

# APSTA-GE 2352

Statistical Computing: Lecture 8

Klinton Kanopka

New York University



NYU Grey Art Gallery

RIVER CENTER

WASHINGTON PL



# Table of Contents

## 1. Statistical Computing - Week 8

1. Table of Contents

2. Announcements

3. Vector Norms

## 2. Motivating Problem

1. Overfitting

## 3. Dealing with Overfitting

1. Regularization

2. Ridge Regression

3. LASSO Regression

## 4. More Resampling Methods

1. The Central Limit Theorem

2. The Jackknife

3. The Bootstrap

## 5. Wrap Up

1. Recap

# Announcements

- PS4 due the day before Halloween
  - Very spooky
  - How's it going?
- If you haven't already, please do the mid-semester survey
  - <https://forms.gle/Ljh25bvYeoAL5dEU8>

# Check-In

- [PollEv.com/klintkanopka](https://PollEv.com/klintkanopka)

# Vector Norms

- We often want to talk about the *length* of a vector, but how do we measure length?
- Conceptually, think of the length of a vector as the distance it spans from the origin
- We'll talk about two types of norms today

# The Euclidean Norm

# The Euclidean Norm

- The Euclidean distance a vector extends in space from the origin

# The Euclidean Norm

- The Euclidean distance a vector extends in space from the origin
- Often called the  $L^2$  norm and abbreviated  $\|\vec{x}\|_2$

# The Euclidean Norm

- The Euclidean distance a vector extends in space from the origin
- Often called the  $L^2$  norm and abbreviated  $\|\vec{x}\|_2$

$$\|\vec{x}\|_2 = \sqrt{\vec{x}^2}$$

# The Euclidean Norm

- The Euclidean distance a vector extends in space from the origin
- Often called the  $L^2$  norm and abbreviated  $\|\vec{x}\|_2$

$$\|\vec{x}\|_2 = \sqrt{\vec{x}^2}$$

$$\|\vec{x}\|_2 = \sqrt{\vec{x} \cdot \vec{x}}$$

# The Euclidean Norm

- The Euclidean distance a vector extends in space from the origin
- Often called the  $L^2$  norm and abbreviated  $\|\vec{x}\|_2$

$$\|\vec{x}\|_2 = \sqrt{\vec{x}^2}$$

$$\|\vec{x}\|_2 = \sqrt{\vec{x} \cdot \vec{x}}$$

$$\|\vec{x}\|_2 = \sqrt{\sum_{k=1}^K x_k^2}$$

# The Euclidean Norm

- The Euclidean distance a vector extends in space from the origin
- Often called the  $L^2$  norm and abbreviated  $\|\vec{x}\|_2$

$$\|\vec{x}\|_2 = \sqrt{\vec{x}^2}$$

$$\|\vec{x}\|_2 = \sqrt{\vec{x} \cdot \vec{x}}$$

$$\|\vec{x}\|_2 = \sqrt{\sum_{k=1}^K x_k^2}$$

$$\|\vec{x}\|_2 = \sqrt{x_1^2 + \cdots + x_k^2}$$

# The Manhattan Norm

# The Manhattan Norm

- Sometimes called the *Taxicab Norm*

# The Manhattan Norm

- Sometimes called the *Taxicab Norm*
- The Manhattan distance a vector extends in space from the origin

# The Manhattan Norm

- Sometimes called the *Taxicab Norm*
- The Manhattan distance a vector extends in space from the origin
- Often called the  $L^1$  norm and abbreviated  $\|\vec{x}\|_1$

# The Manhattan Norm

- Sometimes called the *Taxicab Norm*
- The Manhattan distance a vector extends in space from the origin
- Often called the  $L^1$  norm and abbreviated  $\|\vec{x}\|_1$

$$\|\vec{x}\|_1 = \sum_{k=1}^K |x_k|$$

# The Manhattan Norm

- Sometimes called the *Taxicab Norm*
- The Manhattan distance a vector extends in space from the origin
- Often called the  $L^1$  norm and abbreviated  $||\vec{x}||_1$

$$||\vec{x}||_1 = \sum_{k=1}^K |x_k|$$

$$||\vec{x}||_1 = |x_1| + \cdots + |x_k|$$

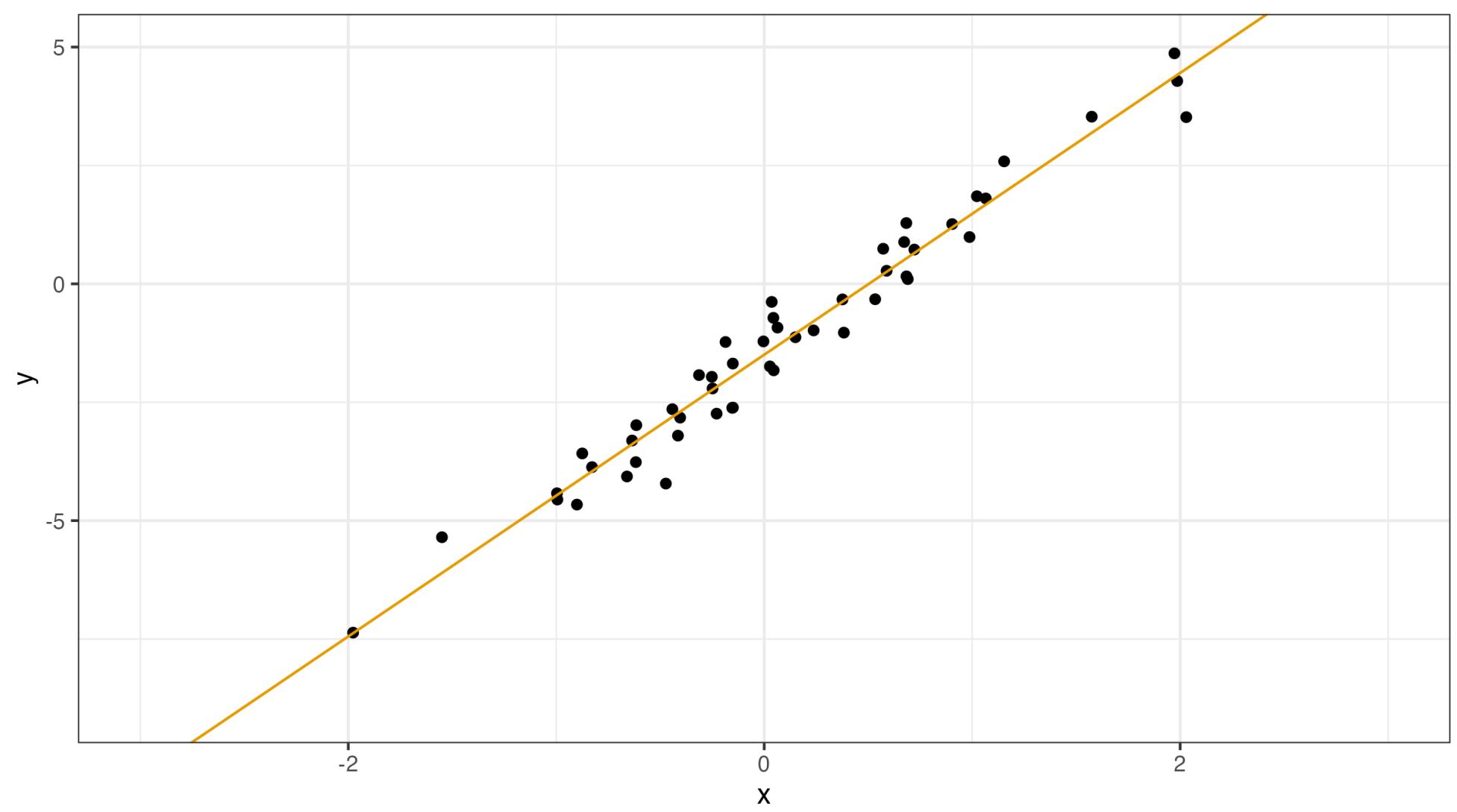
# Motivating Problem

# Overfitting

```
1 d <- data.frame(x = rnorm(50))
2 d$y <- -1.5 + 3*d$x + rnorm(50, sd=0.5)
3
4 m <- lm(y~x, d)
5 coef(m)
```

# Overfitting

```
1 d <- data.frame(x = rnorm(50))
2 d$y <- -1.5 + 3*d$x + rnorm(50, sd=0.5)
3
4 m <- lm(y~x, d)
5 coef(m)
6
7 # (Intercept)           x
8 # -1.425421    2.883035
```

