Model Interpretation and Diagnostics

Francisco Villamil

Applied Quantitative Methods II

MA in Social Sciences, Spring 2026

Today's goals

- Move beyond coefficient tables to meaningful quantities
- Compute and plot predicted values, marginal effects, and first differences
- Present results with publication-quality tables and coefficient plots
- Understand simulation-based uncertainty
- Diagnose common regression problems: heteroskedasticity, non-linearity

This session bridges estimation and communication. Last week we saw that logit coefficients are hard to interpret and learned to use marginal effects and predicted probabilities. Today we generalize that lesson: even in OLS, raw coefficients are often not the best way to communicate results. We'll learn a toolkit for computing and presenting quantities of interest, plus some basic diagnostics to check whether our model assumptions hold.

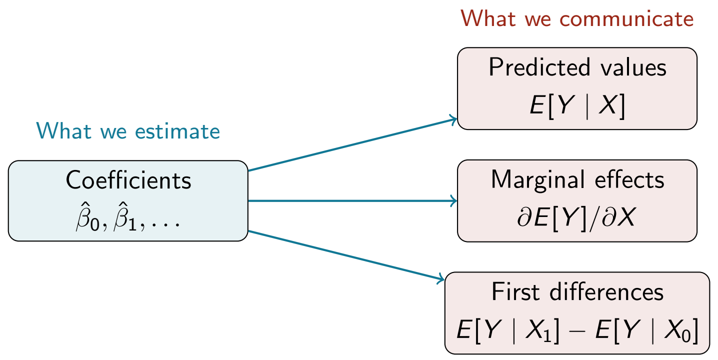
Roadmap

The interpretation problem

- We know how to estimate models and read output
- But what do the numbers actually **mean**?
- Raw coefficients are often not directly informative:
  - → Logit: coefficients are in log-odds (last week)
  - → Interactions: the coefficient on X_1 depends on X_2
  - → Log transformations: coefficients are elasticities or semi-elasticities
  - → Different scales: is $\hat{\beta} = 0.003$ big or small?

Start by recapping the lesson from last week's logit discussion: raw coefficients in log-odds are meaningless to a general audience. But the same problem applies to OLS in many cases. When you have interaction terms, the "main effect" coefficient only applies when the other variable is zero, which may not even be in the data. When variables are on different scales, comparing coefficient magnitudes is misleading. The solution is to compute quantities that have direct substantive meaning.

Coefficients vs. quantities of interest



This diagram captures the key idea of the lecture. Coefficients are the raw output of estimation, but they are rarely the end product. What readers and audiences care about are quantities of interest: predicted values (what does the model predict for a specific scenario?), marginal effects (how much does the outcome change when a predictor changes?), and first differences (how different are two scenarios?). All three are computed from the coefficients, but they present the information in a more interpretable way. The marginaleffects package in R makes all three easy to compute.

You estimate a model with GDP (in dollars), population (in millions), and an interaction between them.

What does the coefficient on GDP tell you?

Discussion prompt. The answer: the coefficient on GDP gives the effect of GDP when population equals zero, which is meaningless. With interactions, the "main effect" is conditional on the other variable being at zero. And the scale issue: GDP in dollars produces tiny coefficients, population in millions produces large ones. You can't compare them directly. This motivates the need for predicted values and marginal effects that account for the full model structure.

Roadmap

Predicted values: $E[Y \mid X]$

- The most intuitive quantity: "What does the model predict for this scenario?"
- Set predictors to specific values, compute $\hat{Y}$
- Example: "What is predicted income for a 35-year-old with a college degree?"
- In R:
  - → `predictions(model)`
  - → `predictions(model, newdata = datagrid(age = 35, college = 1))`

Predicted values are the most direct way to interpret any model. You plug in values for the predictors and see what the model outputs. This works for OLS, logit, or any other model. The `predictions()` function from the marginaleffects package computes predictions with standard errors and confidence intervals. Without `newdata`, it returns predictions for every observation in the data. With `datagrid()`, you can specify hypothetical profiles. The function handles all the math, including the delta method for standard errors.

