# Models in R

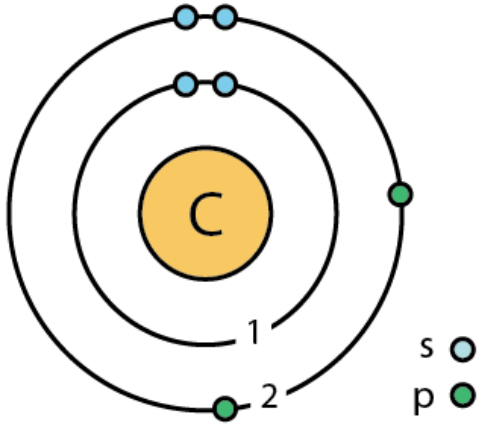
Christopher Wu

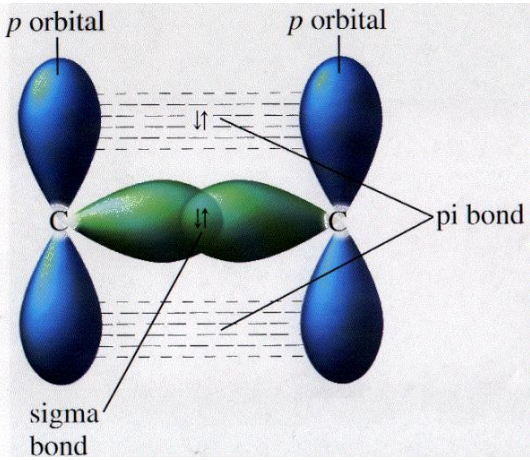
First.Last@yale.edu

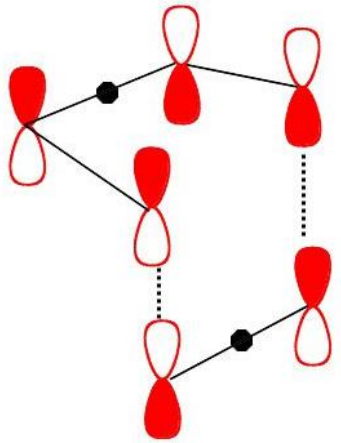
# (Word to the wise)

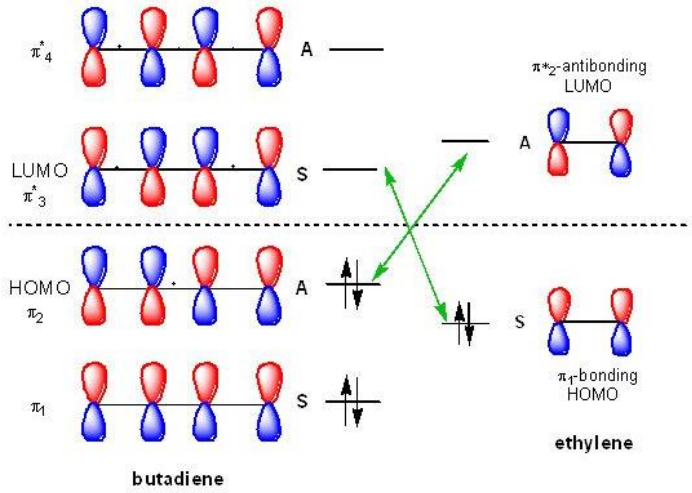
"All models are wrong, but some are useful." –George Box

Don't take prediction too seriously.