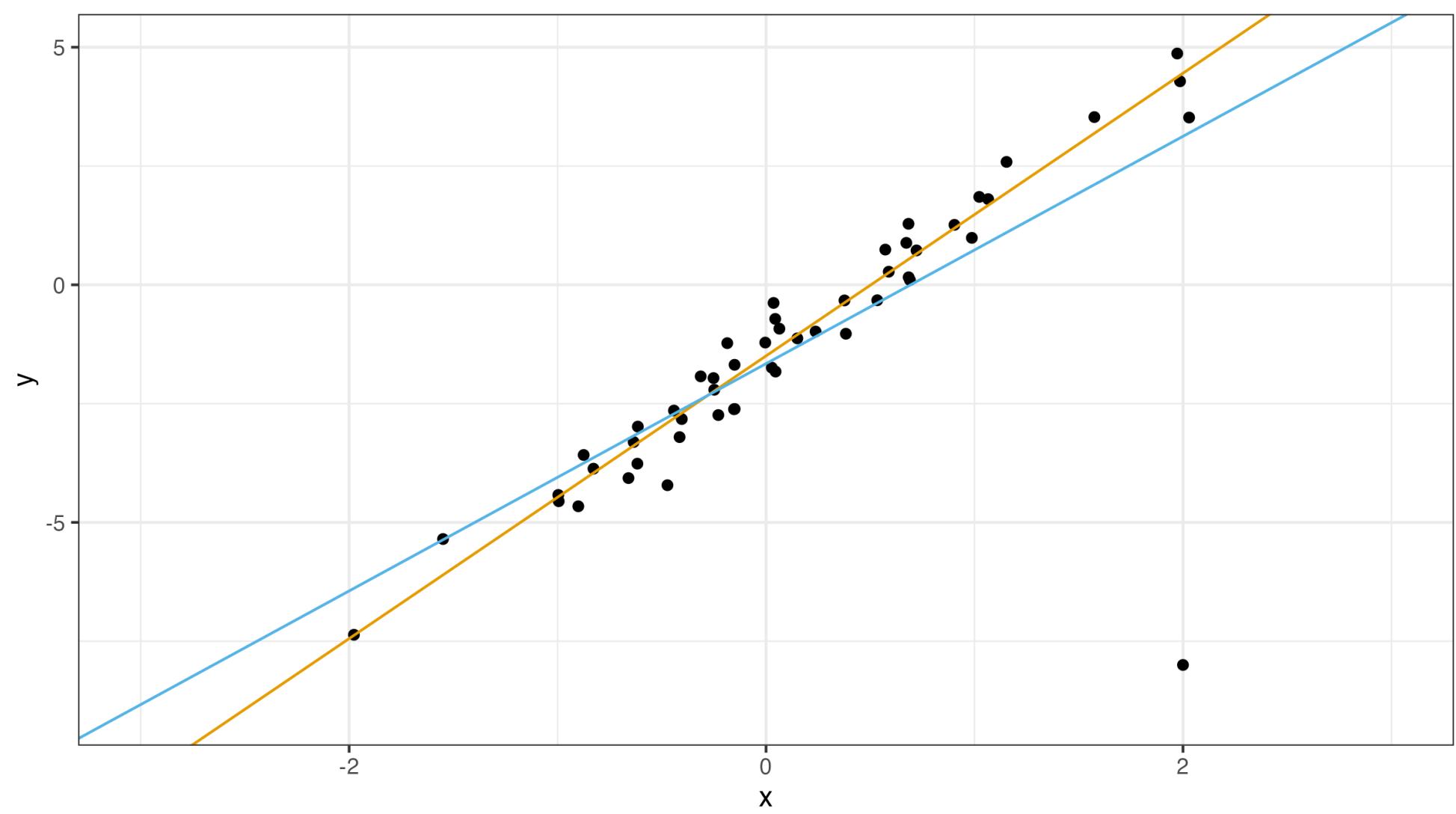


# Overfitting

```
1 d2 <- rbind(d, data.frame(x=2, y=-8))  
2  
3 m2 <- lm(y~x, d2)  
4 coef(m2)
```

# Overfitting

```
1 d2 <- rbind(d, data.frame(x=2, y=-8))
2
3 m2 <- lm(y~x, d2)
4 coef(m2)
5
6 # (Intercept)           x
7 # -1.659093    2.391916
```

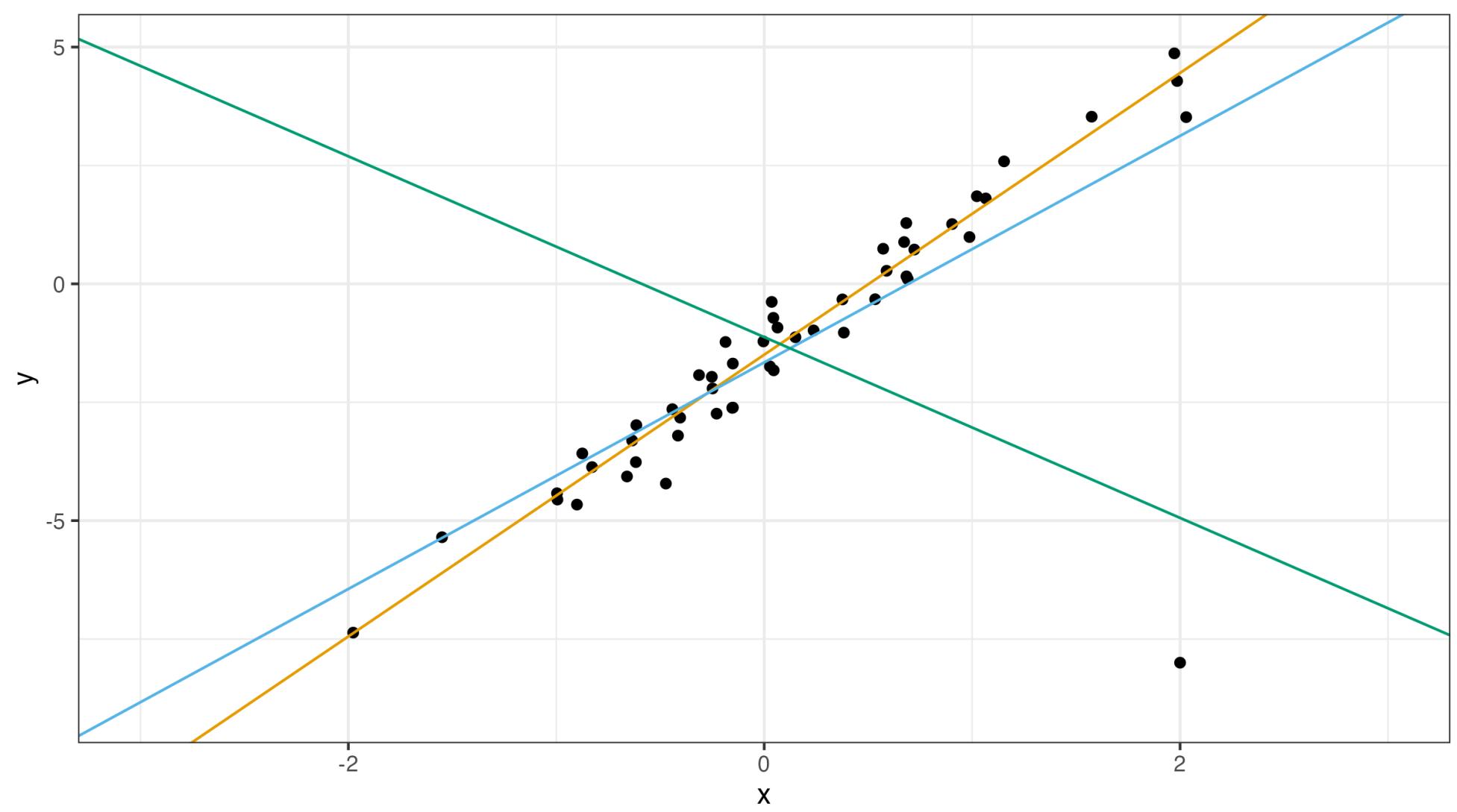


# Overfitting

```
1 d3 <- rbind(d2, data.frame(x=50, y=-100))
2 m3 <- lm(y~x, d3)
3 coef(m3)
```

# Overfitting

```
1 d3 <- rbind(d2, data.frame(x=50, y=-100))
2 m3 <- lm(y~x, d3)
3 coef(m3)
4
5 # (Intercept)           x
6 # -1.124650   -1.908941
```