Plotting predictions

- The gold standard for communicating model results
- `plot_predictions(model, condition = "age")`
  - → Shows $\hat{Y}$ across values of age
  - → Includes 95% confidence band
  - → Holds other variables at their means (or modes)
- With two conditions:
  - → `plot_predictions(model, condition = c("age", "female"))`
  - → Separate lines/panels for each group

Prediction plots are the single most effective way to communicate regression results to any audience. A line showing how predicted income changes with age is immediately understandable, whereas a coefficient table requires statistical training to parse. The `plot_predictions()` function produces ggplot2 objects, so you can customize them further with standard ggplot2 syntax. Emphasize that the confidence band reflects uncertainty in the regression line (not prediction intervals for individual observations). When you use two conditions, you effectively visualize an interaction.

Marginal effects

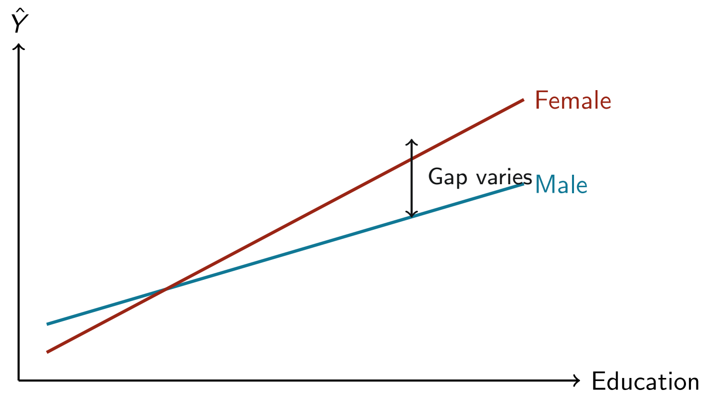
$$\text{Marginal effect of } X_1 = \frac{\partial E[Y \mid X]}{\partial X_1}$$

- "How much does Y change for a small change in X_1?"
- For OLS without interactions: marginal effect $= \hat{\beta}_1$ (constant)
- For logit, interactions, polynomials: marginal effect **varies**
- **Average marginal effect (AME)**: average across all observations
  - → `avg_slopes(model)`

The marginal effect is the derivative of the predicted value with respect to a predictor. For a simple OLS model $Y = \beta_0 + \beta_1 X$, the marginal effect is just β_1, everywhere. But as soon as you introduce interactions ($Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_1 X_2$), the marginal effect of X_1 becomes $\beta_1 + \beta_3 X_2$, which depends on X_2. For logit, the marginal effect depends on all predictors (as we saw last week). The AME averages the observation-specific marginal effects across the sample, giving a single summary number.

Marginal effects with interactions

$$Y = \beta_0 + \beta_1\text{Education} + \beta_2\text{Female} + \beta_3(\text{Education} \times \text{Female}) + \varepsilon$$



- ME of Education for males: β_1
- ME of Education for females: $\beta_1 + \beta_3$

This is the classic case where reporting just the coefficient on Education is misleading. The coefficient β_1 is only the effect for males (Female $= 0$). For females, the effect is $\beta_1 + \beta_3$. Neither the coefficient on Education nor the coefficient on the interaction alone tells you the full story. You need to compute marginal effects at specific values of Female. The plot makes this clear: the slopes differ. Using `avg_slopes(model, variables = "education", by = "female")` gives you the marginal effect separately for each group.

Conditional marginal effects in R

- Marginal effect of X_1 at specific values of X_2:
  →
```
slopes(model, variables = "education",
          newdata = datagrid(female = c(0, 1)))
```
- Plot how the marginal effect varies:
  →
```
plot_slopes(model, variables = "education",
          condition = "female")
```
- This replaces the old manual approach of computing $\beta_1 + \beta_3 \cdot X_2$ by hand

The `slopes()` function computes marginal effects at specific predictor values. Combined with `datagrid()`, you can evaluate the effect at any combination of covariate values. The `plot_slopes()` function visualizes how the marginal effect changes across the conditioning variable, with confidence intervals. This is especially useful for continuous-by-continuous interactions, where the effect of X_1 varies smoothly across values of X_2. Emphasize that this replaces the tedious and error-prone approach of manually combining coefficients and computing standard errors with the delta method.

