Additive

separate non-linear terms are combined by addition

univariate

each non-linear term uses only one predictor

non-linearity

can be fit using various methods we've already learned

GAM: Generalized Additive Model

Additive modeling assumption

- **Linearity** assumption: each predictor has a *coefficient*

$$g(\mathbb{E}[\mathbf{y}|\mathbf{X}]) = \beta_0 + \beta_1\mathbf{x}_1 + \beta_2\mathbf{x}_2 + \cdots + \beta_p\mathbf{x}_p$$

- **Additivity** assumption: each predictor has a *function*

$$g(\mathbb{E}[\mathbf{y}|\mathbf{X}]) = \beta_0 + f_1(\mathbf{x}_1) + f_2(\mathbf{x}_2) + \cdots + f_p(\mathbf{x}_p)$$

Includes linear models as special case if $f_j(\mathbf{x}_j) = \beta_j\mathbf{x}_j$

Assumptions / modeling choices:

- Assume f_j is in some function space / fit with some method
- e.g. global polynomial, `loess`, local/kernel regression, smoothing splines, etc--pick your favorite!
- Can use same/different methods for each predictor

Non-linear regression

Other times it's less clear, based on noise level and sample size

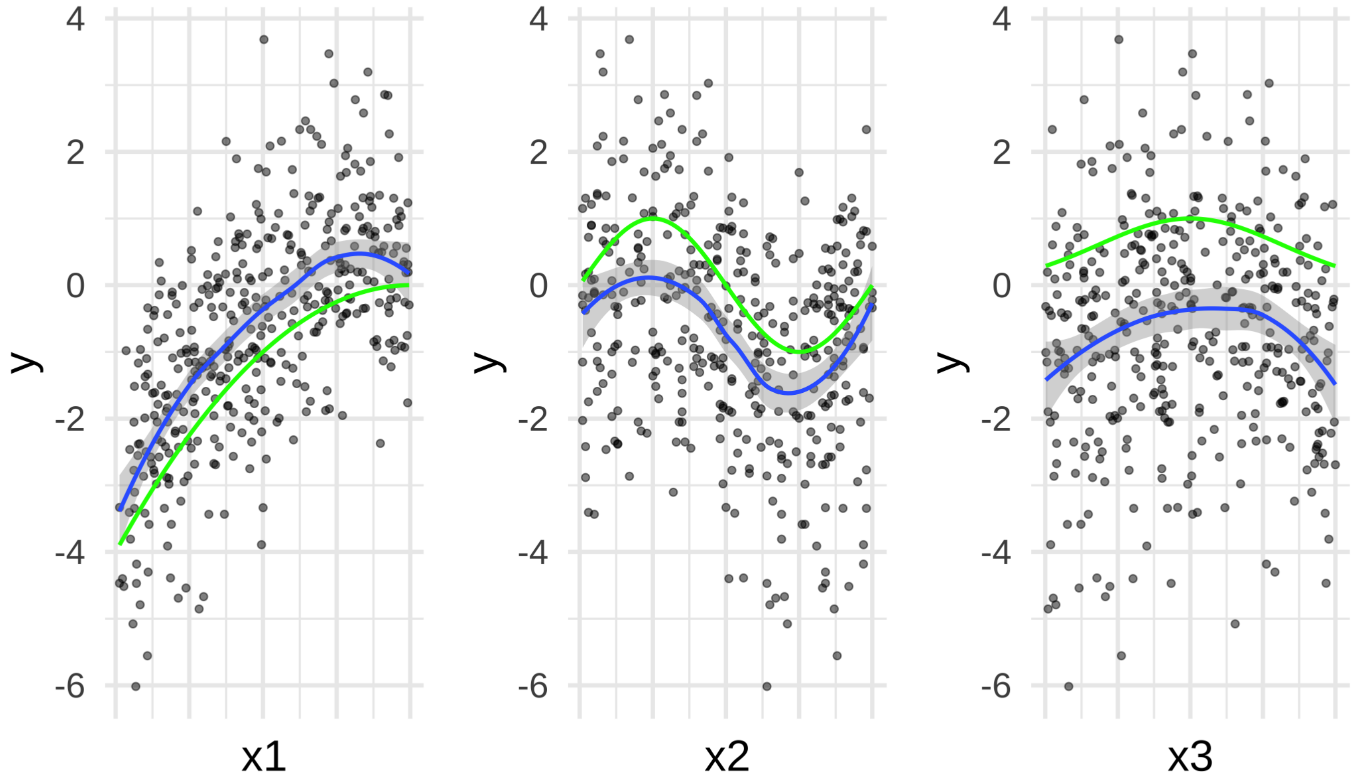
```r
f1 <- function(x) -1 + 2*x - x^2
f2 <- function(x) sin(pi*x)
f3 <- function(x) exp(-5*(x-1/2)^2)

set.seed(1)
n <- 400
df <- data.frame(
  x1 = 2*(runif(n)-1/2),
  x2 = sample(1:100 / 50,  n, replace = TRUE),
  x3 = runif(n)
) %>%
  mutate(
    y = f1(x1) + f2(x2) + f3(x3) + rnorm(n)
)
```

Univariate plots

```
uni_plot <- function(j) {
  xj <- paste0("x", j)
  fj <- paste0("f", j)
  ggplot(df, aes(get(xj), y)) +
    geom_point(alpha = .5) +
    geom_smooth() + xlab(xj) +
    geom_function(fun = get(fj),
                  size = 1,
                  color = "green") +
    theme(axis.text.x=element_blank(),
          axis.ticks.x=element_blank())
}
p1 <- uni_plot(1)
p2 <- uni_plot(2)
p3 <- uni_plot(3)
```

Side by side plots by adding with the `patchwork` library

```r
library(patchwork)
p1 + p2 + p3
```



Bias? Why? 😲

The true model is additive

We plot each variable separately but the `loess` curves are biased...

To fit $\hat{f}_1$ we would *ideally* do `loess` on

$$y - f_2(\mathbf{x}_2) - f_3(\mathbf{x}_3)$$

But we don't know f_2 and f_3, we are trying to estimate them too!

Backfitting algorithm

1. Start with some initial estimates $\hat{f}_j$, e.g. from `y ~ x_j`

2. Iterate over j, updating $\hat{f}_j$ by fitting `r_j ~ x_j` where the partial residual $\mathbf{r}_j$

$$\mathbf{r}_j = \mathbf{y} - \hat{\beta}_0 - \sum_{k \neq j} \hat{f}_k(\mathbf{x}_k)$$

   is computed using the current fits for all the other predictors

3. Repeat until "convergence" (some stopping rule)

Can additivity/GAMs be *importantly wrong*?