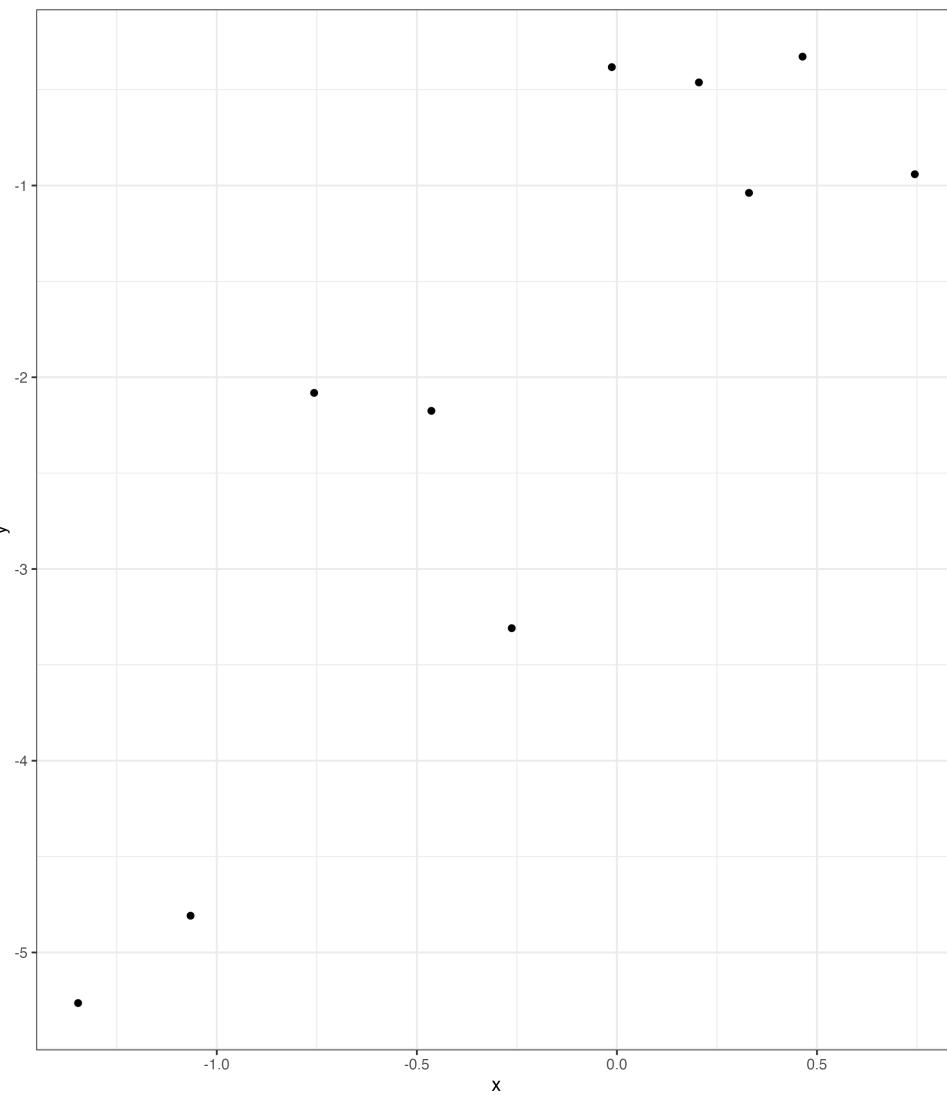
# Overfitting

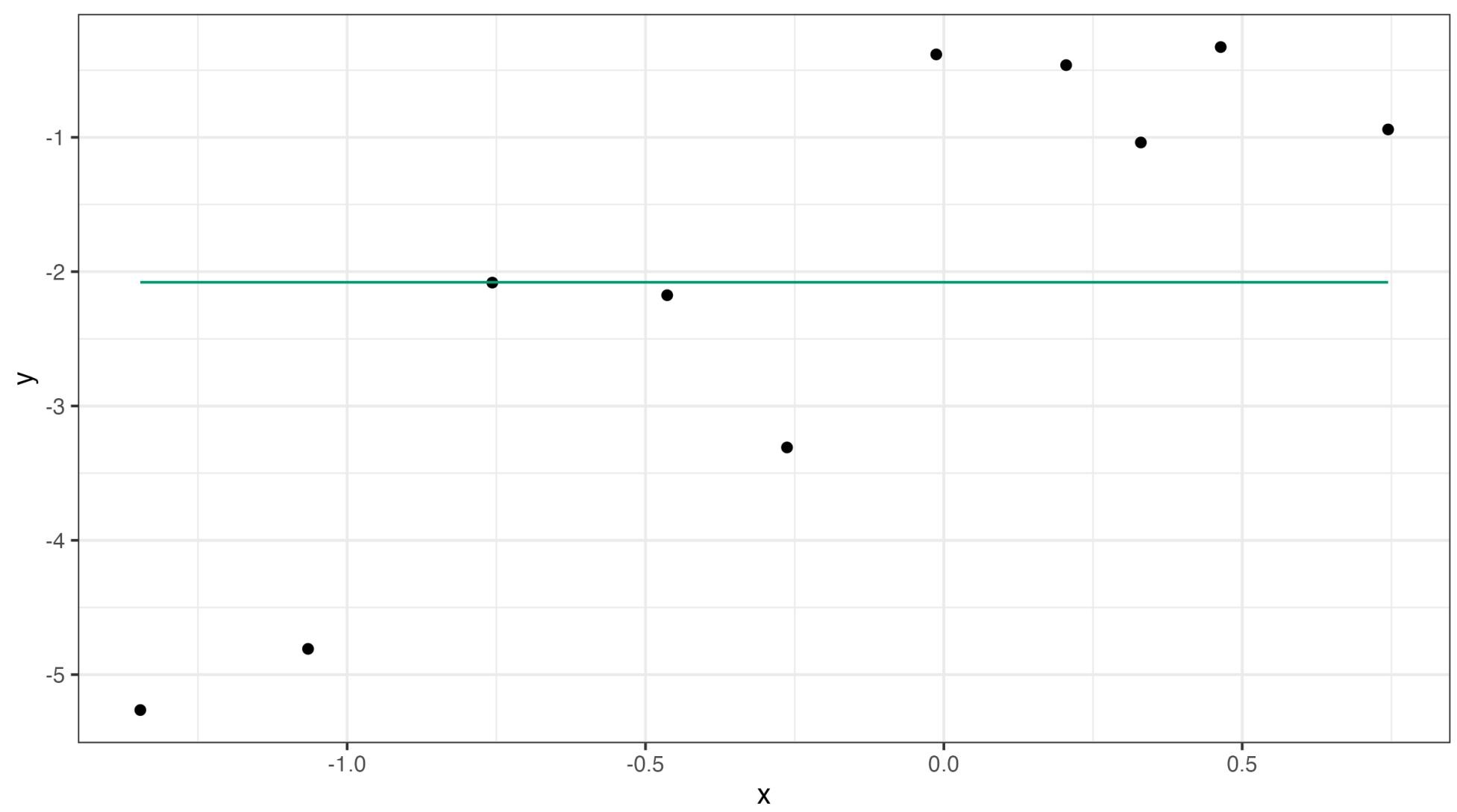
- *Overfitting* occurs when the model you are estimating has too much flexibility and can make large adjustments for small variations in the data it is fit to (or *trained on*)
- Overfitting hurts the ability of your model to *generalize*, or make reasonable predictions with data it hasn't seen yet (we call these predictions *out of sample*, or OOS)
- Overfitting often occurs under one of two conditions:
  1. The individual parameter estimates grow too large
  2. Your model has a large number of estimable parameters relative to the amount of data