First differences (comparisons)

- "How different is $\hat{Y}$ between two scenarios?"
- Example: predicted income for college vs. no college
  - → $\Delta = E[Y \mid \text{college} = 1] - E[Y \mid \text{college} = 0]$
- In R:
  - → `comparisons(model, variables = list(college = c(0, 1)))`
  - → `avg_comparisons(model, variables = "college")`
- Especially useful for:
  - → Binary/categorical predictors
  - → Comparing meaningful scenarios (e.g., min vs. max)

First differences answer the most natural research question: "How much does the outcome change when we move from one scenario to another?" For a binary predictor like college, the first difference is the predicted gap between college-educated and non-college-educated individuals, averaged across all other characteristics. The `comparisons()` function computes this with proper uncertainty. You can also define custom comparisons: `variables = list(age = c(25, 65))` gives you the predicted difference between 25-year-olds and 65-year-olds. This is far more informative than saying "the coefficient on age is 0.003."

Choosing the right quantity

Quantity	**Question**	**R function**
Predicted value	What does the model predict here?	`predictions()`
Marginal effect	How much does Y change per unit of X?	`slopes()`
Average ME	What is the average effect across the sample?	`avg_slopes()`
First difference	How different are two scenarios?	`comparisons()`

- All from the `marginaleffects` package (Arel-Bundock et al.)
- All compute standard errors and CIs automatically
- All work with `lm()`, `glm()`, and many other model types

This is a reference slide that students can come back to. The marginaleffects package provides a unified interface for all these quantities. The key insight: you don't need to know the formula for the delta method or how to compute standard errors for nonlinear transformations. The package handles everything. Emphasize the consistency: the same functions work regardless of whether you're using OLS, logit, or other models. The package documentation (the Arel-Bundock, Greifer, and Heiss online book) is excellent and has many worked examples.

Roadmap

Publication-quality tables with `modelsummary()`

- We already know `modelsummary()` from Session 2
- Key options:
  - → `stars = TRUE` for significance stars
  - → `vcov = "robust"` for robust SEs
  - → `coef_rename` to clean up variable names
  - → `gof_map` to select goodness-of-fit statistics
- Multiple models side by side:
  - → `modelsummary(list("OLS" = m1, "Logit" = m2))`
- Output to LaTeX, Word, HTML, or the console

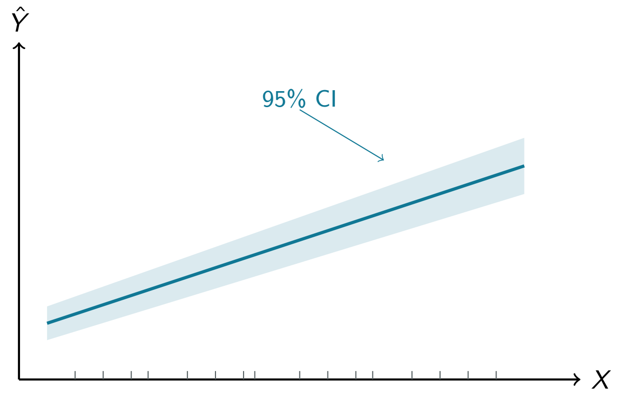
Quick recap since they've seen modelsummary before. The main new points: (1) `coef_rename` takes a named vector to replace ugly variable names with readable ones, e.g., `c("educ_years" = "Education (years)")`. (2) `gof_map` lets you choose which fit statistics appear at the bottom (e.g., drop AIC/BIC, keep N and R^2). (3) You can pass a named list of models to compare specifications side by side. (4) `output = "latex"` produces a LaTeX table you can paste directly into a paper. These are the things that save you from spending hours formatting tables manually.