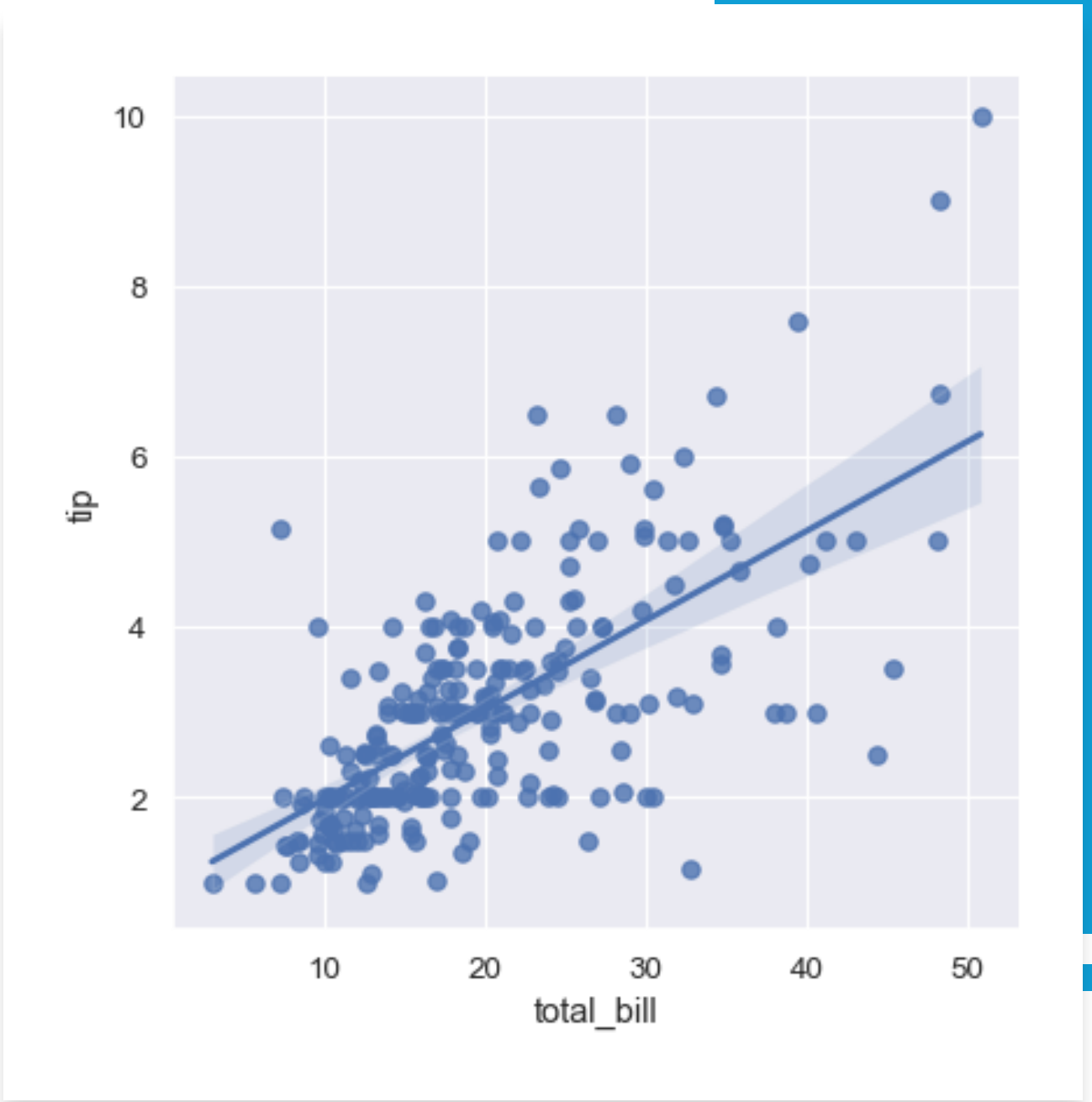
# Prediction basics

- You observe a sample of size $N$ of some outcome (a "dependent variable"): $Y_1, \ldots, Y_N$ (e.g. Income)
- You likewise observe a sample of size $N$ of some predictor (an "independent variable"): $X_1, \ldots, X_N$ (e.g. education)
- The goal of prediction is, given a new $X_{N+1}$, to make a guess $\hat{Y}_{N+1}$
- We denote these predictions with hats

# Prediction basics

- How do we do this? Well we assume that $Y = f(X) + \varepsilon$, where $\varepsilon$ is some random error which is 0 on average
- If we knew $f$, then our best guess would be $\hat{Y} = f(X)$
- But we don't know $f$. We have to guess $\hat{f}$. And then set $\hat{Y} = \hat{f}(X)$
- When we say we "train some model", we really mean we have some method of taking data and spitting out $\hat{f}$