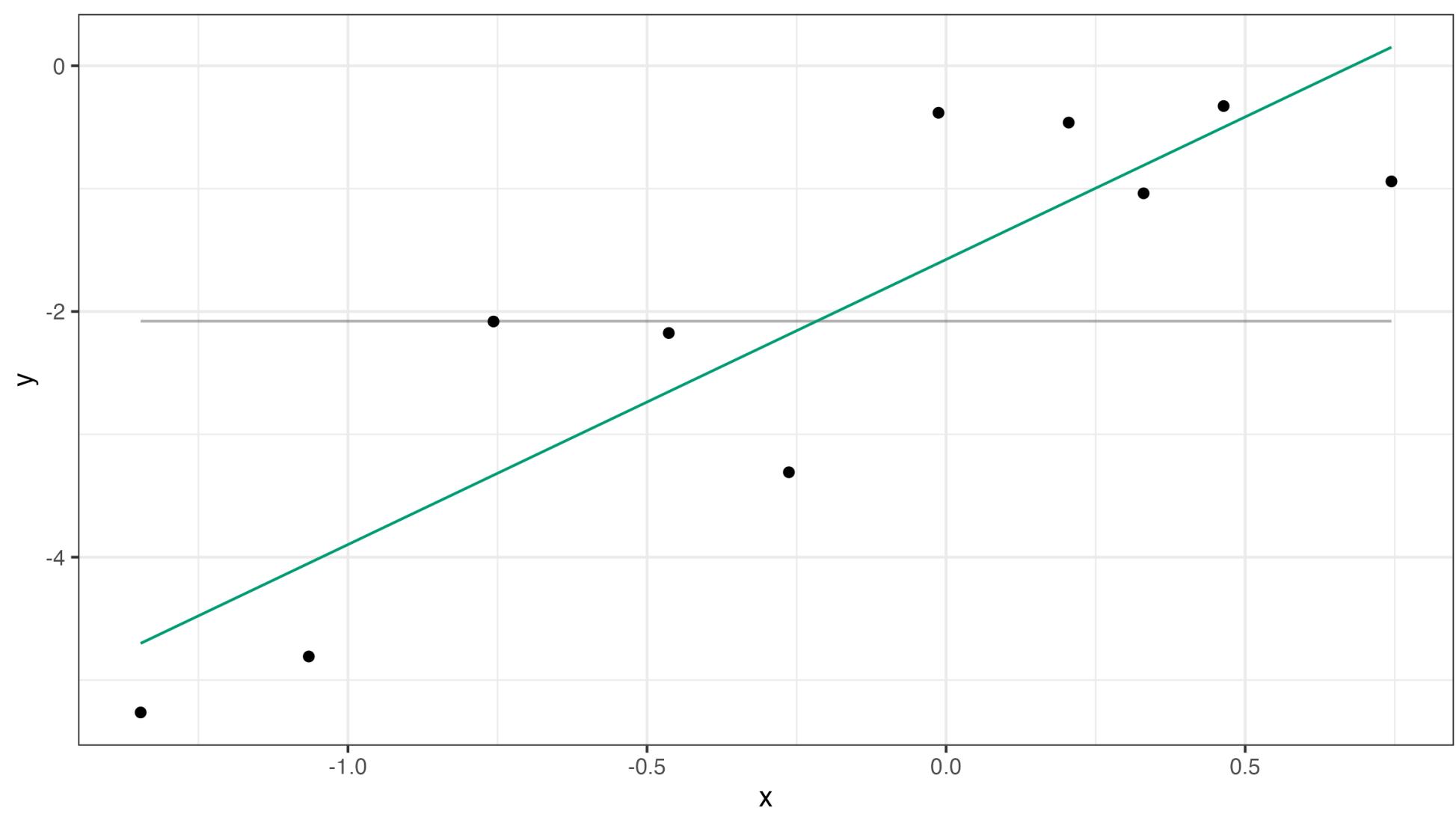
# Overfitting

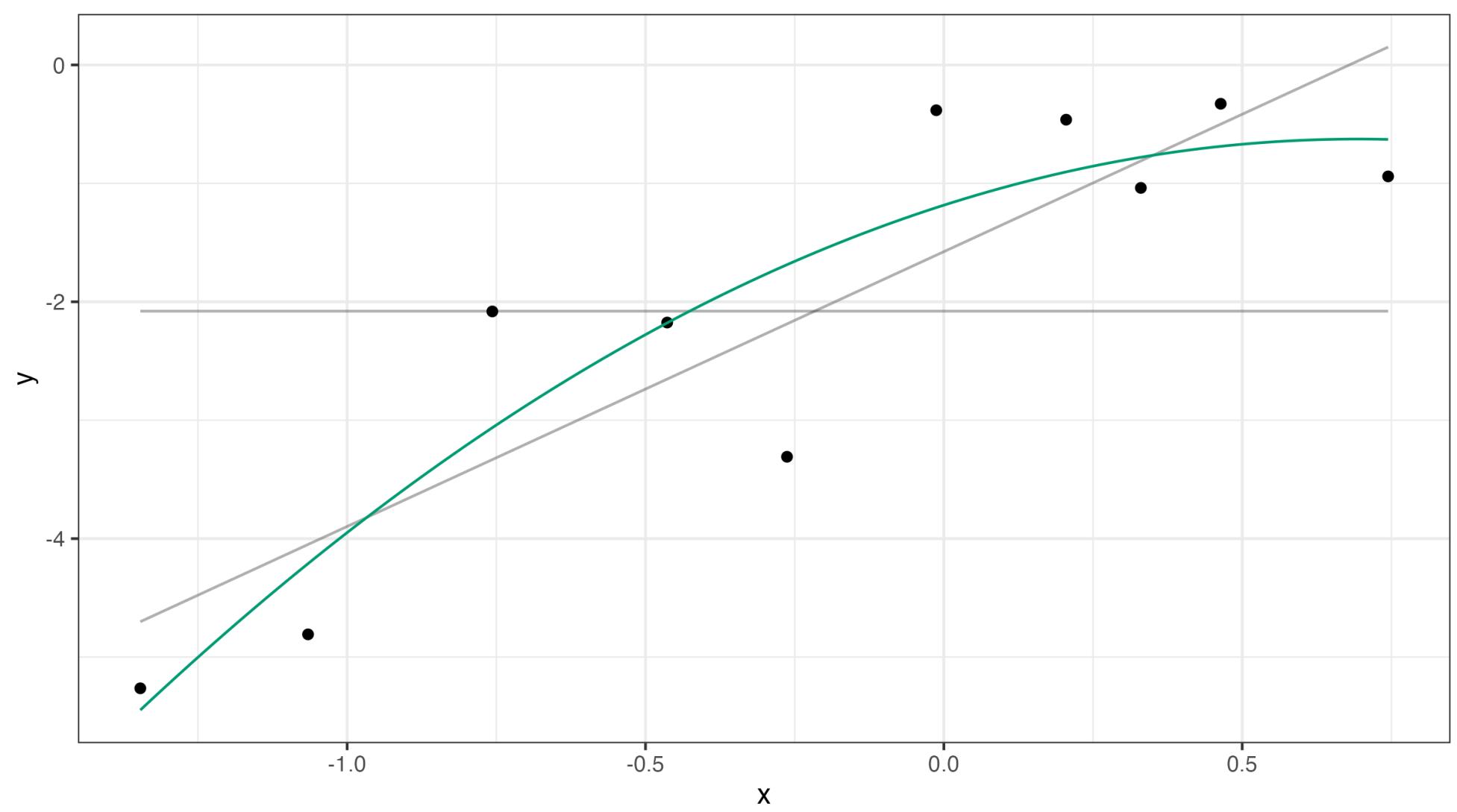
- What happens when functions become too flexible?
- We'll plot ten data points and then a polynomial function of best fit