Interpretation: think carefully about **calculus** and **causality**. To simplify let's consider the identity link function (rather than e.g. logistic regression, those cases are more complicated)

Calculus

Does the CEF really decompose into additive terms? Is this approximation good:

$$\frac{\partial}{\partial x_j}\mathbb{E}[Y|\mathbf{X}] \approx g(x_j)$$

Or does the relationship between the average of Y and x_j vary depending on the value of another predictor x_k?

Can additivity/GAMs be *importantly wrong*?

Interpretation: think carefully about **calculus** and **causality**. To simplify let's consider the identity link function (rather than e.g. logistic regression, those cases are more complicated)

Causality

First, remember that causality is separate from prediction

But also, it may be a reason for doubting additivity

For example, if X_k is a cause of X_j, or if they have a common cause, then we may want to include an interaction term for them

```r
library(ggplot2movies)
df <- movies %>%
  filter(length <= 200, length > 10,
         year > 1918, votes >= 5) #, Short != 1)
```

I asked on Twitter what was missing from the plot of movie length vs movie rating and Thomas Lumley suggested confounding by **year**

Additive combination of non-linear predictors

```r
library(gam)
fit_gam_loess <-
  gam(rating ~ lo(length) + lo(year), data = df)
```

`lo` is for `loess`, but can use different methods too

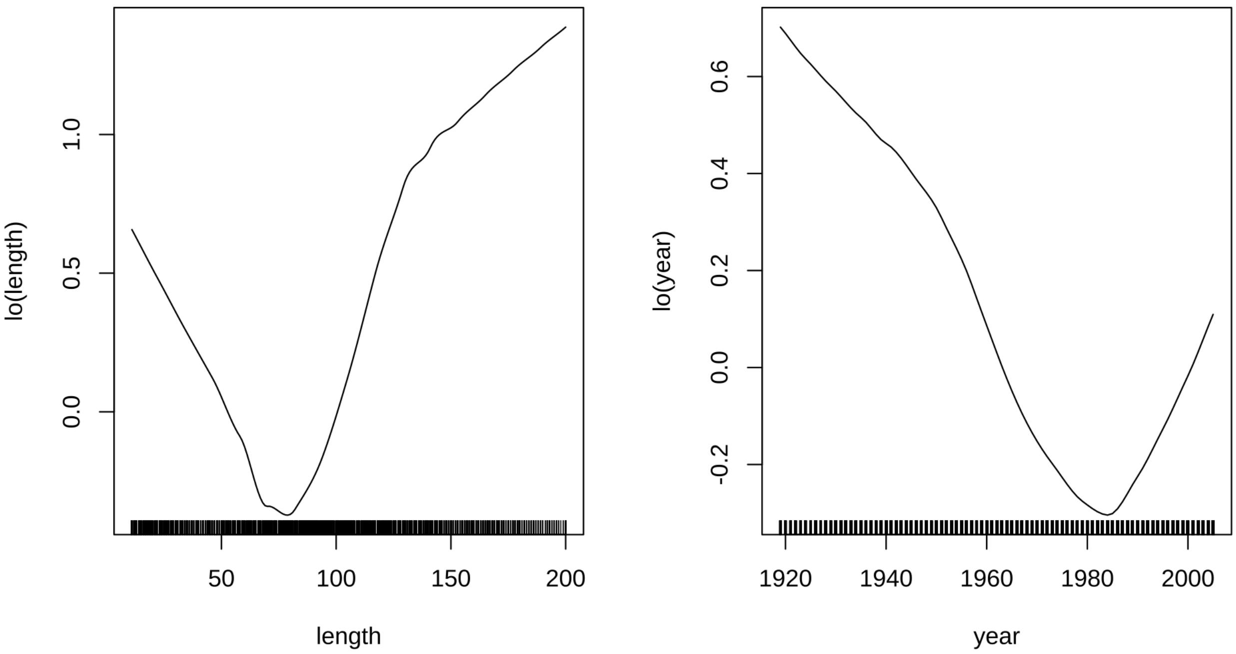
```r
tidy(fit_gam_loess)
```

```
## # A tibble: 3 × 6
##   term           df    sumsq  meansq statistic    p.value
##   <chr>       <dbl>    <dbl>   <dbl>     <dbl>      <dbl>
## 1 lo(length)      1     190.   190.       86.8  1.23e- 20
## 2 lo(year)        1    1561. 1561.       715.   2.12e-156
## 3 Residuals  53380. 116623.     2.18      NA    NA
```

No coefficients, so how do we interpret?

Replace each linear coefficient with 2d plot

```
par(mfrow = c(1,2))
plot(fit_gam_loess)
```



Interpretation: holding other variables constant

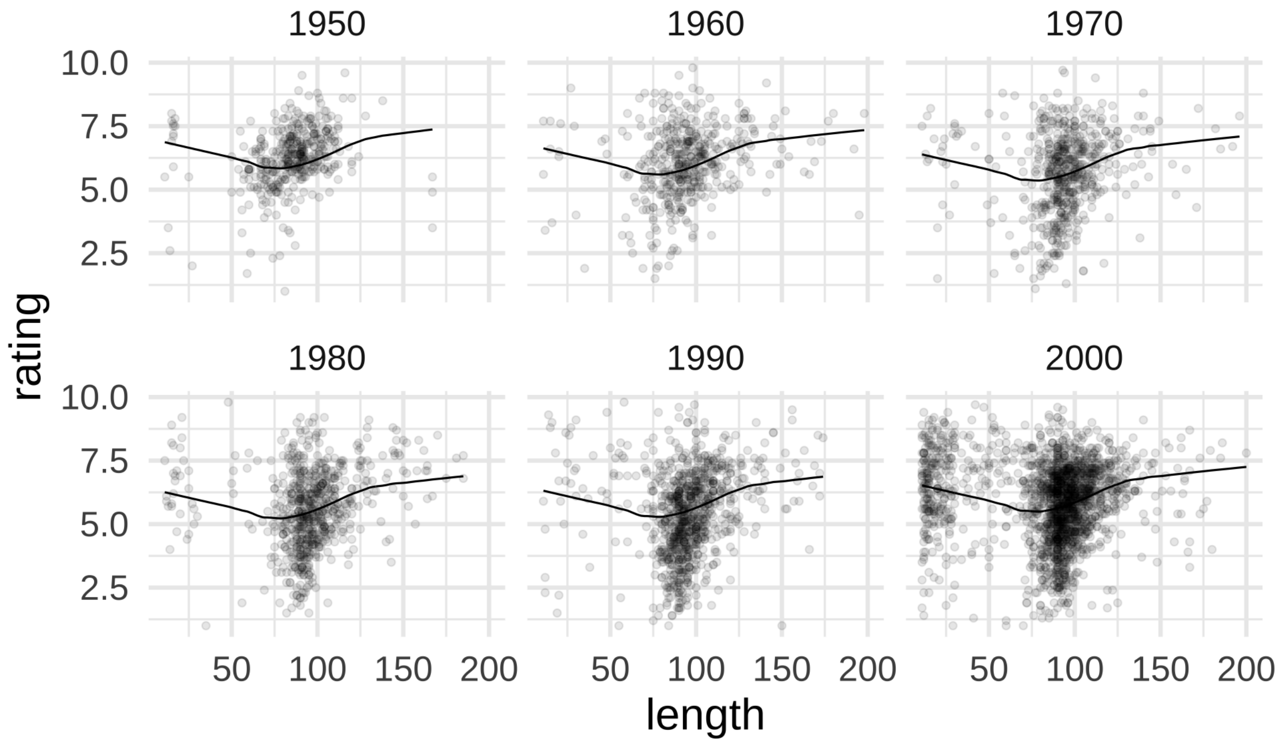
```
df_hat <- df %>%
  mutate(.fitted = predict(fit_gam_loess))

df_fixed_year <- df_hat %>%
  filter(year %in% c(1950, 1960, 1970, 1980, 1990, 2000))

df_fixed_length <- df_hat %>%
  filter(length %in% c(80, 100, 120))
```

