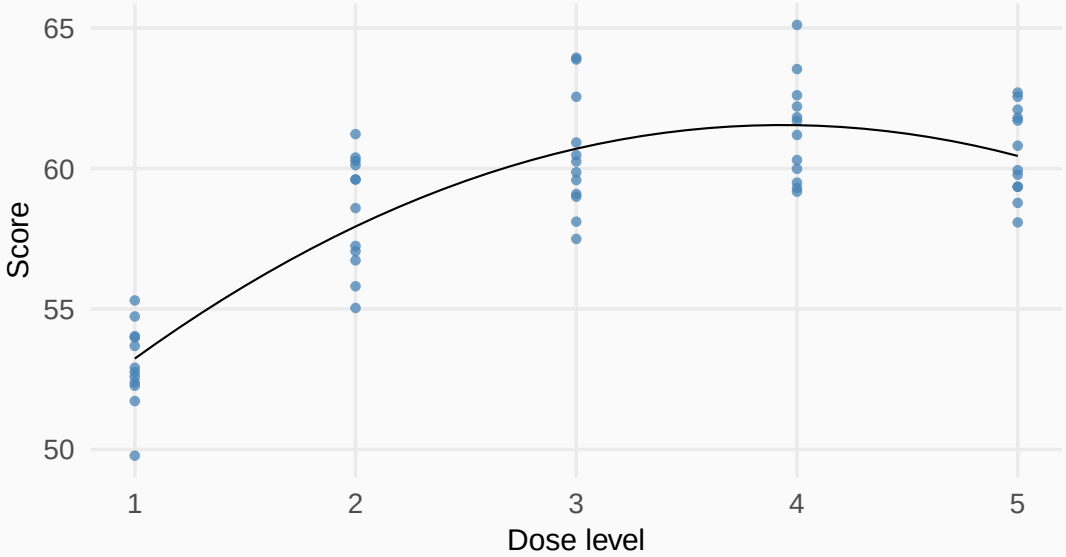
```r
set.seed(602)
orthogonal <- tibble::tibble(dose = rep(1:5, each = 12))
orthogonal$score <- 48 + 7 * orthogonal$dose - 0.9 * orthogonal$dose^2 +
  rnorm(nrow(orthogonal), sd = 1.8)

m_ortho <- lm(score ~ poly(dose, 2), data = orthogonal)
parameters::model_parameters(
  m_ortho, ci = NULL, pretty_names = FALSE, verbose = FALSE
)
```

```
Parameter      | Coefficient |   SE |  t(57) |      p
-----------------------------------------------------
(Intercept)    |       58.77 | 0.23 | 251.52 | < .001
poly(dose, 2)1 |       19.76 | 1.81 |  10.92 | < .001
poly(dose, 2)2 |      -12.52 | 1.81 |  -6.92 | < .001
```

# Code: Plot the Orthogonal Polynomial Fit

```r
curve_ortho <- data.frame(dose = seq(1, 5, by = 0.1))
curve_ortho$score <- predict(m_ortho, newdata = curve_ortho)

p_ortho <- ggplot(orthogonal, aes(x = dose, y = score)) +
  geom_point(color = "steelblue", alpha = 0.75, size = 1.8) +
  geom_line(
    data = curve_ortho,
    aes(x = dose, y = score),
    inherit.aes = FALSE,
    color = "black"
  ) +
  labs(x = "Dose level", y = "Score")
```

# Orthogonal Polynomial Fit



# When Orthogonal Polynomials Are Worth It

For ordinary observational data with many distinct values of a predictor, the simpler workflow is usually enough: scatterplots, lowess, ordinary powers of $X$, and careful interpretation. Orthogonal polynomials are more of a special-purpose tool for ordered conditions than a general workflow for everyday regression.

They are worth knowing about, but they are **not at the center** of this week's practical toolbox.

# Why Transform?

From a statistical perspective, transformations have three overarching goals. They may **simplify the relationship**, they may **eliminate heteroscedasticity**, and they may make residuals **closer to normality**.