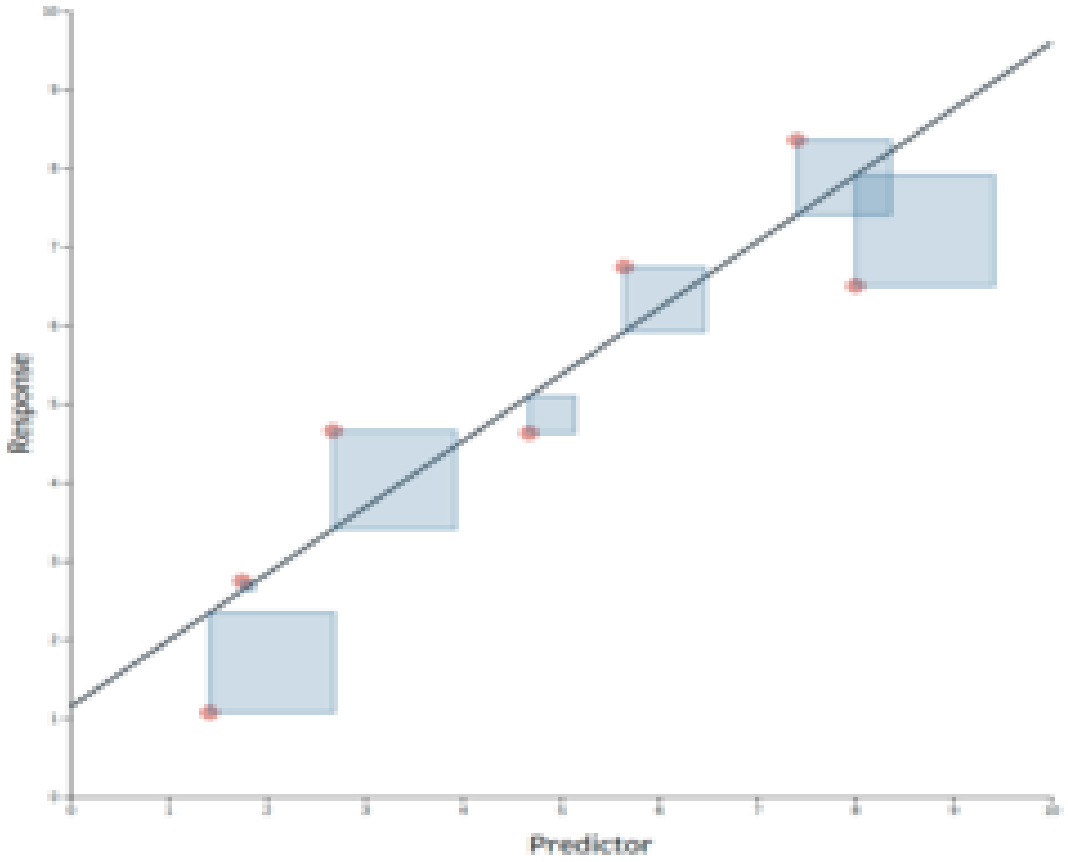
# Part 1: Least Squares



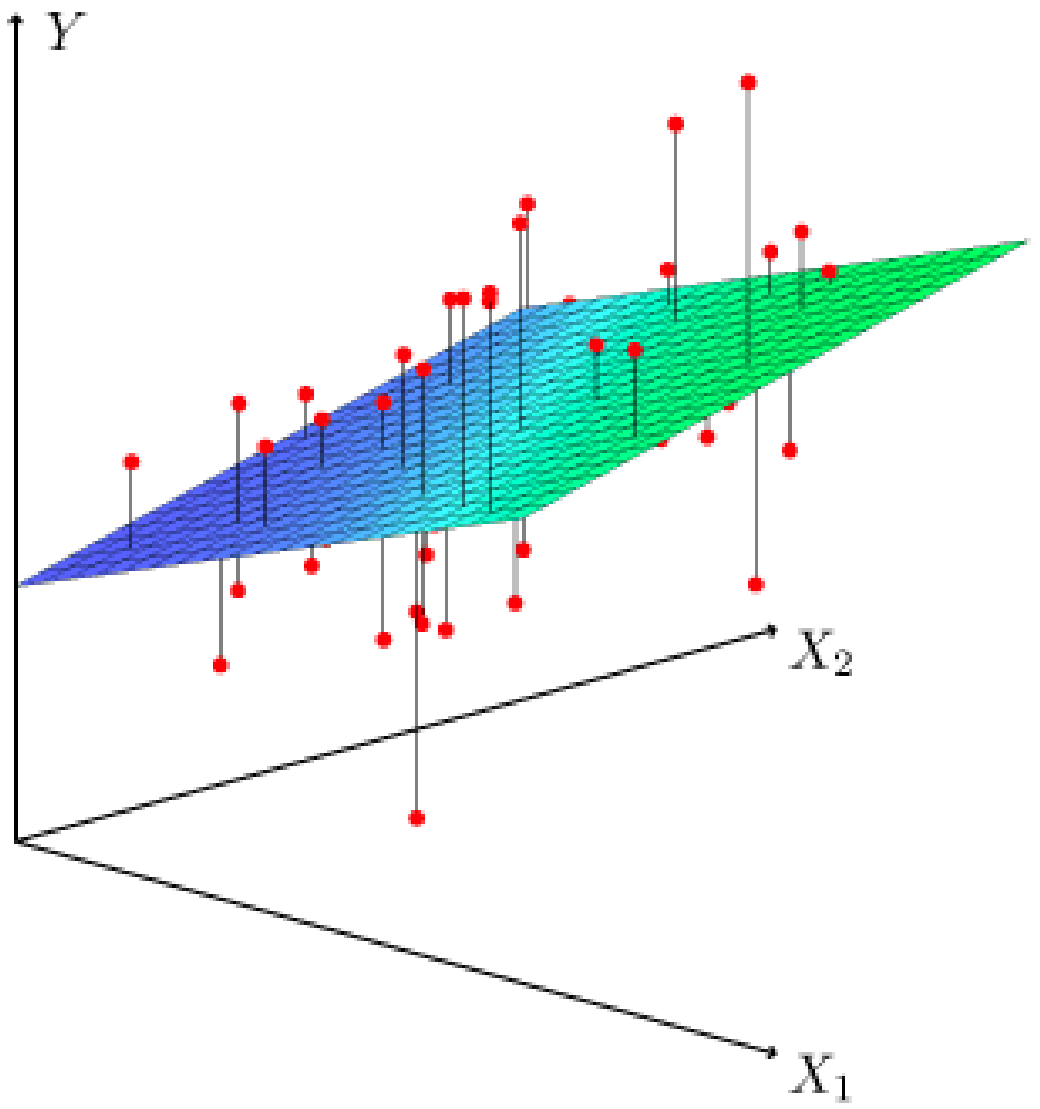
# Ordinary Least Squares

- Guessing $f$ is hard. "Parametric" models assume a functional form (e.g. linear regressions assume $f$ is linear)
- OLS conjectures that $Y = f(X) + \varepsilon = \beta_0 + \beta_1 X + \varepsilon$
- Now we just have to guess $\beta_0, \beta_1$
- OLS fits a straight line that minimizes the squared prediction errors
- Predicted outcome: $\hat{Y} = \widehat{\beta_0} + \widehat{\beta_1} X$
- Actual outcome: $Y_i$
- Error: $\widehat{Y_i} - Y_i$
- OLS chooses $\widehat{\beta_0}, \widehat{\beta_1}$ such that $\left(\widehat{Y_1} - Y_1\right)^2 + \cdots + \left(\widehat{Y_N} - Y_N\right)^2$ is as small as possible