Let's look at a few specific years and plot the **fitted relationship** with length for each of those subsets of the data

Do the same for a few specific lengths and **fitted relationship** with year

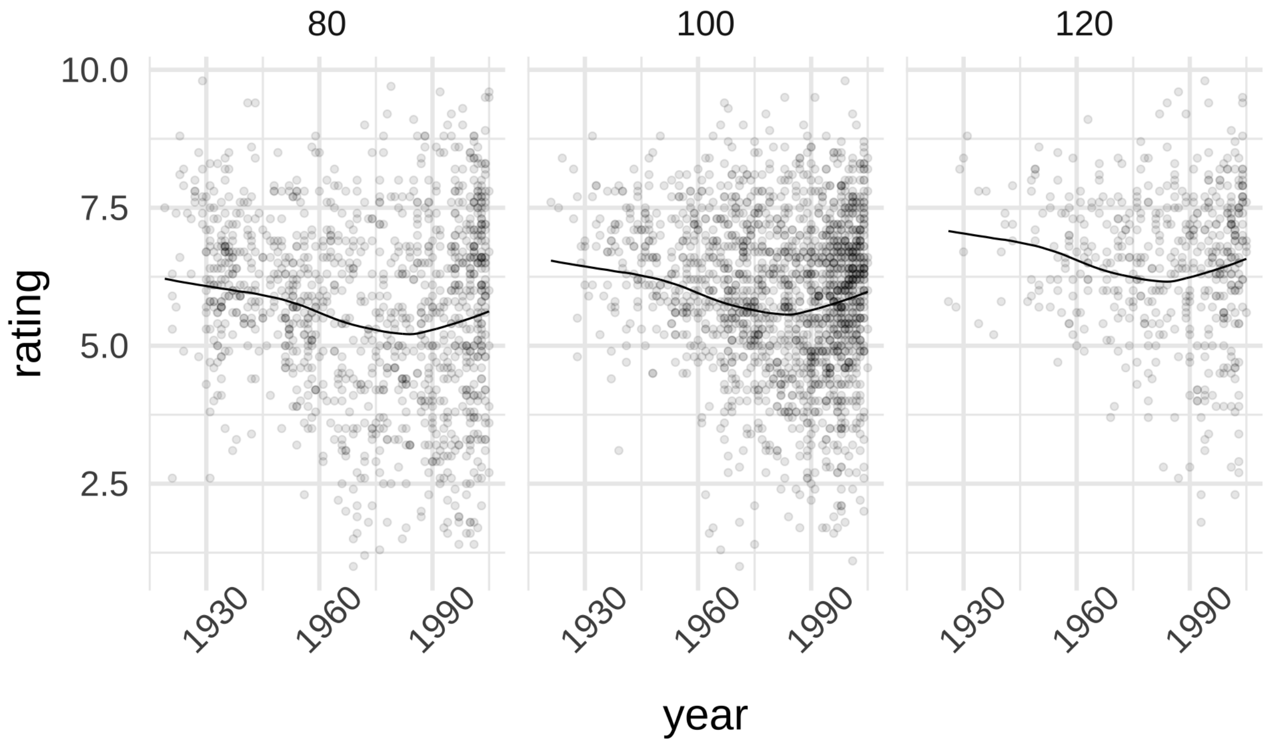
"Coefficient" of length, holding year constant

```
df_fixed_year %>%
  ggplot(aes(length, rating)) + geom_point(alpha = .1) +
  geom_line(aes(y = .fitted)) + facet_wrap(~ year)
```