In many cases, one transformation helps with all three. But that is **not inevitable**. A transformation may improve one aspect of the regression problem while degrading another, so the transformed model must also be checked.

# Check Before and After Transformation

Diagnostic plots should be used *before* transformation to see whether the relationship is nonlinear, whether residual spread changes with the fitted values, and whether a few influential observations are creating the apparent problem.

Before transforming, it is also important to ask whether one or a few influential points are producing the apparent nonlinearity or assumption violations. *After* transforming, the model should be checked again, because a transformation that remedies one aspect of the regression situation may create problems in another aspect of the regression situation.

Many useful transformations involve logarithms and powers. These are nonlinear transformations: they stretch or contract the scale nonuniformly. The essential issue is **scale**. A log transform stretches one part of a scale and compresses another, and by doing so it can make a nonlinear relation easier to represent in a linear model.

Natural logs and base-10 logs differ only by a linear rescaling. For regression work, that means the substantive choice is the log transformation itself, not the base of the logarithm.

# Additive and Proportional Change

One recurring theme is whether changes are *additive* or *proportional*.

If $X$ changes by a constant proportion while $Y$ changes by a constant additive amount, then $Y$ tends to be linear in $\log X$. If constant additive changes in $X$ are associated with proportional changes in $Y$, then $\log Y$ tends to be linear in $X$. If both variables change proportionally, then $\log Y$ tends to be linear in $\log X$.

That logic is the conceptual core behind many commonly used transformations.

# Strong and Weak Theoretical Models

Sometimes a **substantive theory** already implies a functional form. In that case the equation is not just a curve that happens to fit; it is a statement about the process being studied. Transformation then serves to bring that equation into a form that can be estimated with ordinary least squares.

More often, theory is weaker. Then logs, reciprocals, and related transformations are used more modestly, as plausible scale changes rather than as direct tests of a formal model.

When theory is weak, graphical guidance becomes especially useful. The **ladder of re-expression** and the **bulging rule** are pragmatic devices. One inspects the scatterplot, superimposes a lowess curve, and asks what kind of stretching or compression of the scales would straighten the relation.

Two further points matter in practice. First, one may sometimes transform either $X$ or $Y$. Second, the residuals help decide which is wiser: if residuals are already well behaved, transforming $X$ is safer; if residuals are problematic, transforming $Y$ may solve more than one problem at once. The bulging rule does not replace judgment; it gives a disciplined starting point.

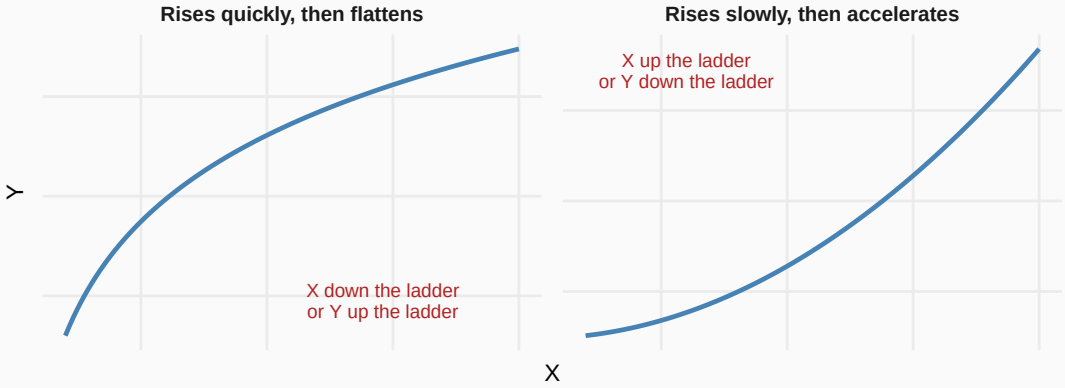
# The Ladder of Re-Expression

The ladder organizes familiar transformations as a series of **power transformations**. Moving down the ladder compresses large values more strongly; moving up stretches large values. The logarithm is treated as the special case for $\lambda = 0$.