# Ordinary Least Squares

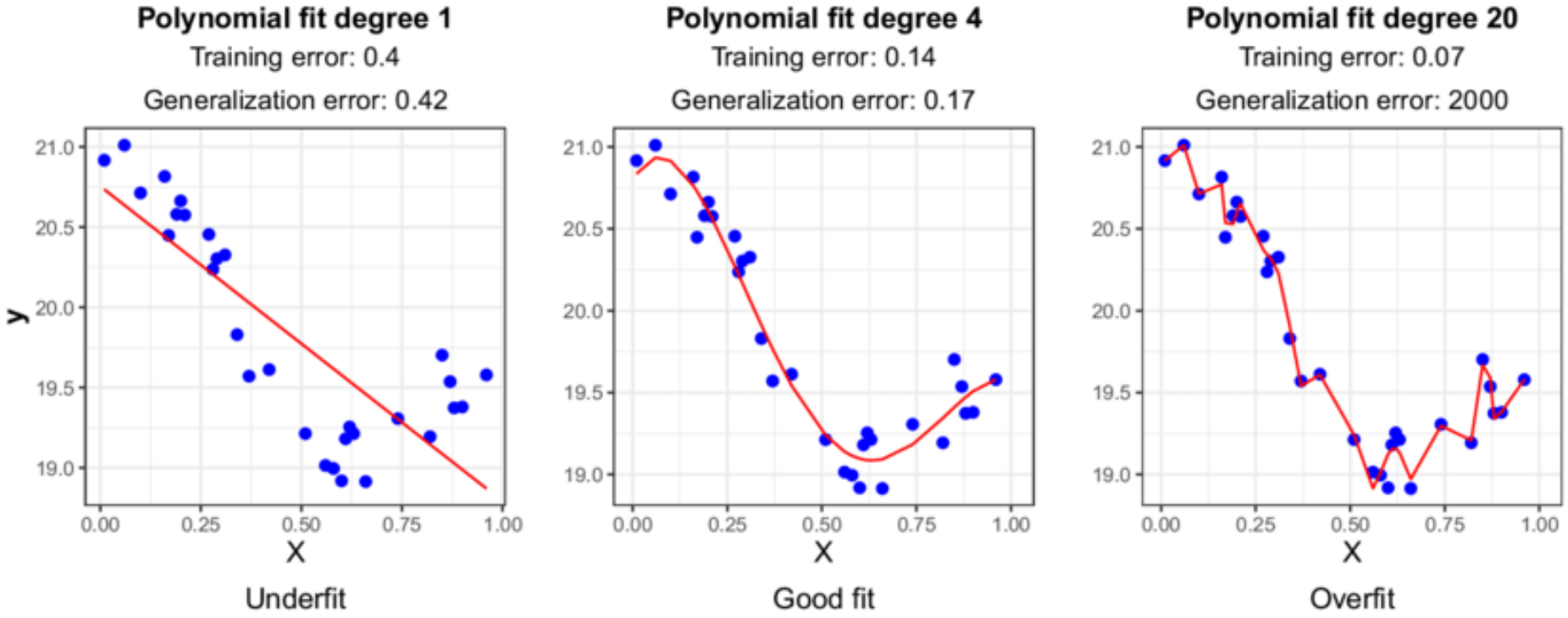


# OLS with many predictors



# Multivariate OLS

- Basically the same thing, but $X$ is now a vector $X = (X_1, \ldots, X_p)$
- Notational note: $p$ is the number of predictor variables, $N$ is the sample size. Here the subscripts are indexing the predictors
- OLS conjectures that $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \cdots + \beta_p X_p + \varepsilon$
- E.g. $Y$ = democratic vote share; $X_1$ = Evangelical population; $X_2$ = black population; $X_3$ = 1 if urban and 0 if rural; $X_4$ = whether it rained on election day…
- Again, the algorithm finds $\widehat{\beta_0}, \widehat{\beta_1}, \ldots$ that minimizes square error
- Some basic issues: number of variables should be much smaller than sample size ($p < N$), variables should not be linearly related to each other (“collinearity”)