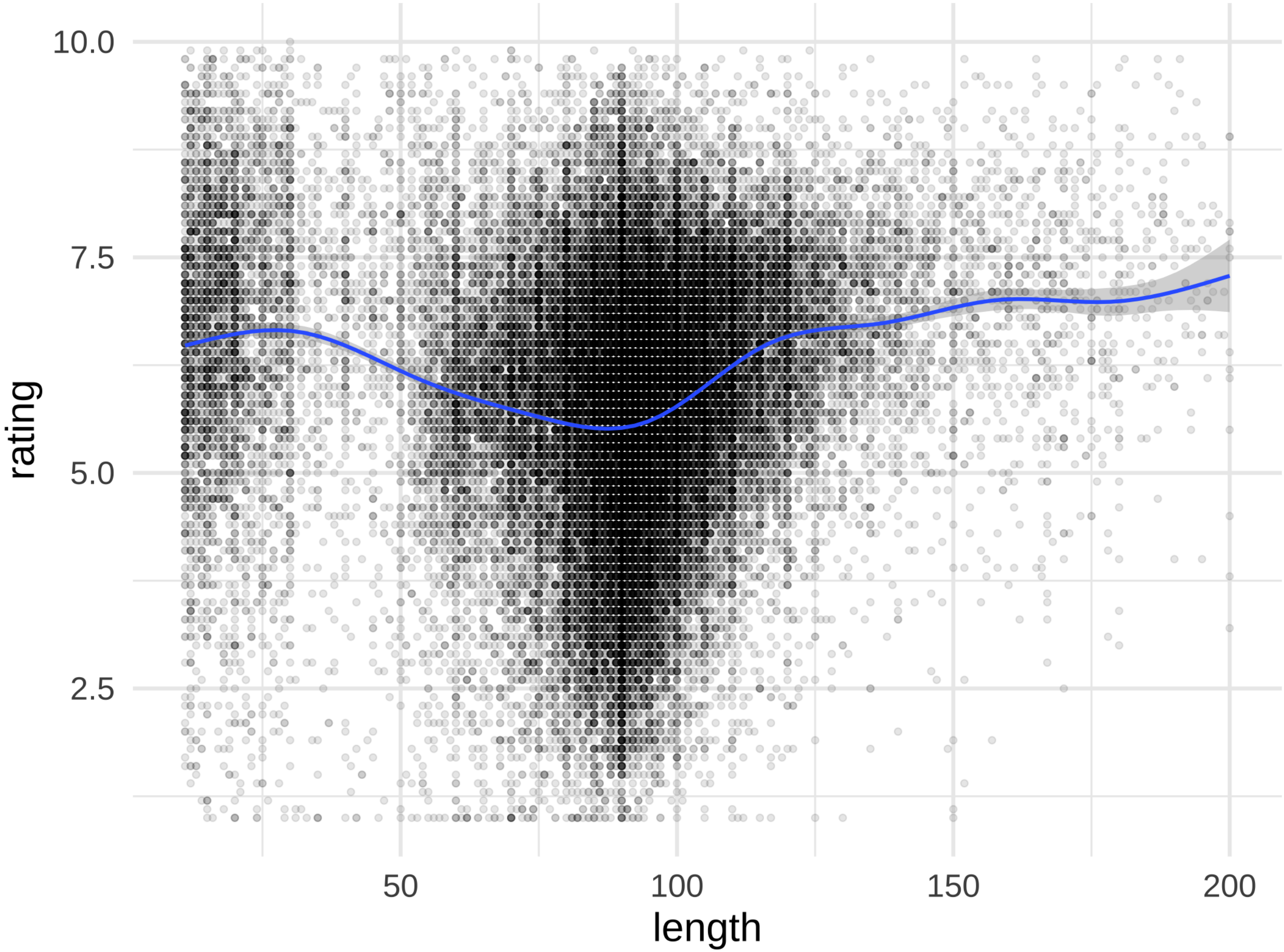


"Coefficient" of year, holding length constant

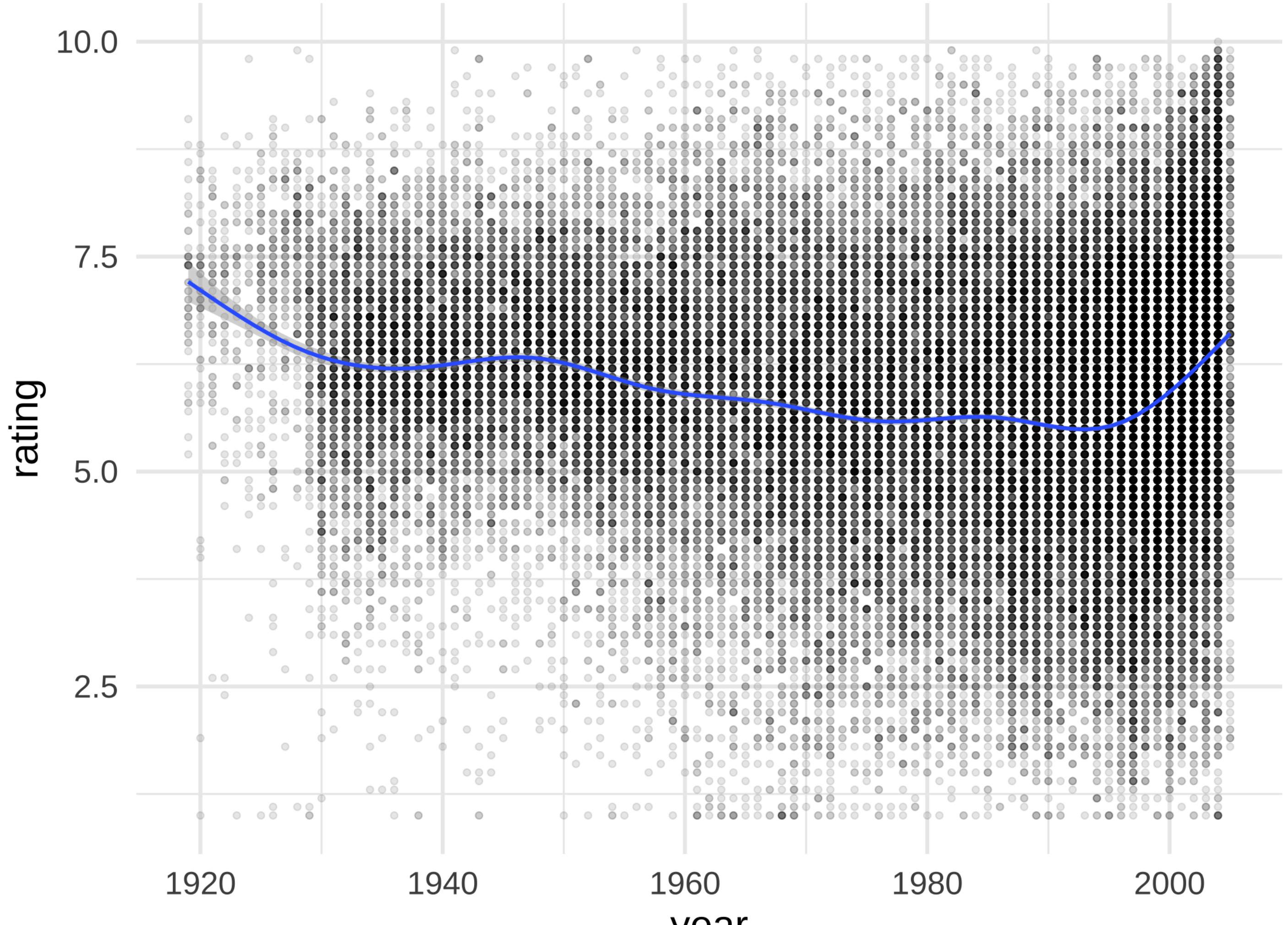
```
df_fixed_length %>%
  ggplot(aes(year, rating)) + geom_point(alpha = .1) +
  geom_line(aes(y = .fitted)) + facet_grid(~ length) + theme(axis.
```