Coefficient plots with `modelplot()`

- Visual alternative to coefficient tables
- `modelplot(model)`
  - → Point = estimate, line = 95% CI
  - → Vertical line at zero = "no effect"
- When are coefficient plots better than tables?
  - → Many predictors
  - → Comparing across model specifications
  - → Audience that finds tables intimidating
- `modelplot(list("Model 1" = m1, "Model 2" = m2))`

Coefficient plots (also called forest plots or dot-and-whisker plots) show the same information as a regression table, but in visual form. The key advantage: you can immediately see which coefficients are statistically different from zero (their CI doesn't cross the vertical line), and you can compare magnitudes visually. The key limitation: they work well when variables are on comparable scales, but can be misleading when one variable has a tiny coefficient and another has a huge one. Remind students that `modelplot()` returns a ggplot2 object, so you can customize it with `+ theme()`, `+ labs()`, etc.

Prediction plots: the gold standard



- Shows the **full relationship**, not just one number
- Any audience can read it
- `plot_predictions(model, condition = "x")`

This schematic illustrates what a prediction plot looks like. The line shows the predicted value of Y across values of X, and the shaded band shows the uncertainty. This is far more informative than a single coefficient because it shows the relationship over the entire range of the data. The rug marks at the bottom show where the actual data points are, which helps the reader gauge where the predictions are well-supported and where they're extrapolating. Emphasize that `plot_predictions()` does all of this automatically, including computing the confidence intervals using the delta method.

Tables vs. plots: when to use which

	Table	Plot
Exact numbers needed	✓	
Many models side by side	✓	
Conveying one key relationship		✓
Non-specialist audience		✓
Interactions / non-linearities		✓
Appendix / robustness checks	✓	

- Best practice: prediction plots in the main text, tables in the appendix
- Both are easy to produce with `modelsummary` and `marginaleffects`

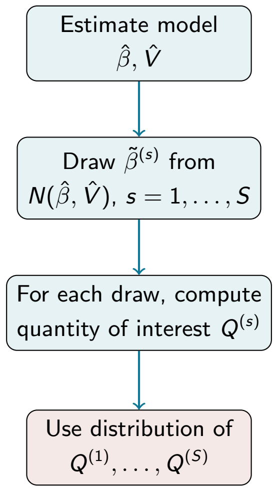
This is practical advice for writing papers and presentations. The trend in top journals is toward more visualization and fewer dense tables. A prediction plot in the main text, accompanied by a full regression table in the appendix, is the current best practice. For a presentation or policy brief, prediction plots are almost always better than tables. For a journal article, you typically need both: plots for the main results, tables for the full specification details. Coefficient tables are still necessary for transparency and replication, but they shouldn't be the primary way you communicate your findings.

Simulation-based uncertainty

- All quantities of interest have **uncertainty**
- Standard approach: delta method (what marginaleffects uses)
- Alternative: **simulation**
  - → Draw $\tilde{\beta}$ from $N(\hat{\beta}, \hat{V})$ many times
  - → Compute the quantity of interest for each draw
  - → Use the distribution of results as the uncertainty estimate
- Why simulate?
  - → Works for any quantity, no matter how complex
  - → Easy to combine multiple quantities
  - → Confidence intervals are automatic

The simulation approach is conceptually simple: (1) your estimated coefficients have a sampling distribution, approximately $N(\hat{\beta}, \hat{V})$; (2) draw many sets of coefficients from this distribution; (3) for each draw, compute whatever quantity you care about (predicted value, marginal effect, first difference, etc.); (4) the distribution of computed quantities gives you the uncertainty. This is the approach advocated by King, Tomz, and Wittenberg (2000) and implemented in the original Clarify package for Stata. The marginaleffects package uses the delta method by default, which is faster and usually equivalent, but simulation is more flexible for complex quantities.