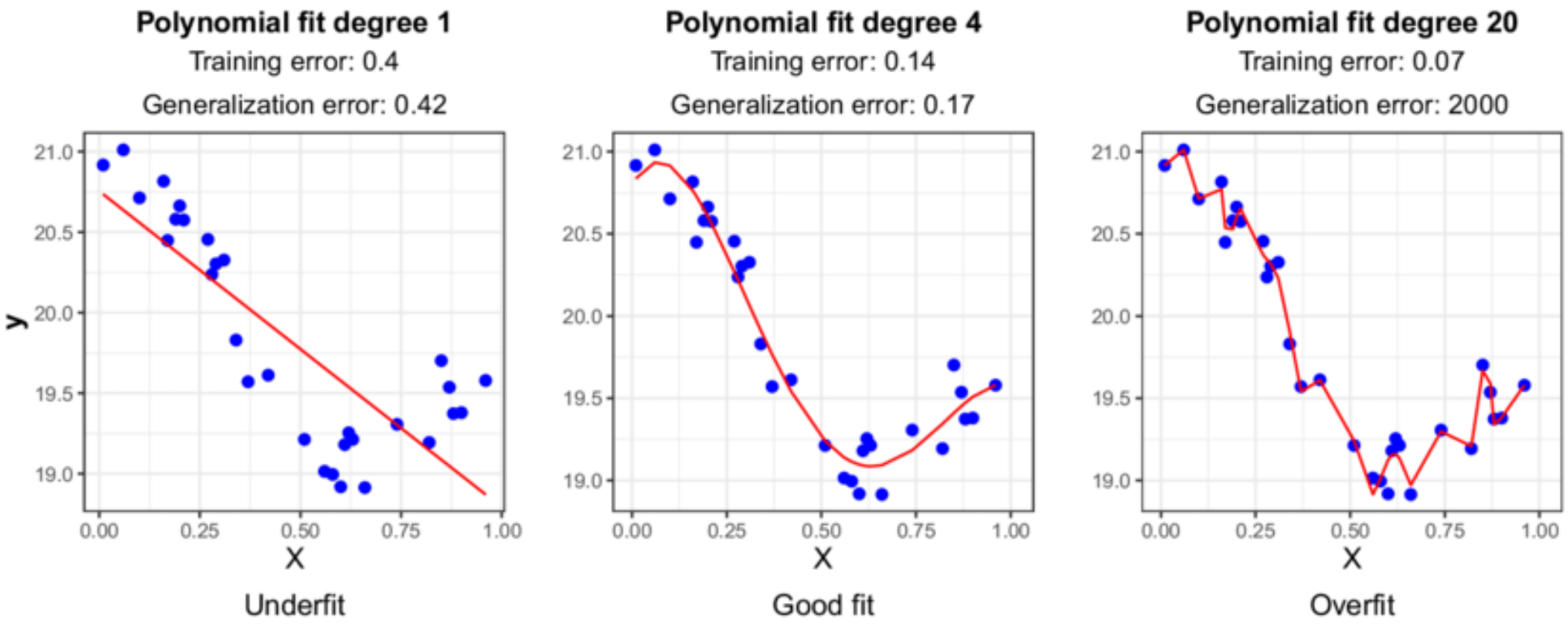

# Polynomial Least Squares

- Fun math fact: you can approximate continuous functions very well with polynomials
- e.g. $Income = \beta_0 + \beta_1 educ + \beta_2 educ^2 + \beta_3 educ^3 + \varepsilon$
- i.e. take 'powers of predictors' as predictors themselves
- The more terms you add, the better the fit

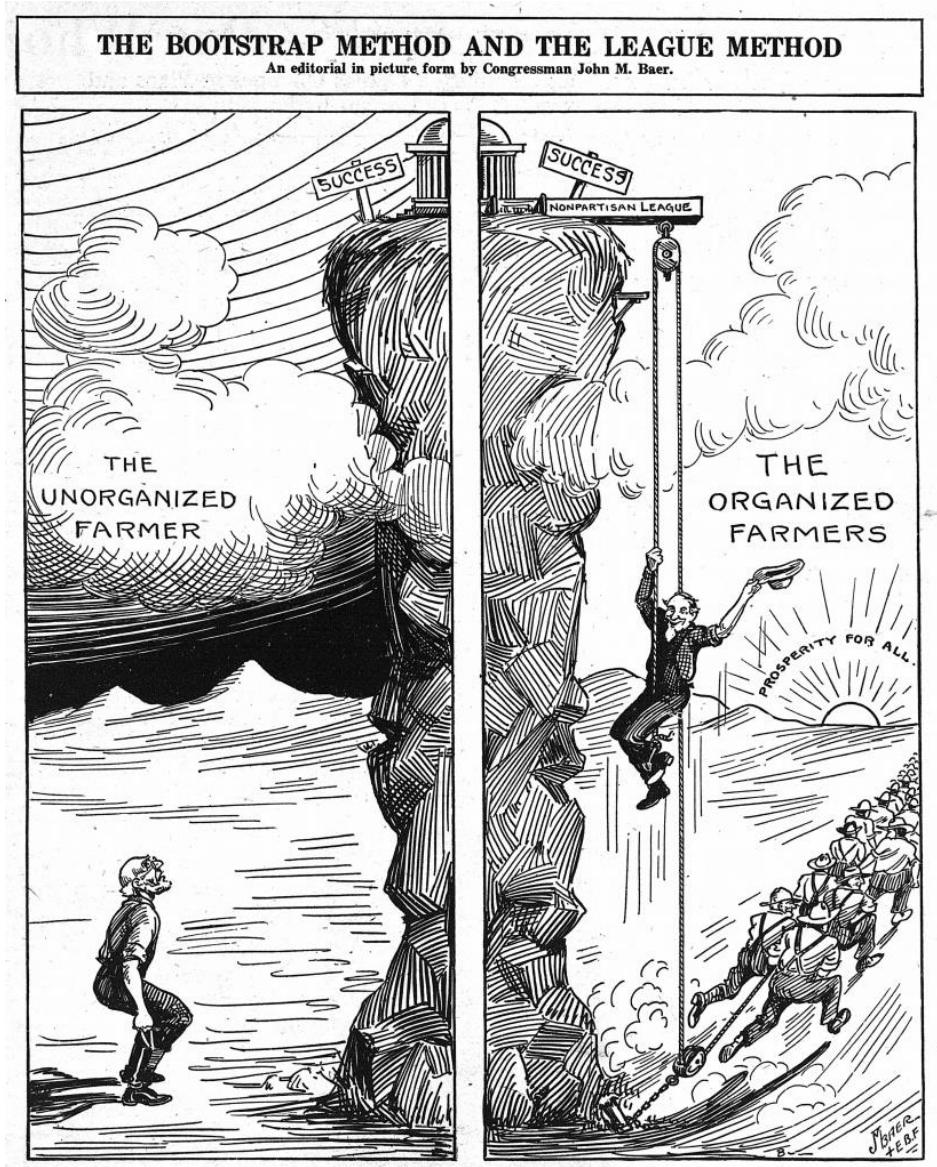
# So why not just throw the kitchen sink?