One univariate non-linear relationship

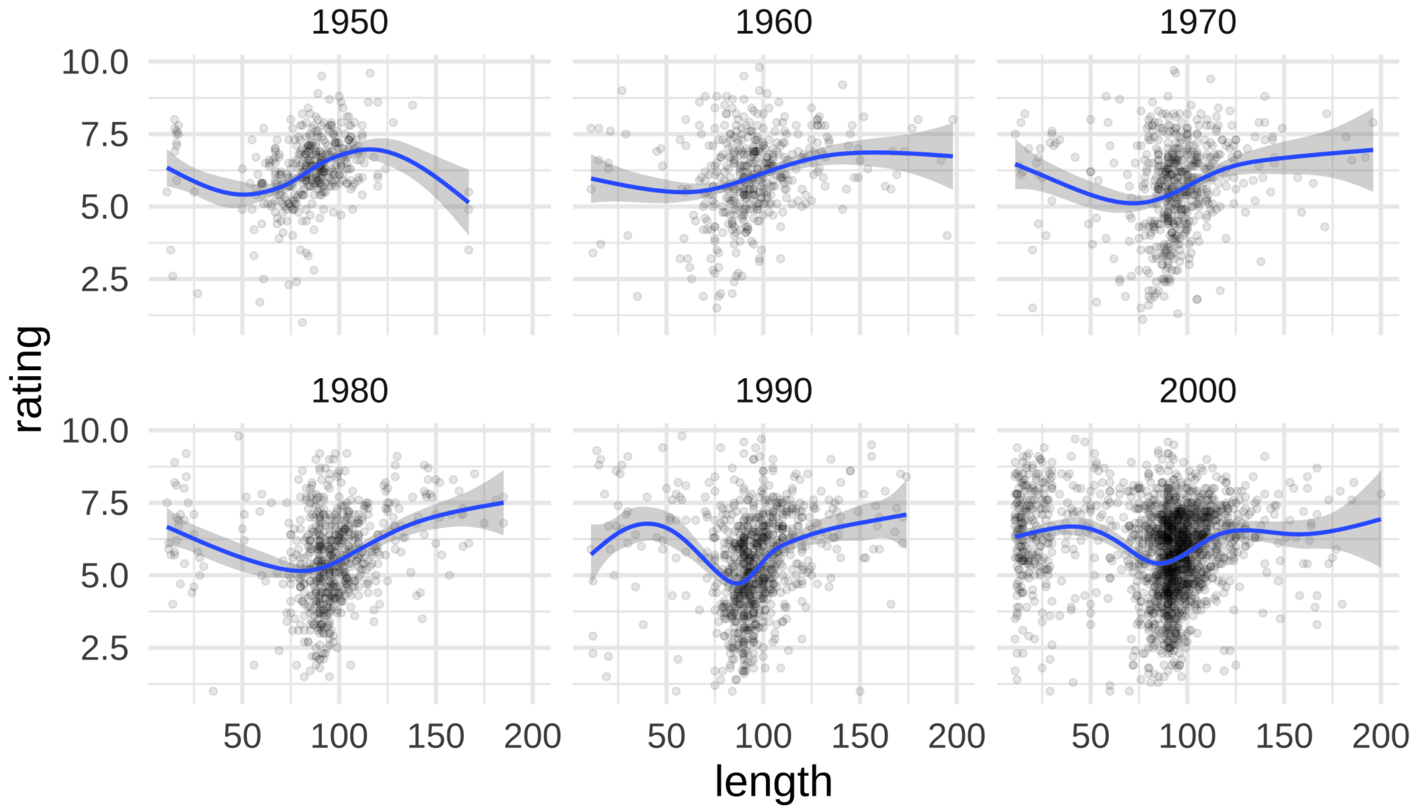


Another univariate non-linear relationship



Interactions in the movies data

Does the relationship between length and rating change depending on the year? Let's check a few years



Misspecification: failure of additivity

Difficult to tell because of small n outside the range of length between 1 and 2 hours

But I think it's possible the *relationship* is changing over time, i.e. there is an interaction

$$\frac{\partial}{\partial \text{length}} \mathbb{E}[\text{rating}|\text{length}, \text{year}] \approx g(\text{length}, \text{year})$$

Since the right hand side does not depend on length *only*, the additive model might be a poor fit

Less accurate predictions

(Possibly importantly) wrong interpretations

"Linear modeling assumption"

Why are we so often *assuming* linearity? (of the right hand side)

$$g(\mathbb{E}[\mathbf{y}]) = \beta_0 + \beta^T \mathbf{x}$$

- Easier to interpret, sure...
- But also easier to estimate

Sometimes non-linearity is clear from the data or domain info

Other times it's less clear, and makes it harder to learn a CEF

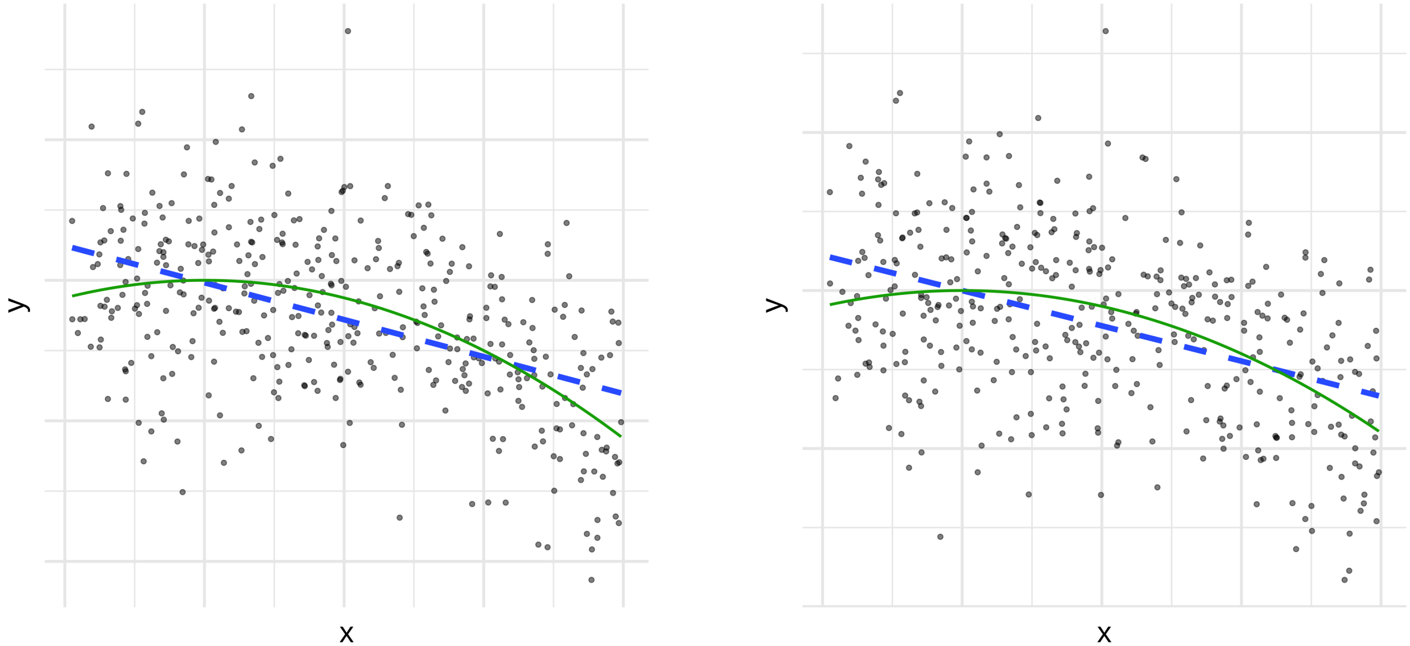
Fundamental limits in non-linearity

Applies to many ML approaches

- GAMs (Generalized Additive Models)
- Nearest neighbors
- Kernels
- Trees
- Networks (deep learning)