Simulation: the logic


Estimate model $\hat{\beta}, \hat{V}$
Draw $\tilde{\beta}^{(s)}$ from $N(\hat{\beta}, \hat{V})$, $s = 1, \ldots, S$
For each draw, compute quantity of interest $Q^{(s)}$
Use distribution of $Q^{(1)}, \ldots, Q^{(S)}$

```
m = lm(y ~ x, data = df)

sim_betas = MASS::mvrnorm(1000, coef(m), vcov(m))

sim_pred = sim_betas %*% x_scenario

quantile(sim_pred, c(0.025, 0.975))
```

Walk through the four steps with the R code on the right. Step 1: estimate the model as usual. Step 2: draw many sets of coefficients from the multivariate normal defined by the estimated coefficients and variance-covariance matrix. `MASS::mvrnorm()` does this. Step 3: for each drawn set of coefficients, compute whatever quantity you care about — here, a predicted value for a specific scenario. Step 4: the 2.5th and 97.5th percentiles of the simulated quantities give you a 95% confidence interval. This is equivalent to the delta method for simple quantities but extends to arbitrarily complex ones. For most practical purposes, the marginaleffects package handles uncertainty automatically, but understanding the simulation approach helps you build intuition about what confidence intervals actually mean.

Worked example: predicted values with uncertainty

```
m = lm(income ~ age + education + female, data = df)

# Using marginaleffects (delta method)
predictions(m, newdata = datagrid(age = 40, education = 16,
female = 1))

# Using simulation
library(MASS)
sim_b = mvrnorm(1000, coef(m), vcov(m))
x = c(1, 40, 16, 1) # intercept, age, educ, female
sim_pred = sim_b %*% x
quantile(sim_pred, c(0.025, 0.5, 0.975))
```

- Both approaches give (nearly) identical results

This worked example shows both approaches side by side. The marginaleffects approach is much shorter and should be the default in practice. The simulation approach is shown for pedagogical purposes: it makes explicit what is happening "under the hood" when we compute standard errors for predicted values. The two approaches will give nearly identical results for linear models and simple quantities. For complex nonlinear quantities (e.g., the ratio of two predicted values), the simulation approach may be more reliable because the delta method relies on a linear approximation. In practice, use marginaleffects; understand simulation so you know what the numbers mean.

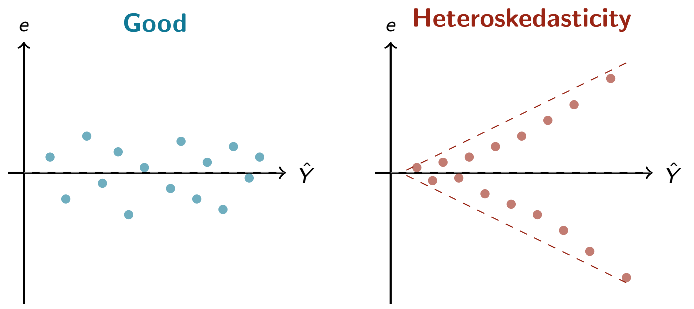
Roadmap

What can go wrong?

- OLS assumes:
  - → Linear relationship between X and $E[Y]$
  - → Constant error variance (**homoskedasticity**)
  - → No extreme outliers driving results
- When these fail:
  - → Coefficients may be biased (non-linearity)
  - → Standard errors may be wrong (heteroskedasticity)
  - → Results may be driven by a few observations (outliers)
- How do we check? **Residual diagnostics**

This section covers the basics of regression diagnostics. We're not going deep into the theory — the goal is practical awareness. Students should know (1) what to check, (2) how to spot problems, and (3) what to do about them. The three issues we focus on are heteroskedasticity (wrong SEs), non-linearity (biased coefficients), and influential observations (fragile results). All three can be diagnosed visually using residual plots.

Residuals vs. fitted values



- In R: `plot(model, which = 1)`
- Look for patterns: funnel shape, curves, clusters

The residuals vs. fitted values plot is the single most useful diagnostic. On the left: what you want to see — a random cloud of points with no discernible pattern, centered around zero. This means the model's assumptions are approximately satisfied. On the right: a funnel (or fan) shape, where the spread of residuals increases with the fitted value. This is the classic signature of heteroskedasticity: the error variance is not constant. In R, `plot(model, which = 1)` produces this automatically. Other patterns to watch for: a curved pattern (non-linearity) or a cluster of points far from the rest (outliers).