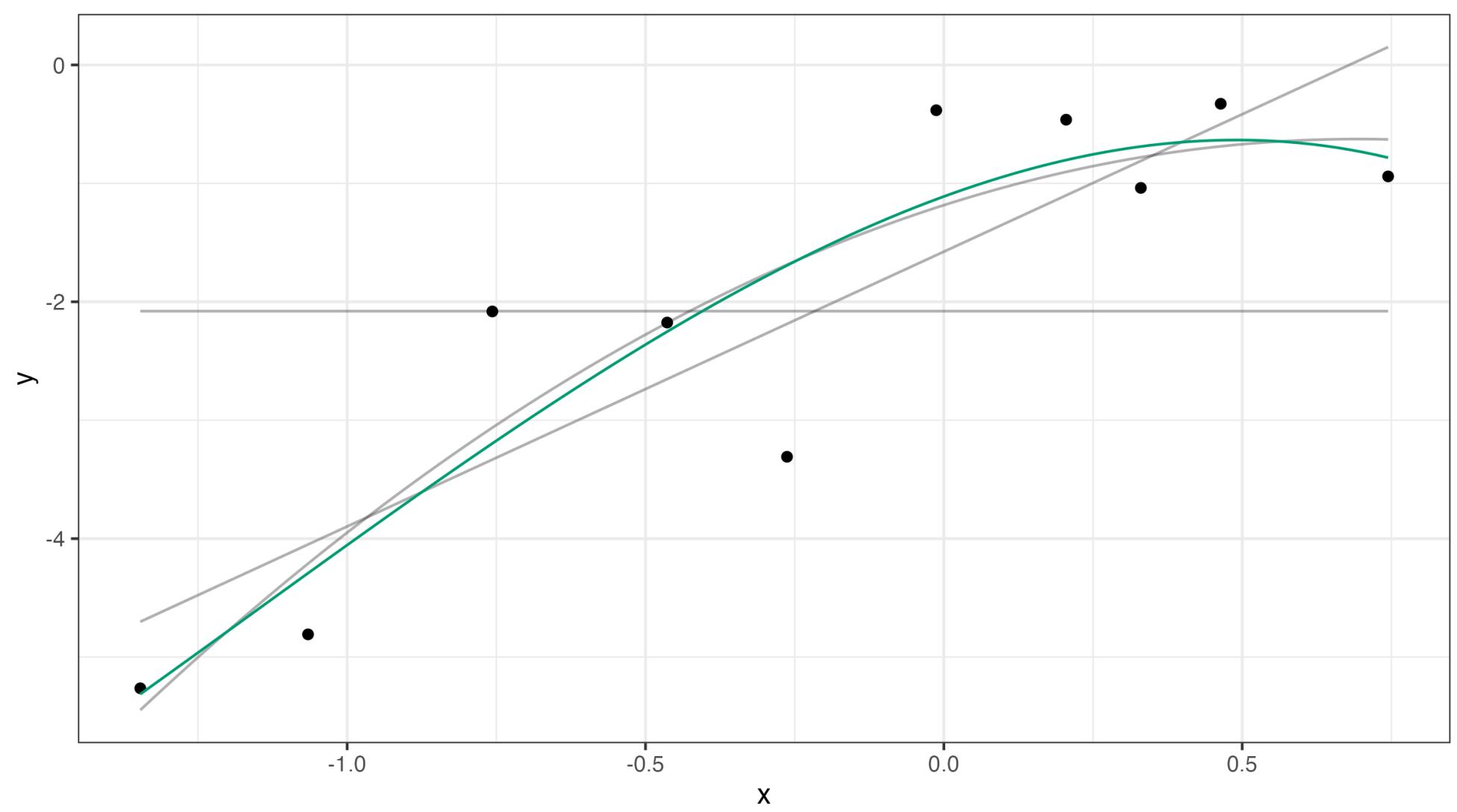
```
1 d <- data.frame(x = rnorm(10))
2 d$y <- -1.5 + 3*d$x + rnorm(10)
3
4 ggplot(d, aes(x=x, y=y)) +
5   geom_point() +
6   theme_minimal()
```