(Can use any for both **regression** and **classification**)

Non-linear regression



One CEF is $f(x) = -1 + 2x - x^2$, the other is $f(x) + g(x)$

Fitting the "true" models

```r
fit <- function(D) {
  list(
  lm(y ~ x, D),
  lm(y ~ f(x), D),
  lm(y ~ f(x) + g(x), D))
}
models_data_f <-
  fit(data_f)
models_data_fg <-
  fit(data_fg)
```

Lists of fitted models on each dataset

- Linear (underfit?)
- $f(x)$
- $f(x) + g(x)$

```r
models_data_f
```

```
## [[1]]
##
## Call:
## lm(formula = y ~ x, data = D)
##
## Coefficients:
## (Intercept)            x
##       1.019       -1.053
##
##
## [[2]]
##
## Call:
## lm(formula = y ~ f(x), data = D)
##
## Coefficients:
## (Intercept)         f(x)
##   0.0009062    1.0027503
##
```

```r
map_dfr(models_data_f, glance) # true CEF = f
```

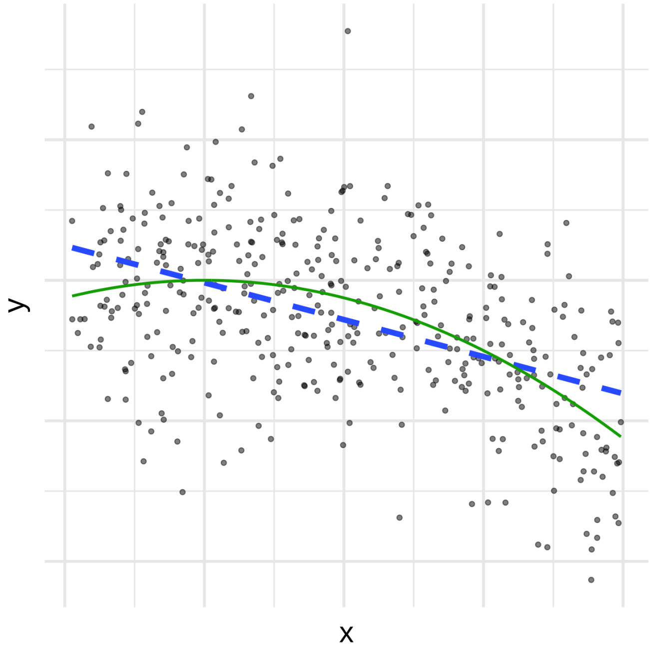
```
## # A tibble: 3 × 12
##   r.squared adj.r.squa…¹ sigma stati…²  p.value    df logLik   AIC   BIC
##       <dbl>        <dbl> <dbl>   <dbl>    <dbl> <dbl>  <dbl> <dbl> <dbl>
## 1     0.227        0.225  1.08   117.  4.20e-24     1  -598. 1201. 1213.
## 2     0.282        0.281  1.04   157.  1.54e-30     1  -583. 1172. 1184.
## 3     0.282        0.279  1.04    78.1 2.42e-29     2  -583. 1174. 1190.
## # … with 2 more variables: df.residual <int>, nobs <int>, and abbreviate
## #   variable names ¹adj.r.squared, ²statistic, ³deviance
```

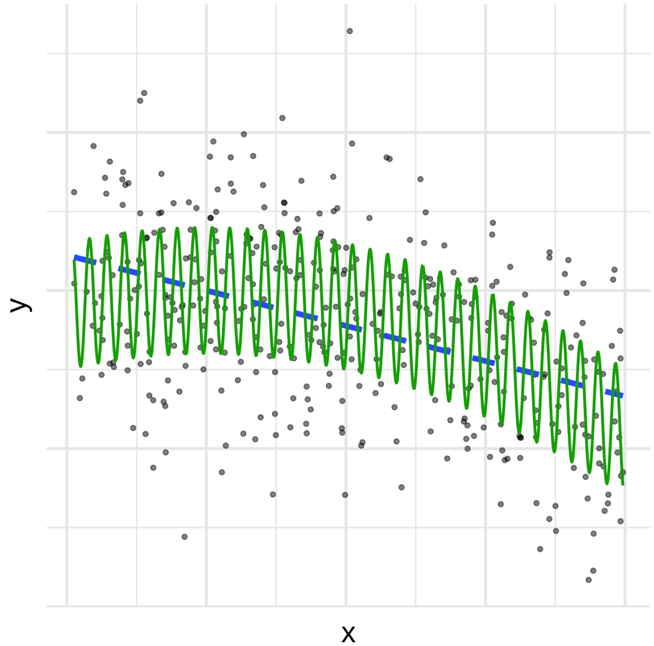
```r
map_dfr(models_data_fg, glance) # CEF = f + g
```

```
## # A tibble: 3 × 12
##   r.squared adj.r.squa…¹ sigma stati…²  p.value    df logLik   AIC   BIC
##       <dbl>        <dbl> <dbl>   <dbl>    <dbl> <dbl>  <dbl> <dbl> <dbl>
## 1     0.189        0.187  1.29    92.9 6.83e-20     1  -668. 1341. 1353.
## 2     0.221        0.219  1.26   113.  2.35e-23     1  -660. 1325. 1337.
## 3     0.469        0.467  1.04   175.  2.52e-55     2  -583. 1174. 1190.
## # … with 2 more variables: df.residual <int>, nobs <int>, and abbreviate
## #   variable names ¹adj.r.squared, ²statistic, ³deviance
```

Both look like high noise level, but 1 has ~double R^2? 🤨

Revealing $f(x) + g(x)$ 🤪





Datasets *look* very similar, but $f + g$ fits one and not the other

If not linear, then what?