Heteroskedasticity

- **Heteroskedasticity**: Var($\varepsilon \mid X$) is not constant
- Consequences:
  - → Coefficients are still unbiased
  - → But standard errors are **wrong**
  - → So confidence intervals and p-values are unreliable
- Very common in practice:
  - → Income models (richer countries, more variance)
  - → Any outcome with natural bounds
  - → Binary outcomes (always heteroskedastic, as in LPM)

Heteroskedasticity is probably the most common violation of OLS assumptions in social science data. The word means “different variance” (hetero = different, skedastic = scatter). The key practical consequence: your coefficients are fine, but your standard errors are wrong. This means you might think an effect is significant when it isn't, or vice versa. The good news: there's a simple fix. The bad news: the standard `lm()` function doesn't account for it by default. Common examples: income varies more in rich countries than poor ones; test scores vary more in heterogeneous classrooms; any outcome that is bounded (like hours worked) will have compressed variance near the bounds.

Robust standard errors

- The fix: **heteroskedasticity-consistent (HC) standard errors**
- Also called "robust" or "White" standard errors
- In R (multiple options):
  - → `modelsummary(model, vcov = "robust")`
  - → `modelsummary(model, vcov = "HC3")`
  - → `estimatr::lm_robust(y ~ x, data = df)`
- Best practice: **always use robust SEs**
  - → If homoskedastic: robust SEs ≈ classical SEs
  - → If heteroskedastic: robust SEs are correct, classical are not

Robust standard errors are the standard solution to heteroskedasticity. The idea: instead of assuming constant variance, estimate the variance of each residual separately and use those to construct the standard errors. HC3 is the recommended variant for small samples (it's the default in `estimatr::lm_robust()`). The "always use robust SEs" advice is standard in economics and increasingly common in political science. The logic: if homoskedasticity holds, robust and classical SEs will be nearly identical, so you lose nothing. If heteroskedasticity is present, robust SEs are correct and classical are not. It's an insurance policy with no cost.

Log transformations: when and why

- Many variables are right-skewed:
  - → Income, GDP, population, firm size, casualties
- Taking $\log(X)$ compresses the right tail
- Makes the distribution more symmetric
- Can reduce heteroskedasticity
- But interpretation changes!

Log transformations are one of the most common data transformations in applied work. Any variable that is strictly positive and right-skewed is a candidate. The log compresses large values and spreads out small values, which often makes the relationship with the outcome more linear and reduces heteroskedasticity. But the trade-off is that interpretation becomes more complex: the coefficient is no longer a simple "one unit change" effect. We need to think in terms of percentage changes. Remind students: always use the natural log (ln), not log base 10. In R, `log()` is the natural log by default.

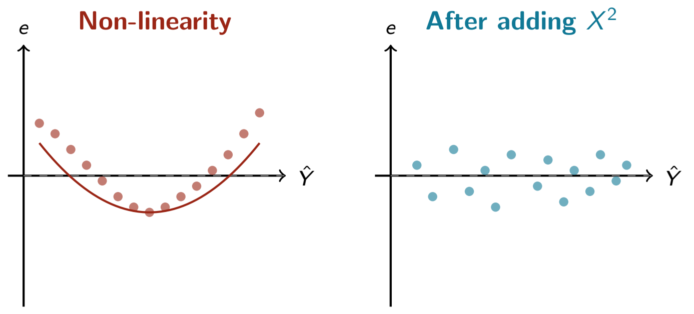
Interpreting logs

Model	Equation	Interpretation of β_1
Level-level	$Y = \beta_0 + \beta_1 X$	$\Delta X = 1 \Rightarrow \Delta Y = \beta_1$
Log-level	$\log(Y) = \beta_0 + \beta_1 X$	$\Delta X = 1 \Rightarrow \%\Delta Y \approx 100 \cdot \beta_1$
Level-log	$Y = \beta_0 + \beta_1 \log(X)$	$\%\Delta X = 1 \Rightarrow \Delta Y \approx \beta_1/100$
Log-log	$\log(Y) = \beta_0 + \beta_1 \log(X)$	$\%\Delta X = 1 \Rightarrow \%\Delta Y \approx \beta_1$