| Power $\lambda$ | Re-expression | Main scale effect |
|---|---|---|
| $2$ | $Y^2$ | stretches the high end |
| $1$ | $Y$ | leaves the scale unchanged |
| $.5$ | $\sqrt{Y}$ | mildly compresses the high end |
| $0$ | $\log(Y)$ | strongly compresses the high end |
| $-1$ | $1/Y$ | very strong compression and reversal |

# The Bulging Rule

The bulging rule uses the shape of a scatterplot with a lowess curve to suggest which scale to re-express. The aim is not to find a magical rule, but to choose a plausible transformation that would pull the curve closer to a straight line.



# Synthetic Example B: Practice and Completion Time

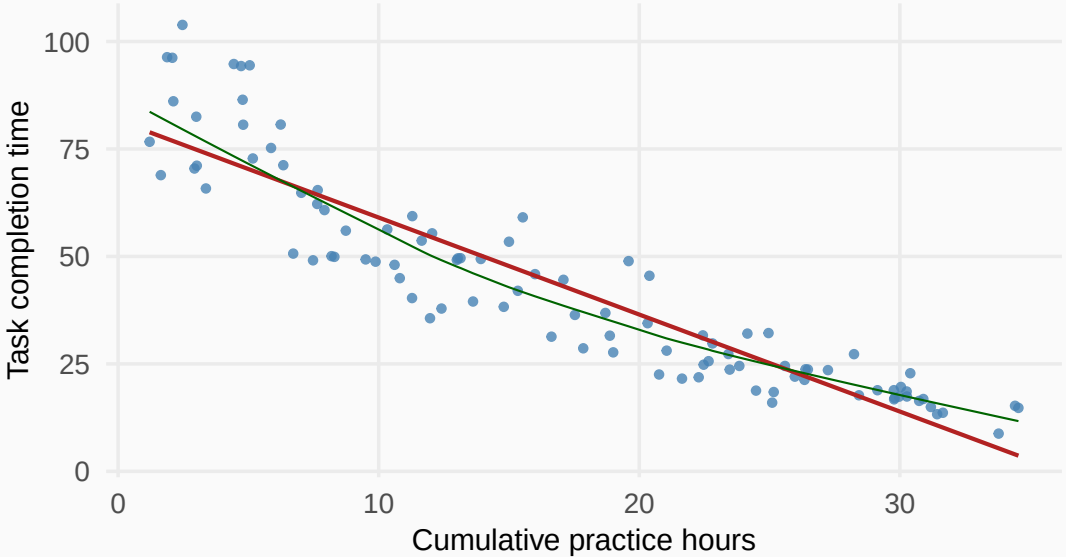
The transformation example again uses synthetic data. Suppose cumulative practice hours predict the time needed to complete a task. Early practice produces large improvements, but later practice produces smaller and smaller gains. Completion time remains positive throughout, and its variability is larger on the raw scale than on a suitable transformed scale.

This is the kind of monotonic nonlinear relationship in which a log transformation of the outcome is substantively plausible and statistically useful.

# Code: Raw Relationship, Line, and Lowess

```r
curve_raw <- lowess(practice$practice_hours, practice$completion_time) |>
  as.data.frame()

p_raw <- ggplot(practice, aes(x = practice_hours, y = completion_time)) +
  geom_point(color = "steelblue", alpha = 0.8, size = 1.8) +
  geom_smooth(method = lm, formula = y ~ x, se = FALSE, color = "firebrick") +
  geom_line(
    data = curve_raw,
    aes(x = x, y = y),
    inherit.aes = FALSE,
    color = "darkgreen"
  ) +
  labs(x = "Cumulative practice hours", y = "Task completion time")
```

# Raw-Scale Relationship



# Fit the Raw-Scale Model in R

```
m_raw <- lm(completion_time ~ practice_hours, data = practice)
parameters::model_parameters(
  m_raw, ci = NULL, pretty_names = FALSE, verbose = FALSE
)
```

```
Parameter      | Coefficient |   SE |  t(98) |      p
-----------------------------------------------------
(Intercept)    |       81.61 | 1.93 |  42.29 | < .001
practice_hours |       -2.26 | 0.10 | -22.97 | < .001
```

```
performance::model_performance(m_raw, metrics = "R2")
```