Choose a **space of functions** to optimize over

- Linear functions in p variables $\leftrightarrow$ vector space $\mathbb{R}^p$
- Polynomials up to a fixed, maximum degree: also finite dimensional vector space
- Many (non-linear) function spaces are **infinite dimensional** vector spaces
  - $\{f_k(x) = \sin(k\pi x) : k \in \mathbb{Z}\}$ (Fourier basis)
  - Spaces of integrable functions, or differentiable
- Underlying math: linear algebra $\to$ functional analysis

Intuitions about function spaces

- Optimize over a larger space $\rightarrow$ fit more complex models
- Bias-variance trade-off: *both* choice of right/good space of functions *and* amount of complexity in that space
    - e.g. periodic (like last example), right wavelengths
    - e.g. smooth, right amount of wiggliness
    - e.g. "Shape constraints" like monotonic, unimodal, (log-)concave (*Application*: epidemic trajectory)

Science/modeling/inference approach: domain knowledge, first principles

ML approach: whichever function space has current SOTA software (with easy to use default settings 😆)

Optimizing over a large function space

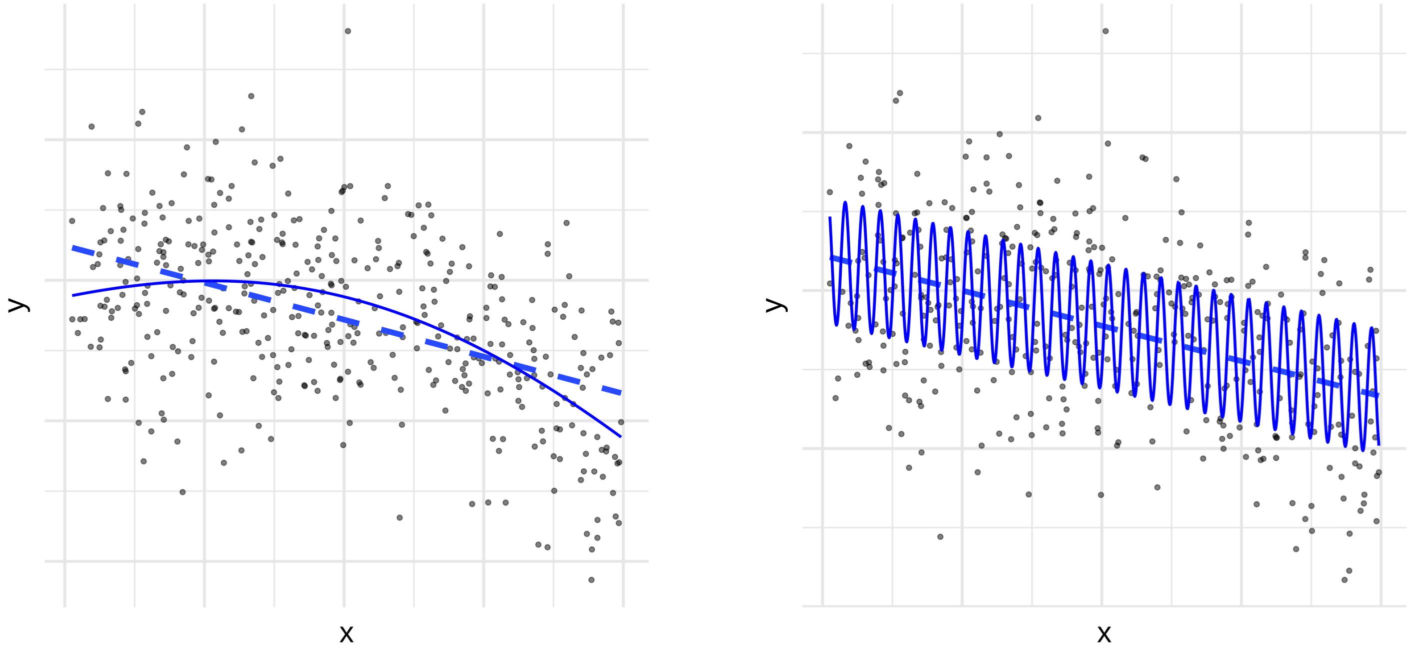
```
overfit <- function(D, k_range = 0:200) {
  fit_sin_k <- function(k) {
    fit_k <- lm(y ~ x + sin(k*x), data = D)
    glance(fit_k)$r.squared
  }
  r_squareds <- map_dbl(k_range, fit_sin_k)
  best_k <- k_range[which.max(r_squareds)]
  best_k
}
khat_f <- overfit(data_f)
khat_fg <- overfit(data_fg)
c(khat_f, khat_fg)
```

```
## [1]   1 100
```

$$\hat{f}(x) = \beta_0 + \beta_1 x + \beta_2 \sin(\hat{k}x)$$

Apparently $\hat{k} = 1$ or $\hat{k} = 100$, respectively

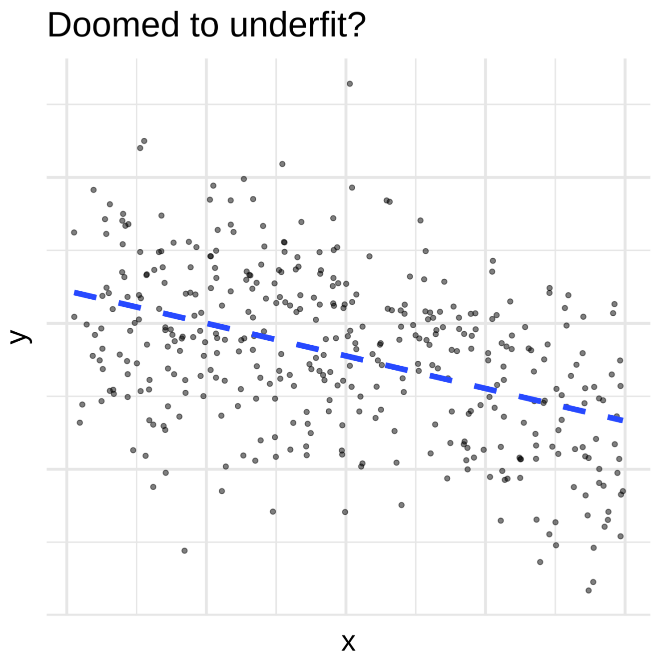
Plotting the "best" models



Can we believe this?

So which is it?