- R will be very mad at you if you have too many predictors. Things will stop working
- Too many predictors -> tendency for model predictions to change drastically when given a new dataset
- Too much squiggliness -> tendency to overreact to small changes in $X$
- “Bias variance tradeoff”
- “Overfitting”

# Part 2: Dealing with Overfitting

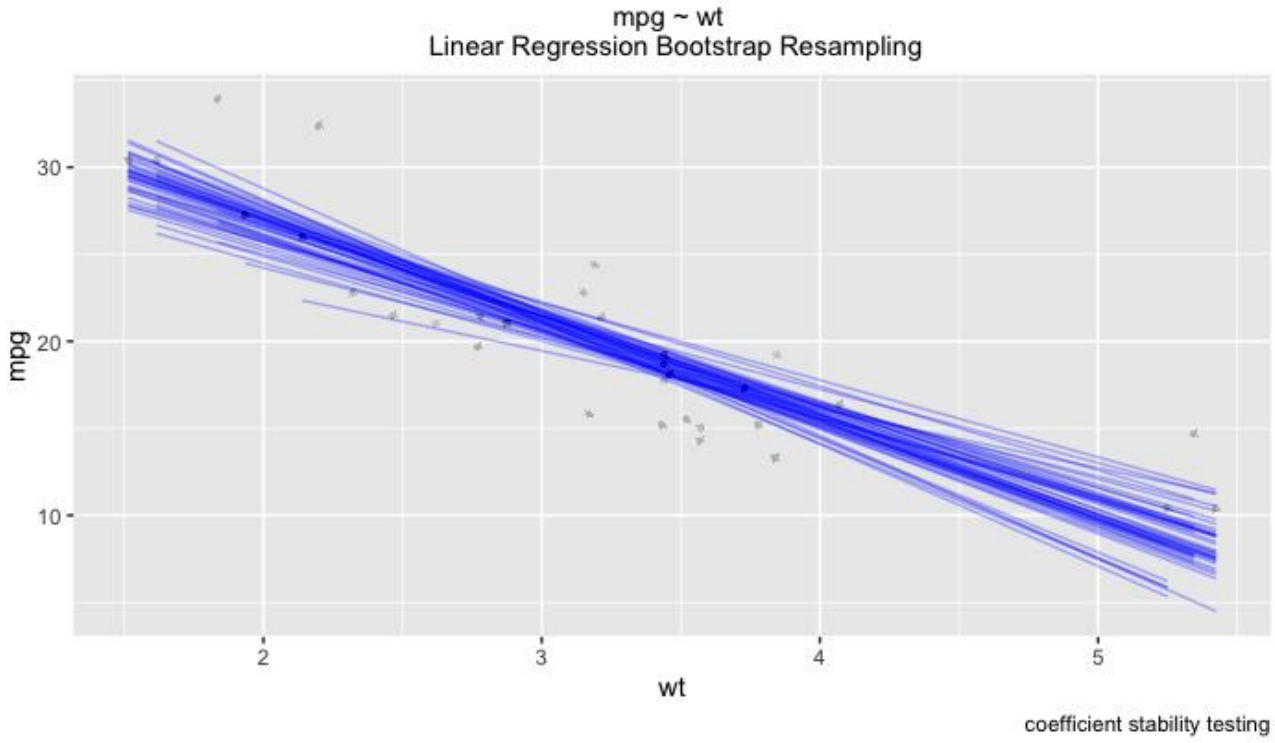


# Diagnosing the problem: pulling yourself up by the bootstraps



# Bootstrap

- Suppose you fit a model. How do you know how different the fit would be if the data had been different?
- “Bootstrapping” refers to figuring this out by randomly taking out come of the data, using the same modeling procedure, and seeing how much the model fit fluctuates



# Sample splitting: training-validation-test

- **You should do this basically always if you have enough data**
- Take your data sample, randomly split it (say, 60-20-20)
- Train your model (i.e. figure out how many variables to include, how to "fit" the data, what $\lambda$ to choose) on the 60%
- Figure out how well you did on the 20%
- Then, at the very very end when you are choosing from a handful of good models, test on the last 20%
- Different people do this differently. e.g. some do training-test split (often 70-30 split)
- The idea is the same: if you want to know how externally valid your model is, you want to set some data aside