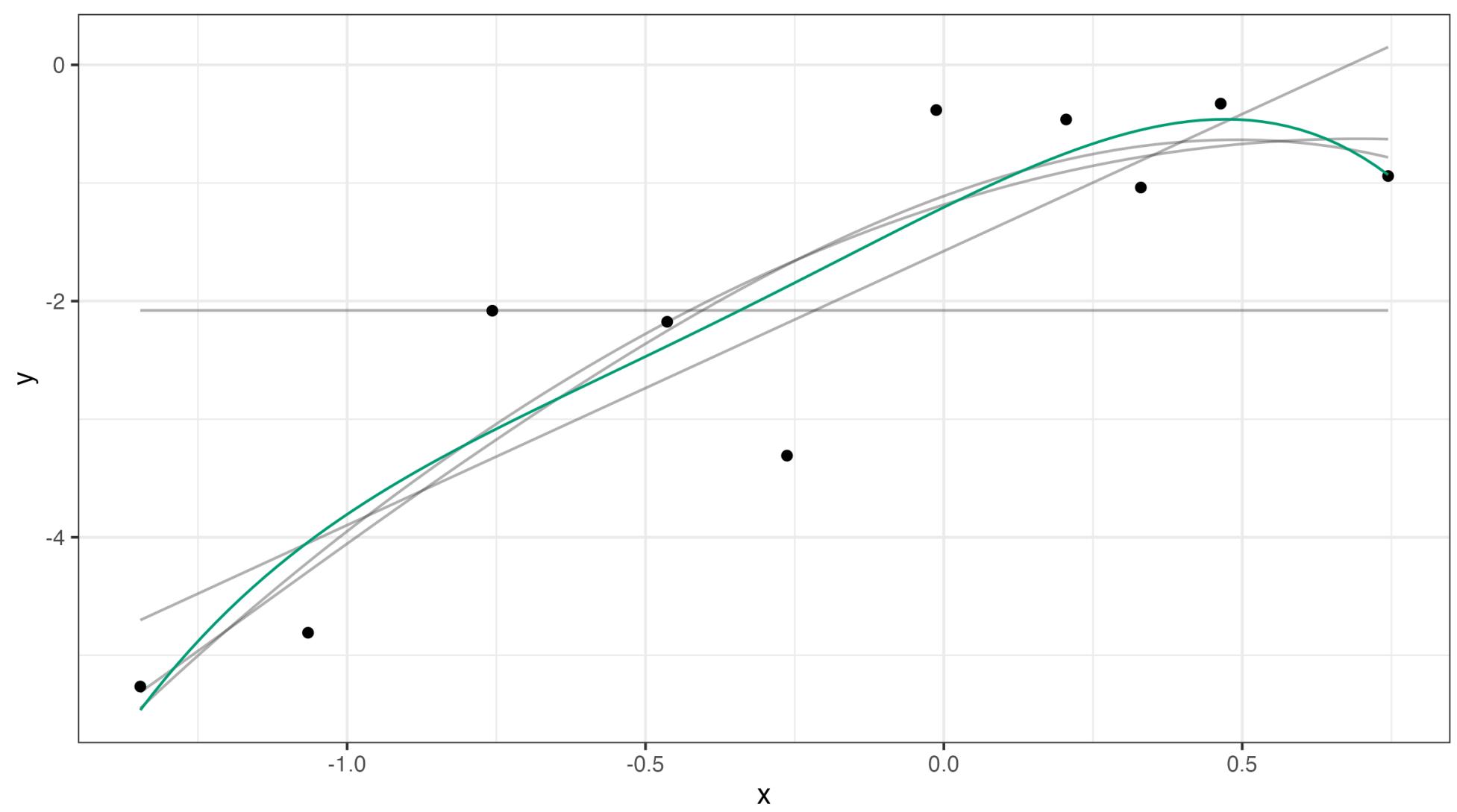


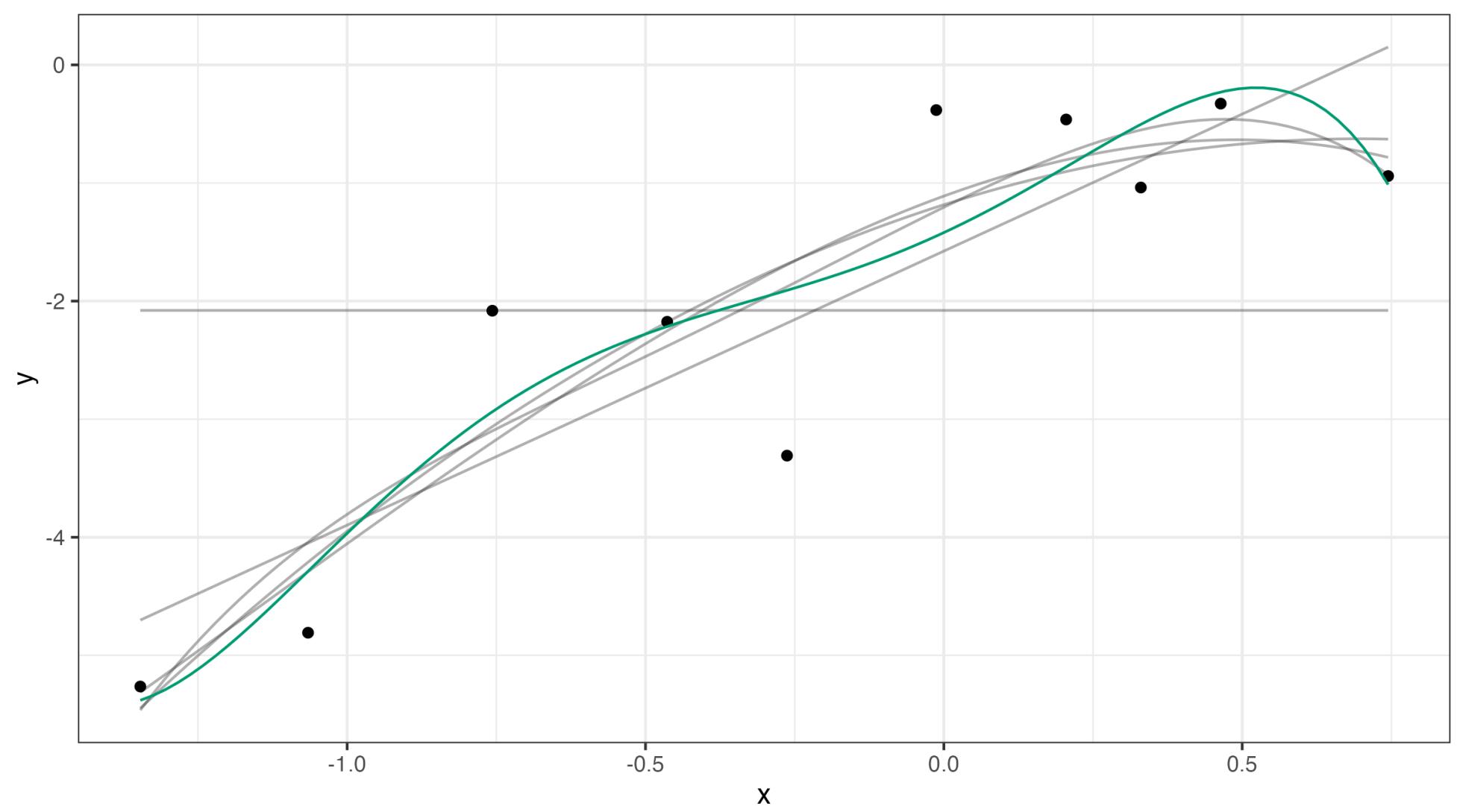


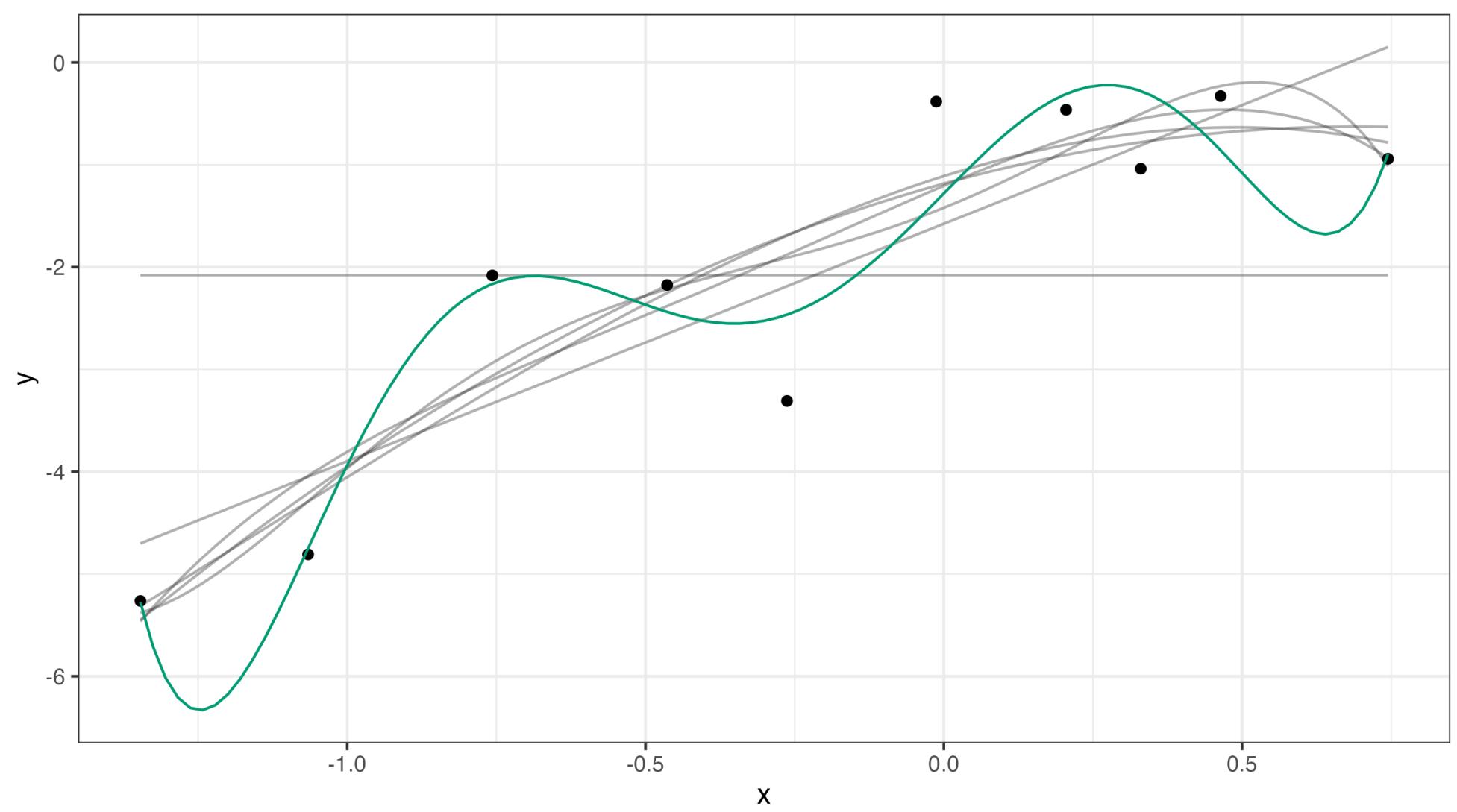


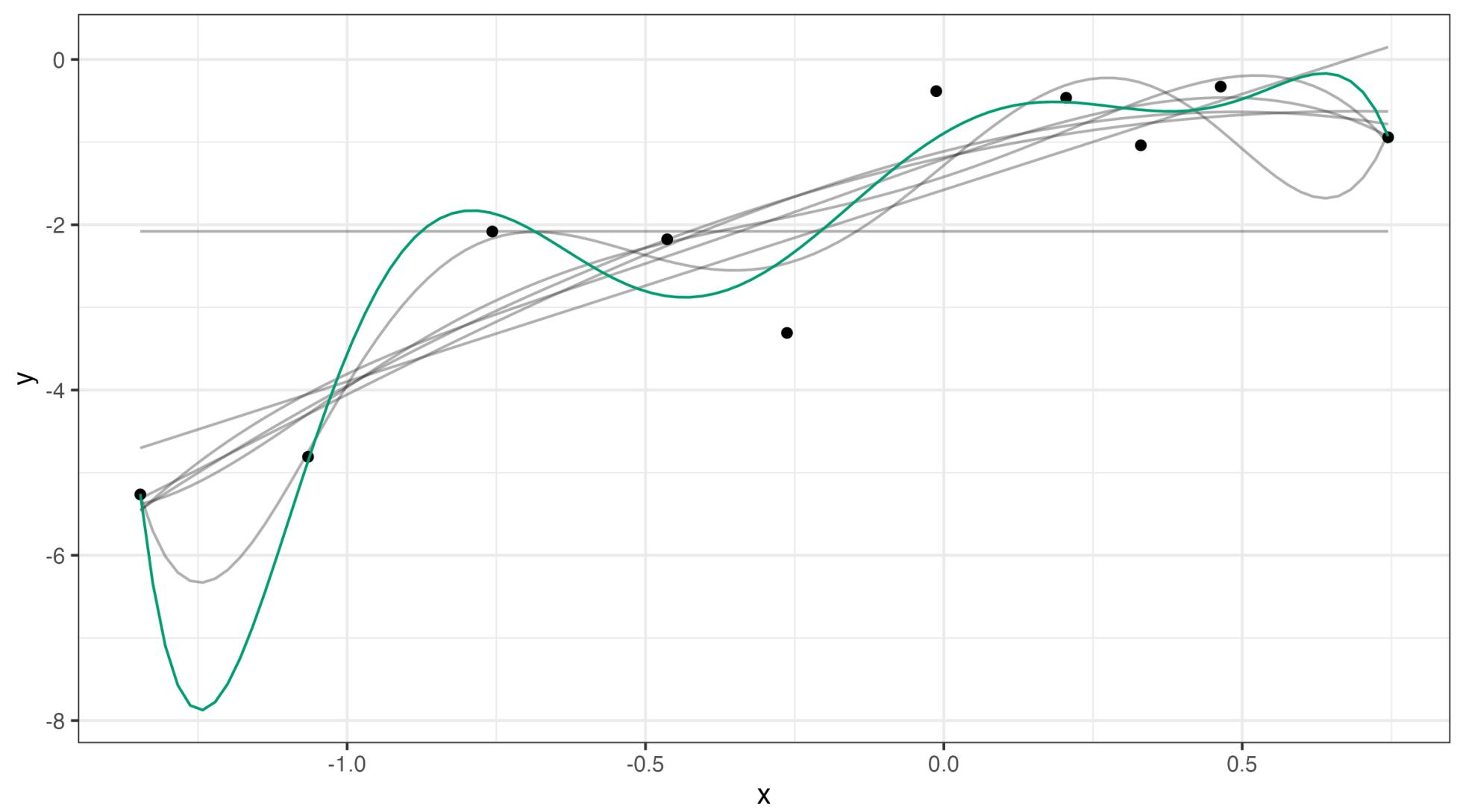


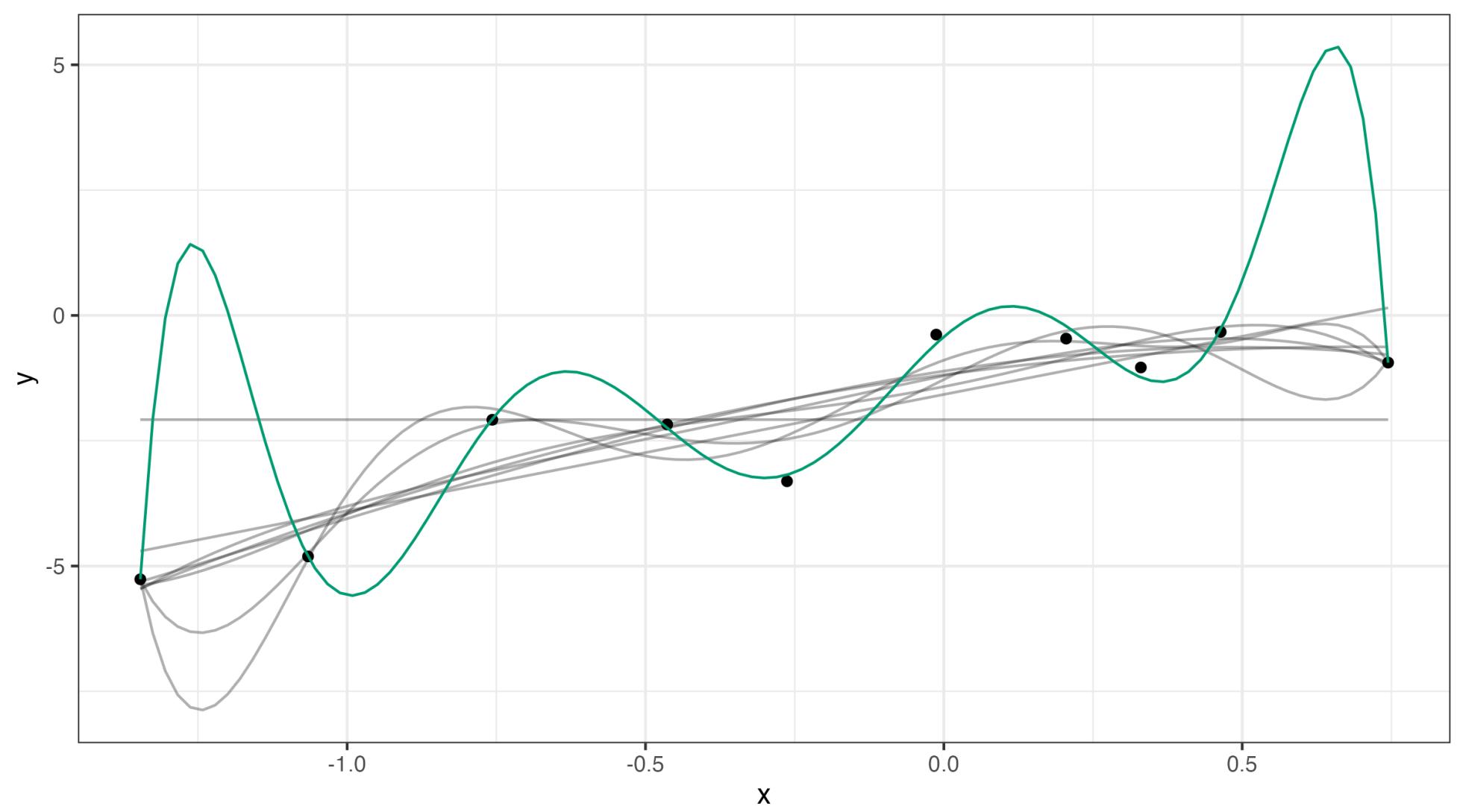


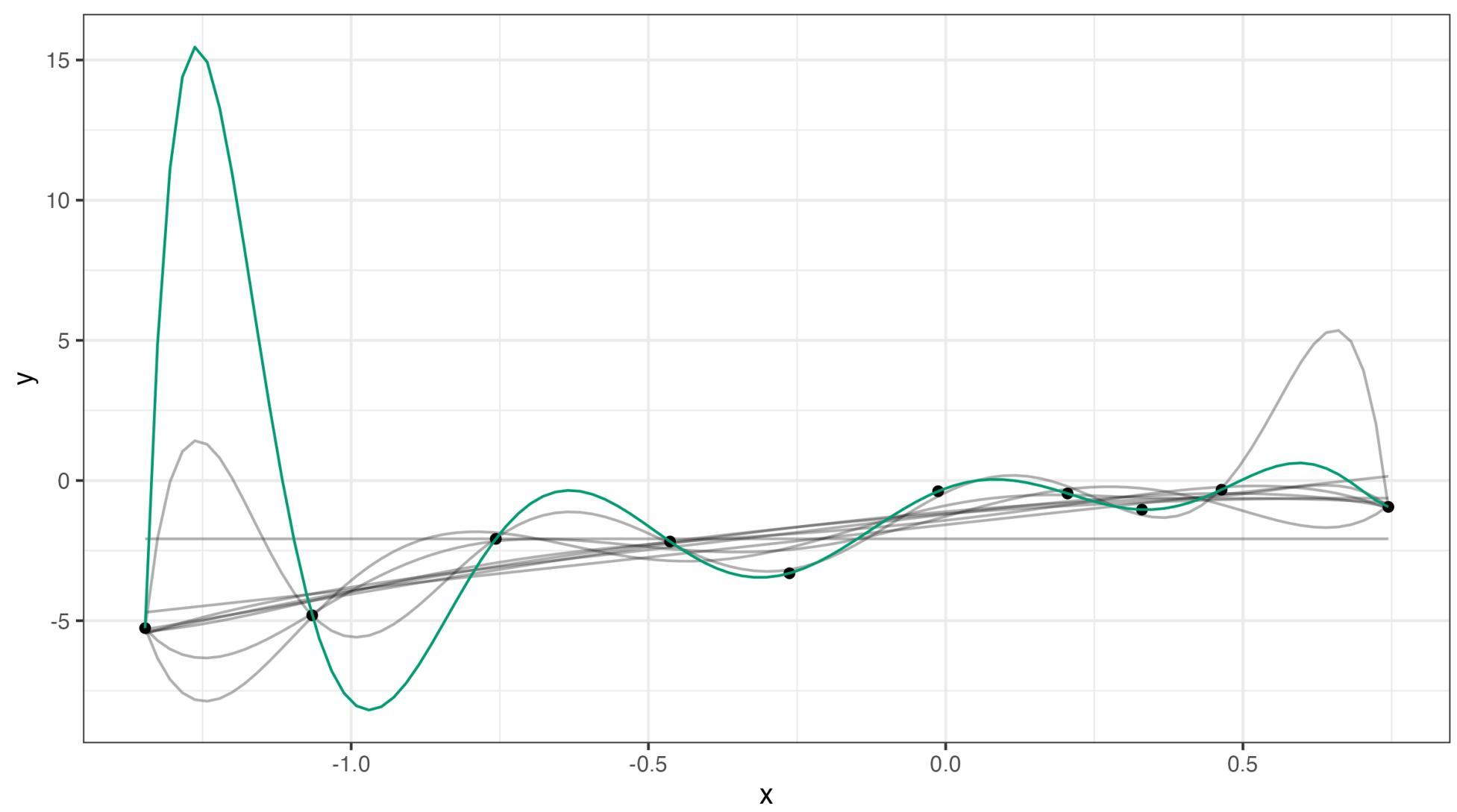












# Dealing with Overfitting

# Regularization

# Regularization

- **Core Idea:** Regularization adds a penalty to your loss function (or likelihood) when you go to estimate your parameters that helps to fight against overfitting