- Log-log: β_1 is the **elasticity**
- Log-level: β_1 is a semi-elasticity
- Most common in practice: log-level and log-log

This table is the key reference for interpreting logged variables. Walk through each row. Level-level: the standard OLS interpretation. Log-level: a one-unit change in X is associated with a $100\beta_1\%$ change in Y. For example, if $\beta_1 = 0.05$, a one-unit increase in X is associated with a 5% increase in Y. Level-log: a 1% change in X is associated with a $\beta_1/100$ change in Y. Log-log: the elasticity — a 1% change in X is associated with a $\beta_1\%$ change in Y. The approximation $\log(1 + r) \approx r$ for small r is what makes these interpretations work. For large changes (say $\beta_1 > 0.2$), the exact formula $e^{\beta_1} - 1$ gives the percent change more accurately.

Residual plots: checking linearity



- A U-shape in residuals $\Rightarrow$ missing non-linearity
- Fixes: add X^2, use $\log(X)$, or use a more flexible model

Non-linearity shows up as a systematic curved pattern in the residuals. The left panel shows the classic U-shape: the model under-predicts at the extremes and over-predicts in the middle. This means the true relationship is not linear. Common fixes: (1) add a quadratic term (X^2), (2) log-transform X if it's right-skewed, (3) use a polynomial or spline. The right panel shows what the residuals look like after adding the quadratic — the pattern disappears. In R, `plot(model, which = 1)` shows this. The built-in smooth line (red) helps identify the pattern. If it's roughly flat, you're fine; if it curves, you need to address non-linearity.

Influential observations

- Some observations can disproportionately influence results
- **Cook's distance**: measures how much the model changes if you remove observation i
  - → In R: `plot(model, which = 4)`
  - → Observations with Cook's $D > 4/n$: investigate further
- What to do with influential observations?
  - → Do NOT automatically delete them
  - → Check if they are data errors
  - → Re-run the model without them as a robustness check
  - → Report both results

Cook's distance combines leverage (how unusual is the observation's X values?) and residual size (how far is the observation from the fitted line?) into a single measure. High Cook's distance means the observation is both unusual and poorly fitted, so it pulls the regression line toward it. The threshold $4/n$ is a common rule of thumb. Important: influential observations are not automatically "bad" — they might be the most interesting observations in your data. A country that is a large outlier might be the case that challenges the theory. The advice is to investigate, not to blindly delete. Re-running without the influential observations tells you whether your conclusions depend on them.

Roadmap

Summary: key takeaways

- Raw coefficients are rarely enough — compute **quantities of interest**
- Predicted values, marginal effects, and first differences are your tools
- The `marginaleffects` package handles everything

- Present results with plots (prediction plots, coefficient plots)
- Tables go in the appendix, plots in the main text

- Always check diagnostics: residual plots, heteroskedasticity
- Always use robust standard errors
- Use log transformations for skewed variables (but interpret carefully)

Recap the four main themes. (1) Interpretation: move beyond raw coefficients to predicted values, marginal effects, and first differences. (2) Presentation: use prediction plots as the primary communication tool, with tables for completeness. (3) Uncertainty: understand that all quantities have uncertainty, computed via the delta method or simulation. (4) Diagnostics: check residual plots for heteroskedasticity and non-linearity, use robust SEs as standard practice, and investigate influential observations. These skills apply to all the models we've covered so far (OLS and logit) and will carry forward to panel data and other models.

For next week

- Complete Assignment 4

- Next session: Best Practices in Computing
  - → Reproducible workflows
  - → Project organization
  - → Writing clean R code

Brief logistics slide. The assignment will ask students to apply the tools from today: compute and plot predicted values, marginal effects, and first differences for a model of their choice. They should also run basic diagnostics (residual plot, test for heteroskedasticity) and use robust standard errors. Next week shifts gears to computing best practices — project organization, reproducible workflows, and writing clean code. This is a practical session that will help them with their final projects.

Questions?