# Ways to deal with overfitting

- Selecting $X$'s carefully: fit a bunch of models, then balance the tradeoff between fit and how "complicated" the model is
- Regularization: instead of minimizing squared errors, we minimize squared errors plus some "penalty" for how complicated the model is
- Sample splitting
- Repeating/combining some of the above

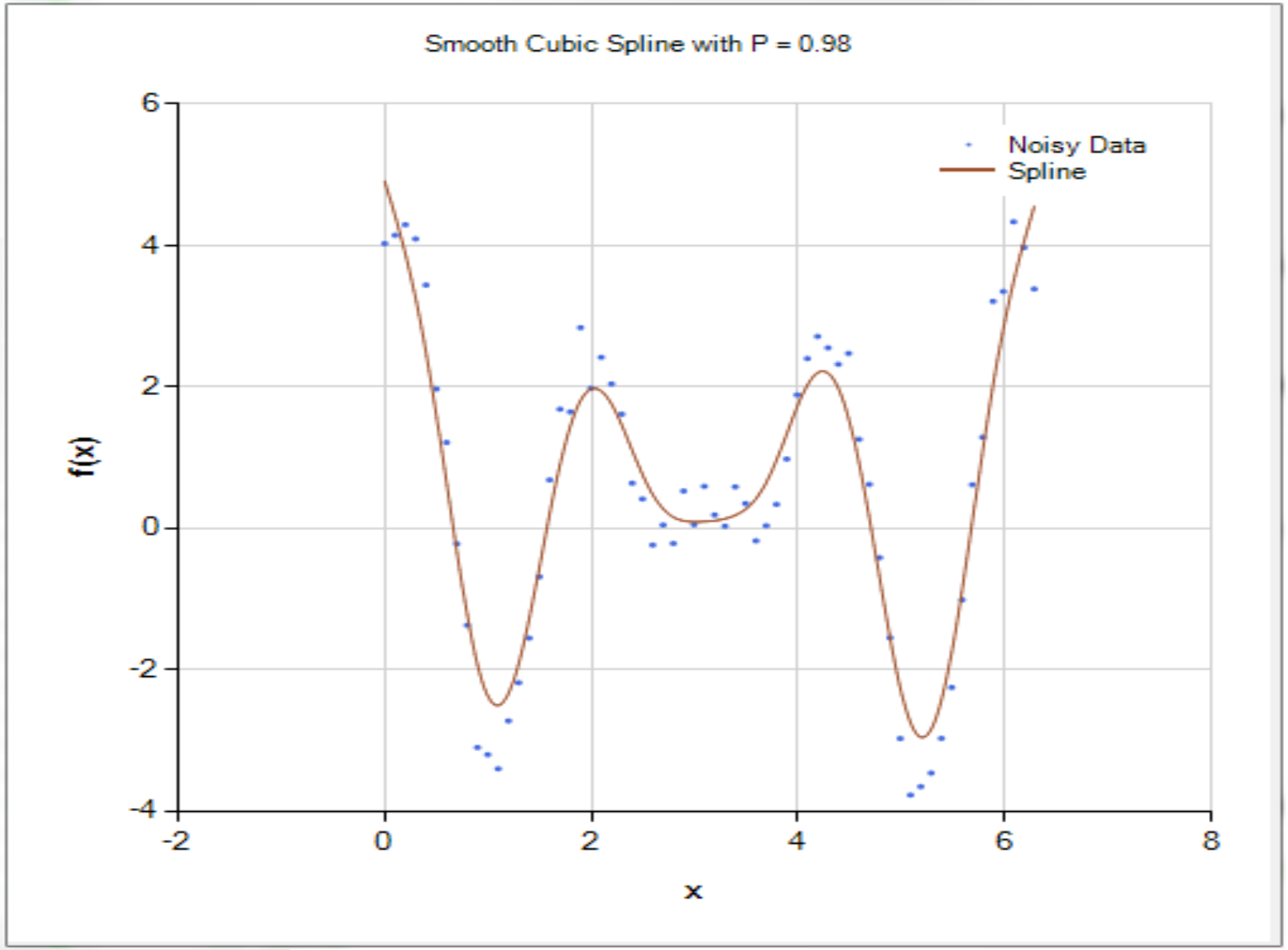
# AIC

- A metric which balances the tradeoff between model fit and how complicated your model is (for example, based on how many $X$ variables in your model)
- Run a bunch of models
- Calculate their AICs
- Choose the one with the smallest AIC
- Other metrics available: e.g. BIC, adjusted $R^2$, Mallow's $C_p$

# Smoothing spline

- Say we have just $Y$ and a scalar $X$
- Like OLS, but adds a penalty for $\hat{f}$ being too squiggly
- Formally, we choose $\hat{f}$ to minimize $\sum_i \left(Y_i - \hat{f}(X_i)\right)^2 + \lambda \int \hat{f}''(x)dx$
- You get to choose $\lambda$. The higher $\lambda$, the higher the penalty
- This also generalizes to a vector $X$