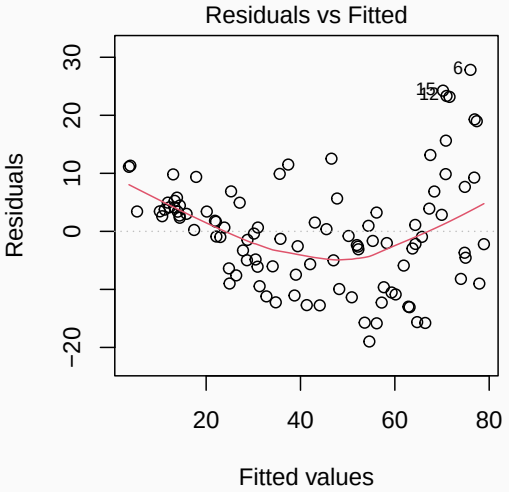
```
# Indices of model performance

R2
-----
0.843
```

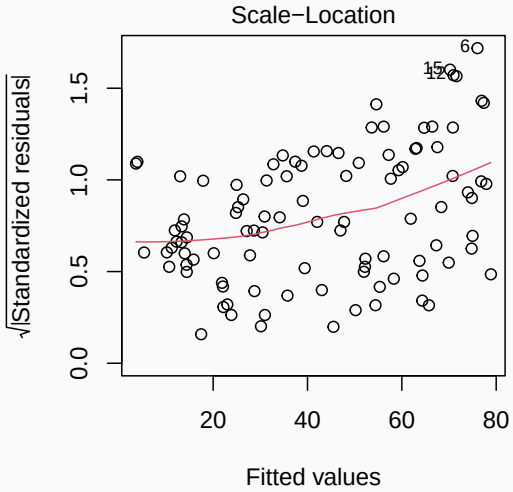
# Code: Raw-Scale Diagnostics

```
m_raw <- lm(completion_time ~ practice_hours, data = practice)
par(mfrow = c(1, 2))
plot(m_raw, which = 1)
plot(m_raw, which = 3)
```

# Raw-Scale Diagnostics

Residual structure and increasing spread are still visible on the raw scale.



The nice thing about this workflow is that the syntax remains very simple.

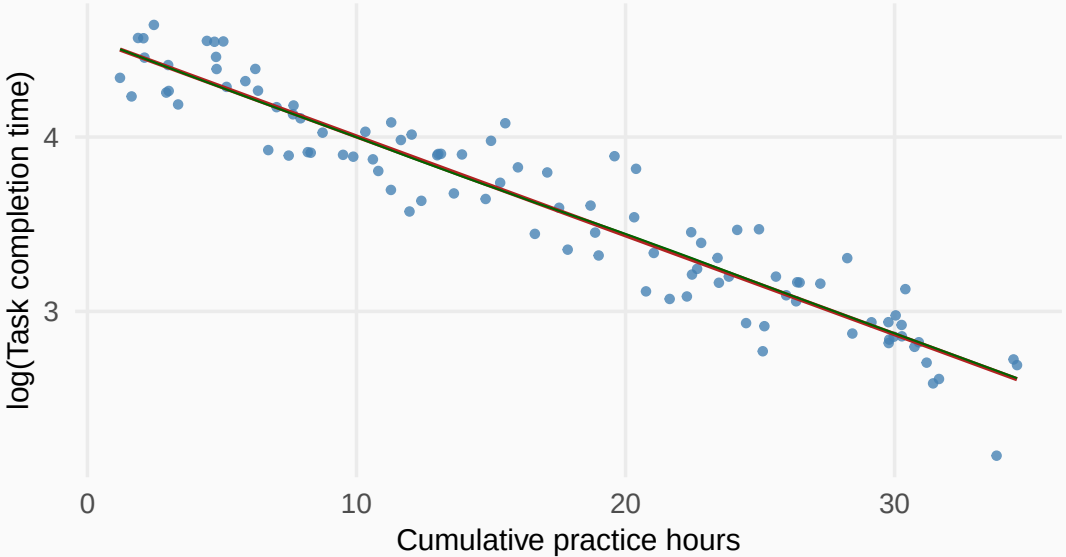
```
fit_practice_log <- lm(
  log(completion_time) ~ practice_hours,
  data = practice
)

fit_train_quad <- lm(
  retention_score ~ training_hours + I(training_hours^2),
  data = training
)
```