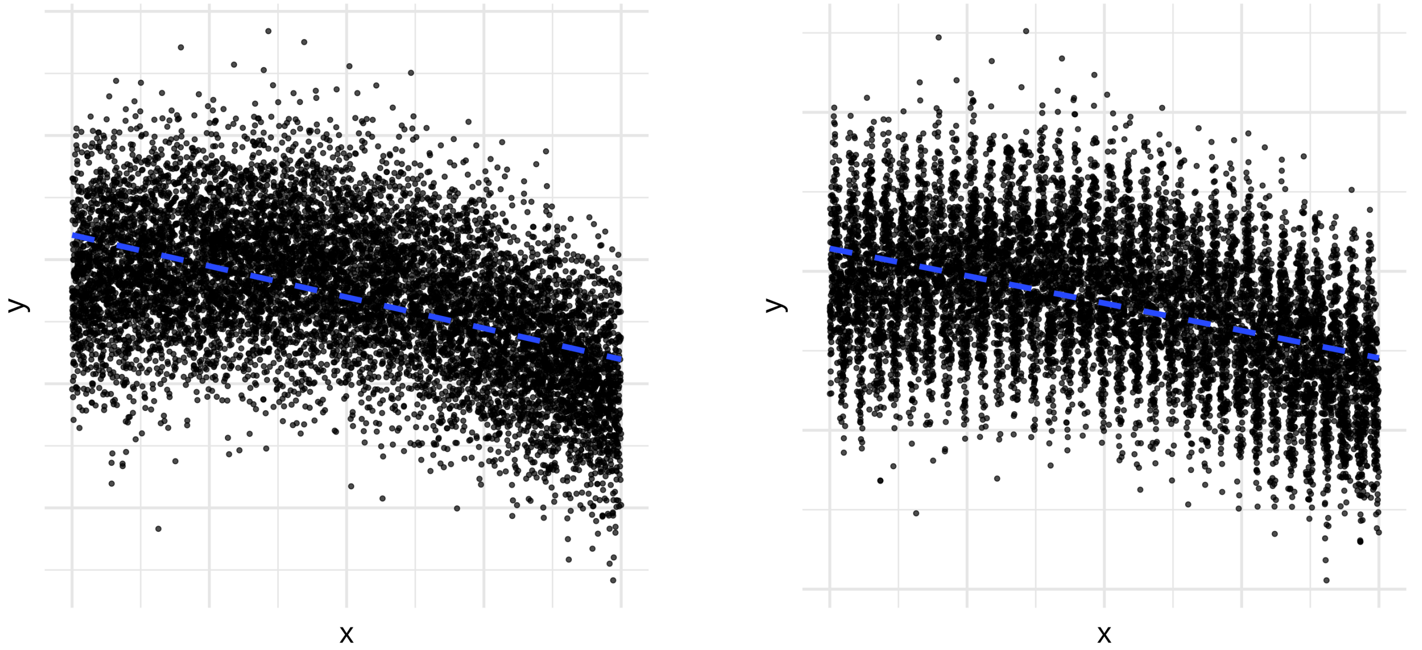
When we aren't doing simulations we just have the data



We don't know signal/noise level, function space, complexity...

The "big data" advantange

With larger samples we could tell these two cases apart



Use more data for validation / in-distribution generalization

Non-linearity and overfitting

Much of machine learning and "AI" is about having large enough datasets to search large spaces of functions and fit complex models without **variability problems** from overfitting

i.e. good in-distribution generalization (new data, same DGP)

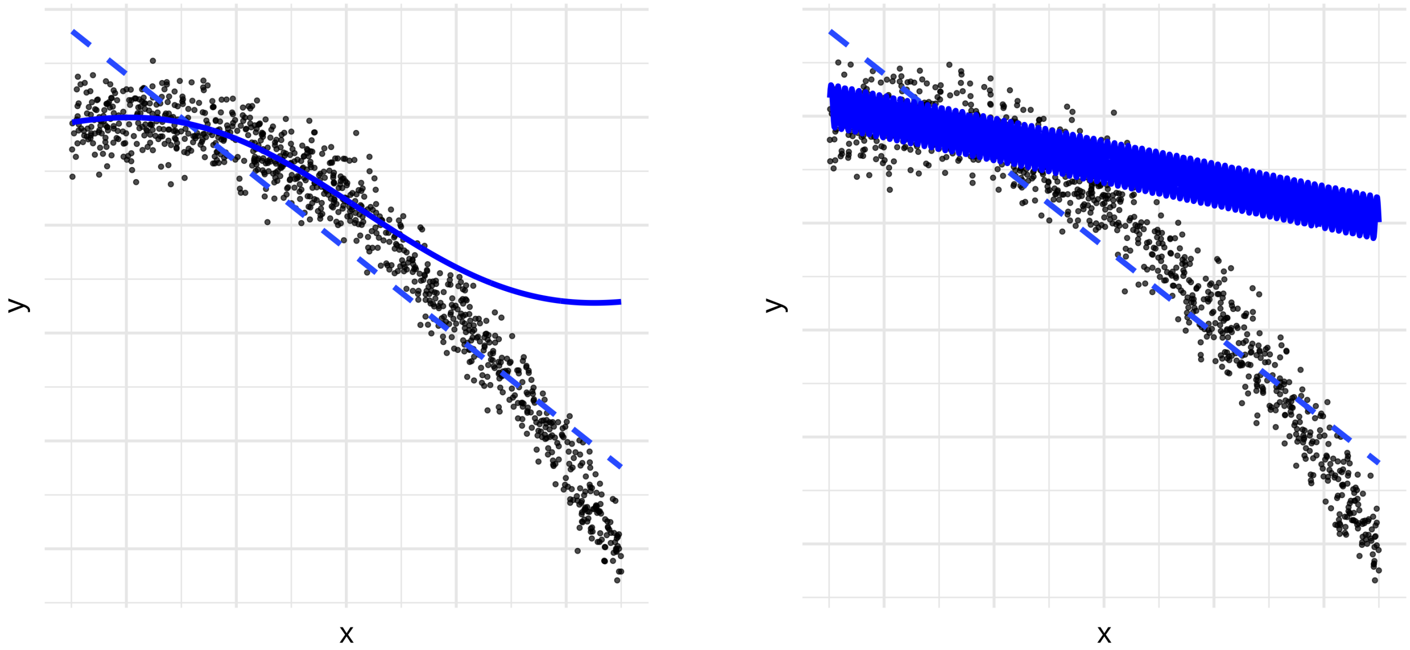
Intuition: more complex models are more sensitive to small changes in the data, or more "brittle"

Statistical wisdom: another reason to prefer simpler models may be better out-of-distribution generalization

i.e. avoiding **bias problems** from overfitting

Out-of-distribution generalization

What if we test on data outside the original range/distribution?



Simpler/"underfit" models (dashed lines) *might* do better

Choosing function spaces and methods

Since this is a course in ML, we won't assume these choices can be informed by domain knowledge

A few examples based on high level **properties of the data** and **goals of the analysis** -- not an exhaustive list or flowchart

(Assuming data shape is rectangular and i.i.d., otherwise we need specialized models for other data/dependence types)

Goals	$n > p$ **(tall)**	$n \approx p$ **or** $p > n$ **(wide)**
Prediction only	Network methods	Ridge
+ Interpretation	See below	Lasso

Additivity $\to$ GAMs. Interactions $\to$ tree methods