# Smoothing spline



# Shrinkage

- You can also reduce overfitting in linear regressions by "shrinking" the $\beta$'s
- Shrinkage methods automatically penalize models where $\hat{\beta}$'s are too large
- One particularly cool shrinkage technique is called LASSO, which is given by minimizing: $\sum_i (\hat{Y} - Y)^2 - \lambda(|\beta_1| + \cdots + |\beta_p|)$
- LASSO is especially useful when you have a lot of independent variables because LASSO turns most of the $\hat{\beta}$'s into zero

# Cross-Validation

- Kind of like repeated sample splitting
- Suppose you have a training set and a test set (70-30, say)
- You split the training sample randomly into $k$ groups (called "folds")
- For each fold, you use that as the validation set and the remaining data as your training set
- So you do this $k$ times and average over the validation errors
- Could be 10-fold, 20-fold, "leave-one-out"

# Part 3: Some more models to play around with

# Autoregressive Models

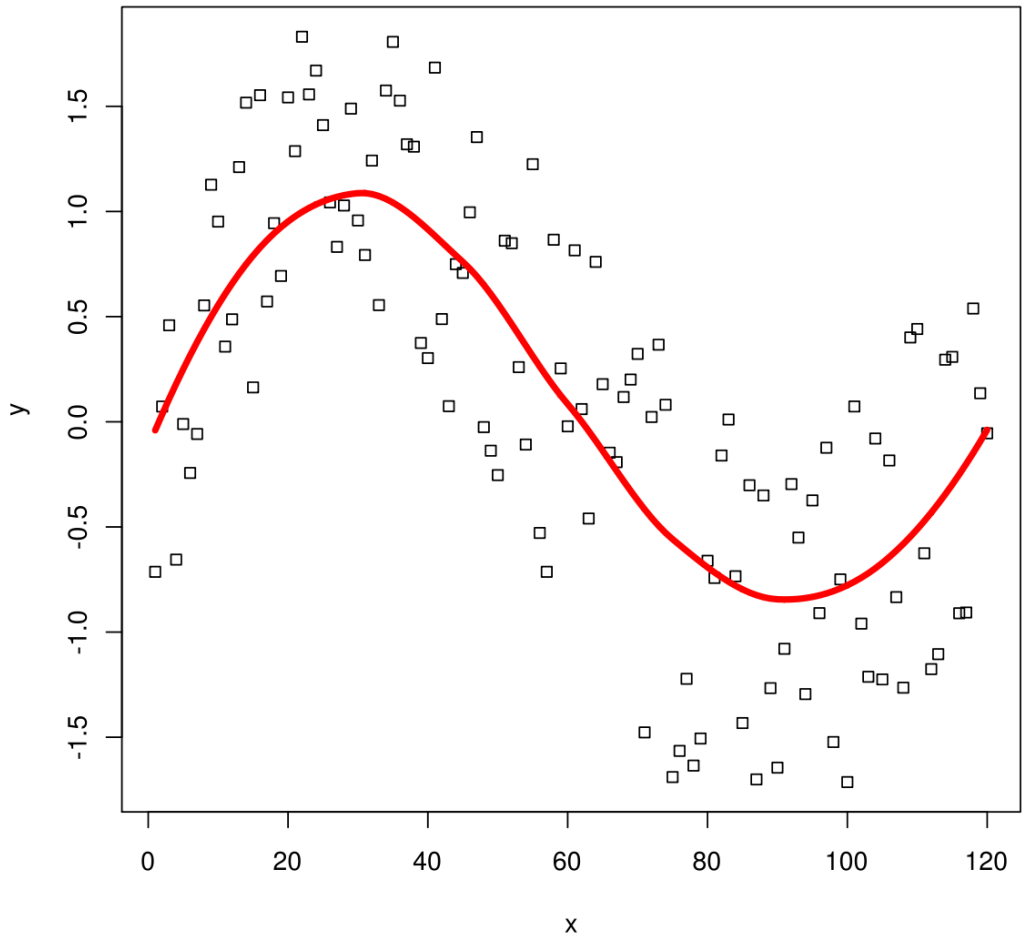
- Time series data
- AR(1) model conjectures that today's outcome is linear in yesterday's outcome plus an error
- $Y_t = \beta Y_{t-1} + \varepsilon$
- More generally, an AR(p) process conjectures that the data is generated by $Y_t = \beta_1 Y_{t-1} + \cdots + \beta_p Y_{t-p} + \varepsilon$
- There is a spatial analog of this model (for 2-D spatial data, rather than 1-D time series data) called the spatial AR model

# Moving Average

- Time series data
- MA(q) model conjectures that the data is some linear combination of today's error and the last q errors
- $Y_t = \varepsilon_t + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \cdots + \theta_q \varepsilon_{t-q}$
- Combine AR(p) and MA(q) and you have the ARMA(p,q) model

# Local smoothing

- If you go on Wikipedia and look at polling data, it's probably smoothed-over using some smoothing technique
- Basic idea: the curve is a local average of the datapoints around it, with more weight given to closer points
- Many ways to weight, no right answer



# End

- Lots of models
- No right answers with prediction
- Just don’t overfit
- A lot of room for qualitative input
- Have fun