The important work is **conceptual rather than computational**.

# Code: Log-Transformed Relationship

```
curve_log <- lowess(
  practice$practice_hours,
  log(practice$completion_time)
) |>
  as.data.frame()

p_log <- ggplot(practice, aes(x = practice_hours, y = log(completion_time))) +
  geom_point(color = "steelblue", alpha = 0.8, size = 1.8) +
  geom_smooth(method = lm, formula = y ~ x, se = FALSE, color = "firebrick") +
  geom_line(
    data = curve_log,
    aes(x = x, y = y),
    inherit.aes = FALSE,
    color = "darkgreen"
  ) +
  labs(x = "Cumulative practice hours", y = "log(Task completion time)")
```

# The Log-Transformed Relation



# Fit the Log-Scale Model in R

```
m_log <- lm(log(completion_time) ~ practice_hours, data = practice)
parameters::model_parameters(
  m_log, ci = NULL, pretty_names = FALSE, verbose = FALSE
)
```

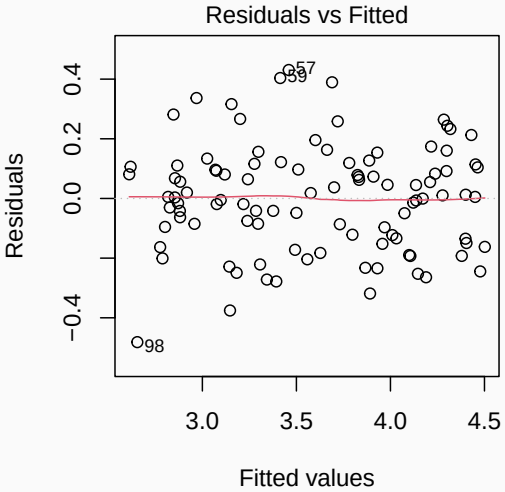
```
Parameter      | Coefficient |       SE |  t(98) |      p
-------------------------------------------------------------
(Intercept)    |        4.57 |     0.04 | 127.09 | < .001
practice_hours |       -0.06 | 1.83e-03 | -30.98 | < .001
```

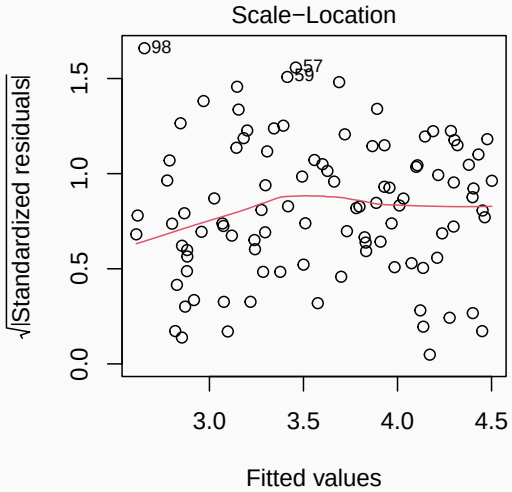
```
performance::model_performance(m_log, metrics = "R2")

# Indices of model performance

R2
-----
0.907
```

# Code: Diagnostics After Transformation

```
m_log <- lm(log(completion_time) ~ practice_hours, data = practice)
par(mfrow = c(1, 2))
plot(m_log, which = 1)
plot(m_log, which = 3)
```

After transformation, both the residual structure and the variance pattern are milder.



# Box-Cox Makes the Same Question More Systematic

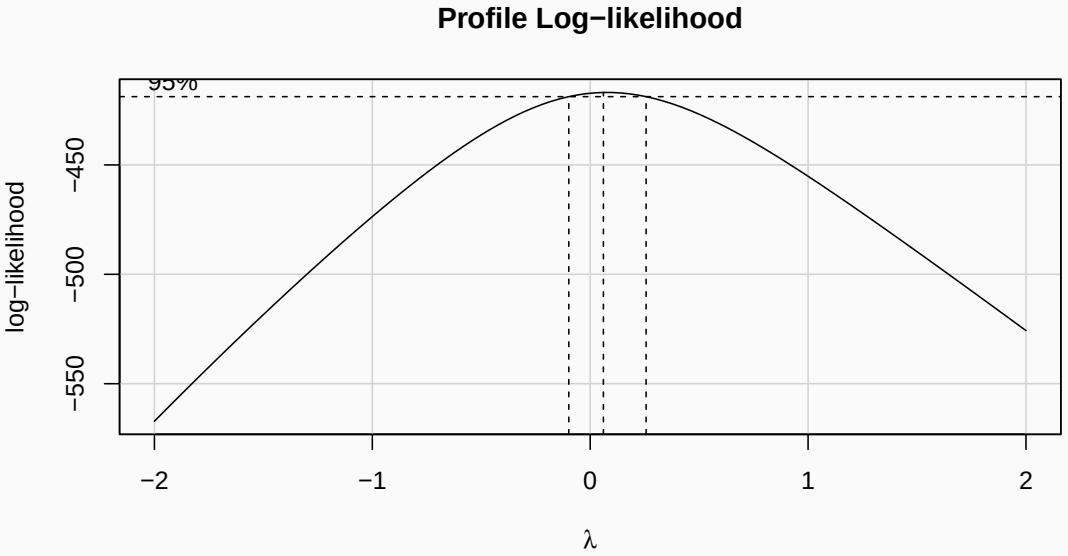
For positive outcomes, Box-Cox embeds the log transform in a family of power transformations:

$$y^{(\lambda)} = \begin{cases} \dfrac{y^{\lambda} - 1}{\lambda}, & \lambda \neq 0 \\ \log(y), & \lambda = 0 \end{cases}$$

Its value is that it lets us examine a family of plausible scales in a **principled** way rather than by ad hoc trial and error.

# Code: Box-Cox Transformation

```r
m_raw <- lm(completion_time ~ practice_hours, data = practice)
car::boxCox(m_raw, lambda = seq(-2, 2, 0.1))
```


Profile Log−likelihood
95%
log−likelihood
−450
−500
−550
−2
−1
0
1
2
λ

# Extract the Estimated Lambda in R

```
pt <- car::powerTransform(completion_time ~ practice_hours, data = practice)
coef(pt)
```

```
        Y1
0.07771351
```

# Interpreting the Box-Cox Result

For this dataset, the estimated transformation parameter is **close to zero**, $\hat{\lambda} \approx 0.08$. That is exactly the situation in which a **log transform** is a reasonable simplification of the broader power family.

# Compare Fits on the Original Scale

```
c(
  raw = cor(practice$completion_time, predict(m_raw))^2,
  log_back = cor(practice$completion_time, exp(predict(m_log)))^2
)
```

```
      raw  log_back
0.8433736 0.8852057
```

Two cautions matter here. First, one must **inspect the transformed model rather than assuming improvement**. Second, one should **not compare $R^2$ values mechanically** when the dependent variable has been changed.

In the practice data, the raw model has $R^2 \approx 0.84$ and the log-scale model has $R^2 \approx 0.91$, but those quantities live on **different outcome scales**. To compare fits fairly, predicted values from the transformed model must be moved back into the **original metric** before model fit is judged.

The choice between an untransformed and a transformed analysis depends on several things: whether theory implies a particular scale, whether the transformed equation gives a better account of the phenomenon, whether overall fit is materially improved, and whether transformation introduces new difficulties into the model.

The warning is just as important. **An effect can be transformed away.** A curvilinear relation that is central to the theory should not be removed merely because a monotonic transformation produces a straighter plot. **Interpretability** remains one of the main criteria of judgment.

Nonlinear regression is needed when the model is **intrinsically nonlinear** and cannot be reduced to an ordinary least-squares equation by an appropriate transformation.

In that setting, the analyst specifies the nonlinear equation **directly**. The coefficients are then found **iteratively** rather than by the closed-form machinery of ordinary least squares.

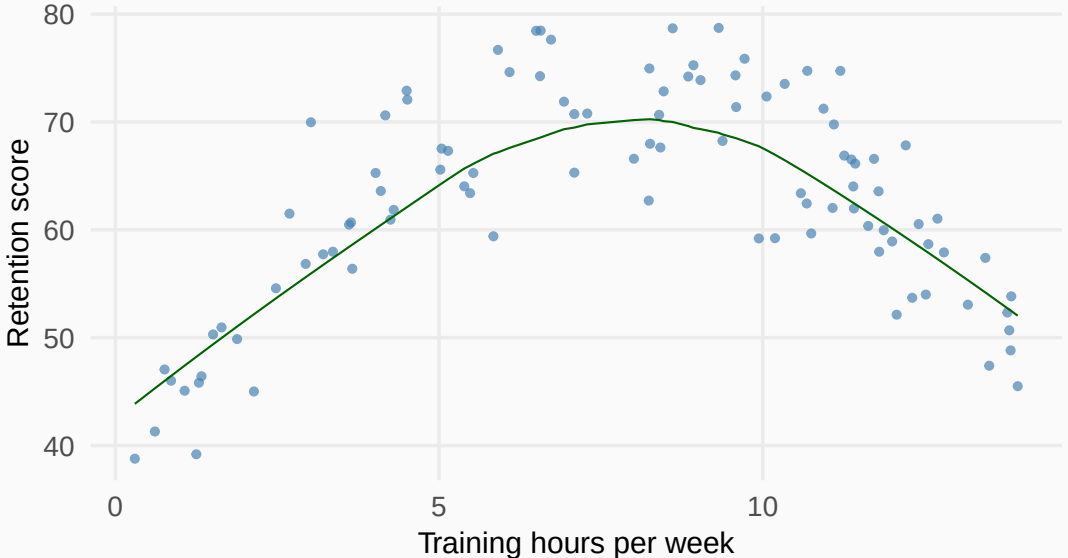
Nonparametric regression begins even more modestly. Instead of choosing a parametric equation first, it lets a smooth function trace the relation that the data suggest.

**Lowess** is the key example. It does not produce a regression coefficient in the usual sense, but it is extremely useful for seeing whether a relation is straight, curved, or bent in a way that suggests a transformation, a polynomial term, or a genuinely nonlinear model.

# Code: Lowess as a Shape Guide

```
curve_lowess <- lowess(training$training_hours, training$retention_score) |>
  as.data.frame()

p_lowess <- ggplot(training, aes(x = training_hours, y = retention_score)) +
  geom_point(color = "steelblue", alpha = 0.7, size = 1.7) +
  geom_line(
    data = curve_lowess,
    aes(x = x, y = y),
    inherit.aes = FALSE,
    color = "darkgreen"
  ) +
  labs(x = "Training hours per week", y = "Retention score")
```

# Lowess as a Guide Rather Than a Final Answer



# Summary

Multiple regression can be used to study the **shape of the relationship** between predictors and the dependent variable when variables are measured on ordinal, interval, or ratio scales. **Polynomial terms** capture curvilinear relations. **Transformations** are used to achieve linearity and to improve residual behavior. **Nonlinear and nonparametric regression** remain available when those routes are not enough.

The practical toolbox for this week is correspondingly small: `lowess()`, `lm()`, polynomial terms written with `I(x^2)`, diagnostic plots from `plot(fit)`, and Box-Cox from `car::boxCox()`.