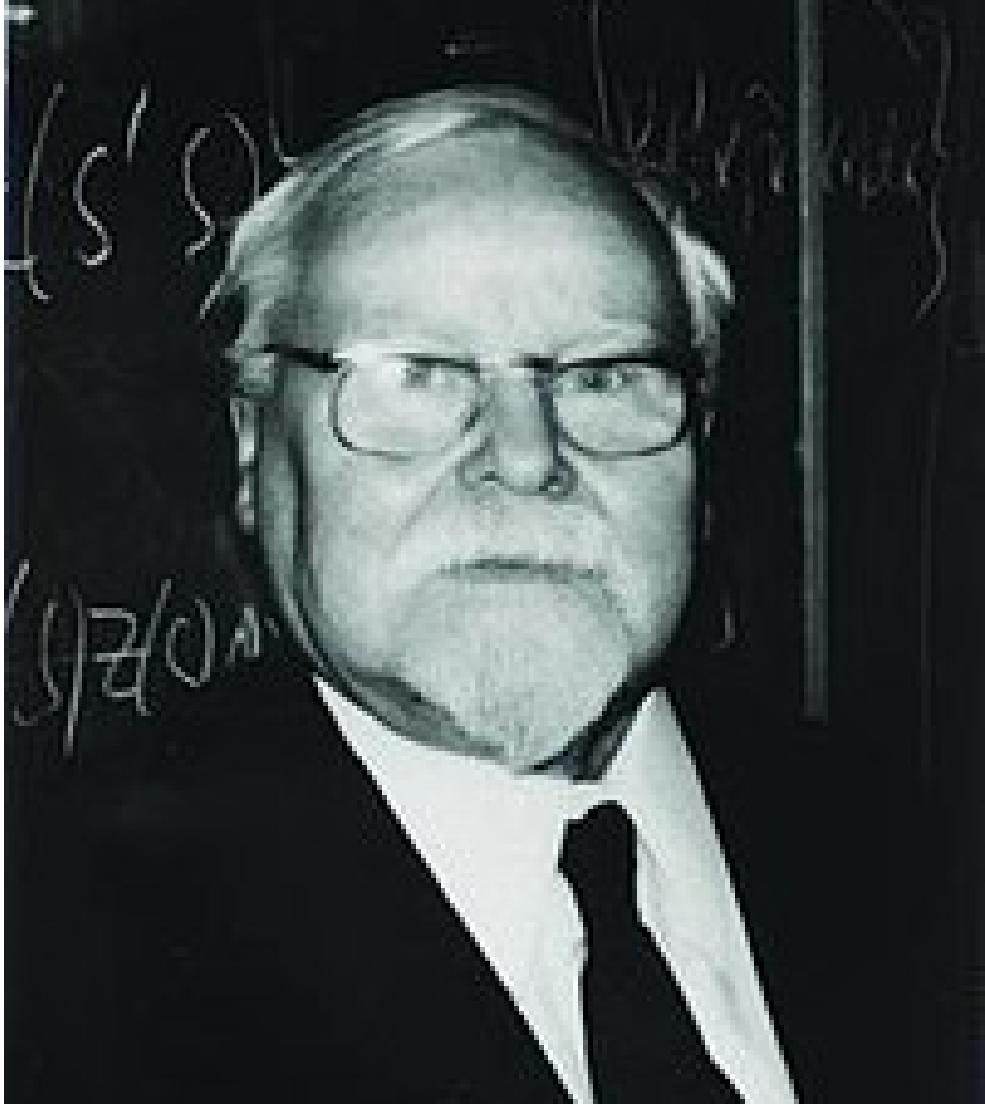
# Regularization

- **Core Idea:** Regularization adds a penalty to your loss function (or likelihood) when you go to estimate your parameters that helps to fight against overfitting
- **Downside:** Induces some bias in your parameter estimates
  - This means you technically get "the wrong answer" when you go to interpret coefficients
  - You can control how much bias you introduce, however!

# Regularization

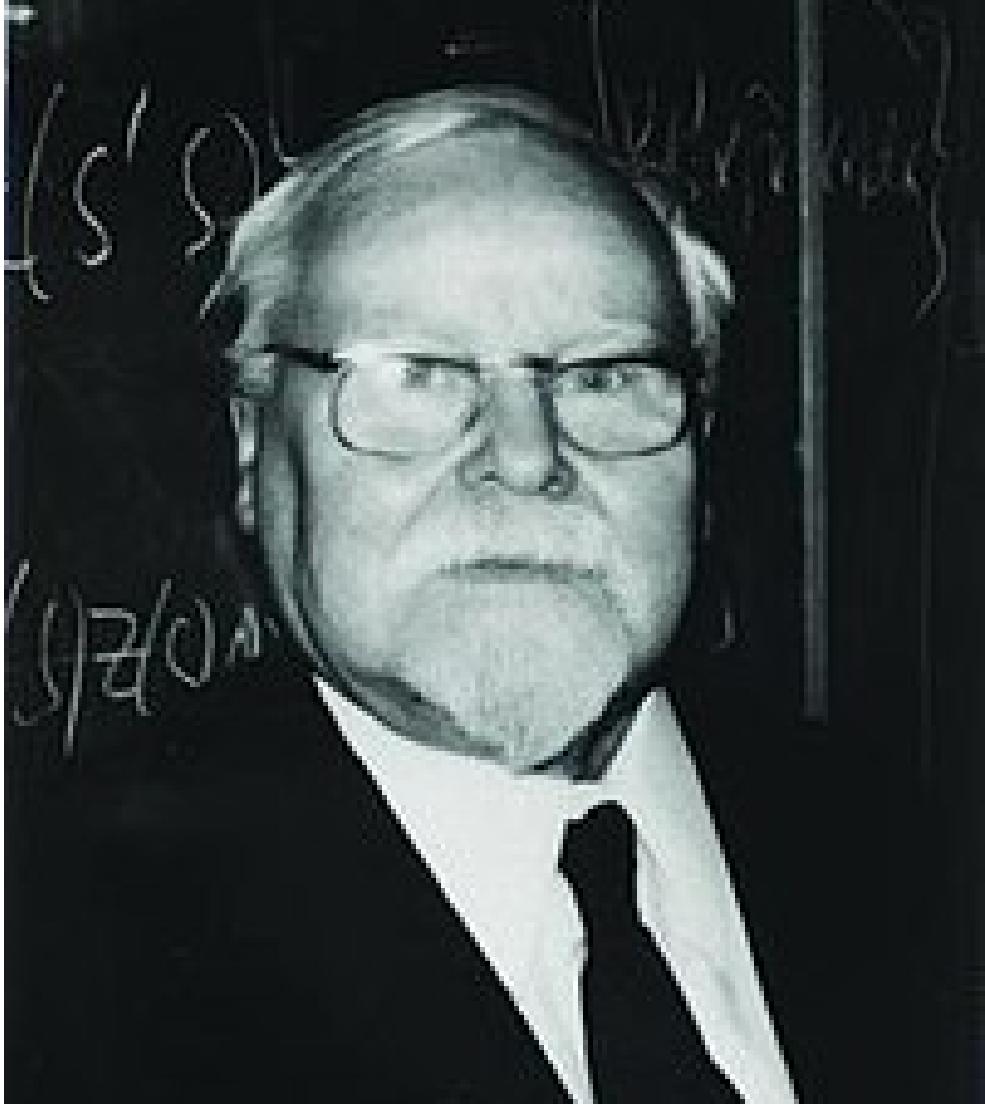
- **Core Idea:** Regularization adds a penalty to your loss function (or likelihood) when you go to estimate your parameters that helps to fight against overfitting
- Downside: Induces some bias in your parameter estimates
  - This means you technically get "the wrong answer" when you go to interpret coefficients
  - You can control how much bias you introduce, however!
- Upsides: Lots!
  - Helps models generalize to OOS data
  - Useful when you have multicollinearity
  - Can allow you to estimate models that would be otherwise unidentified
  - Easy to implement
  - Works with (almost) any type of model

Andrey Tikhonov  
(1906-1993)



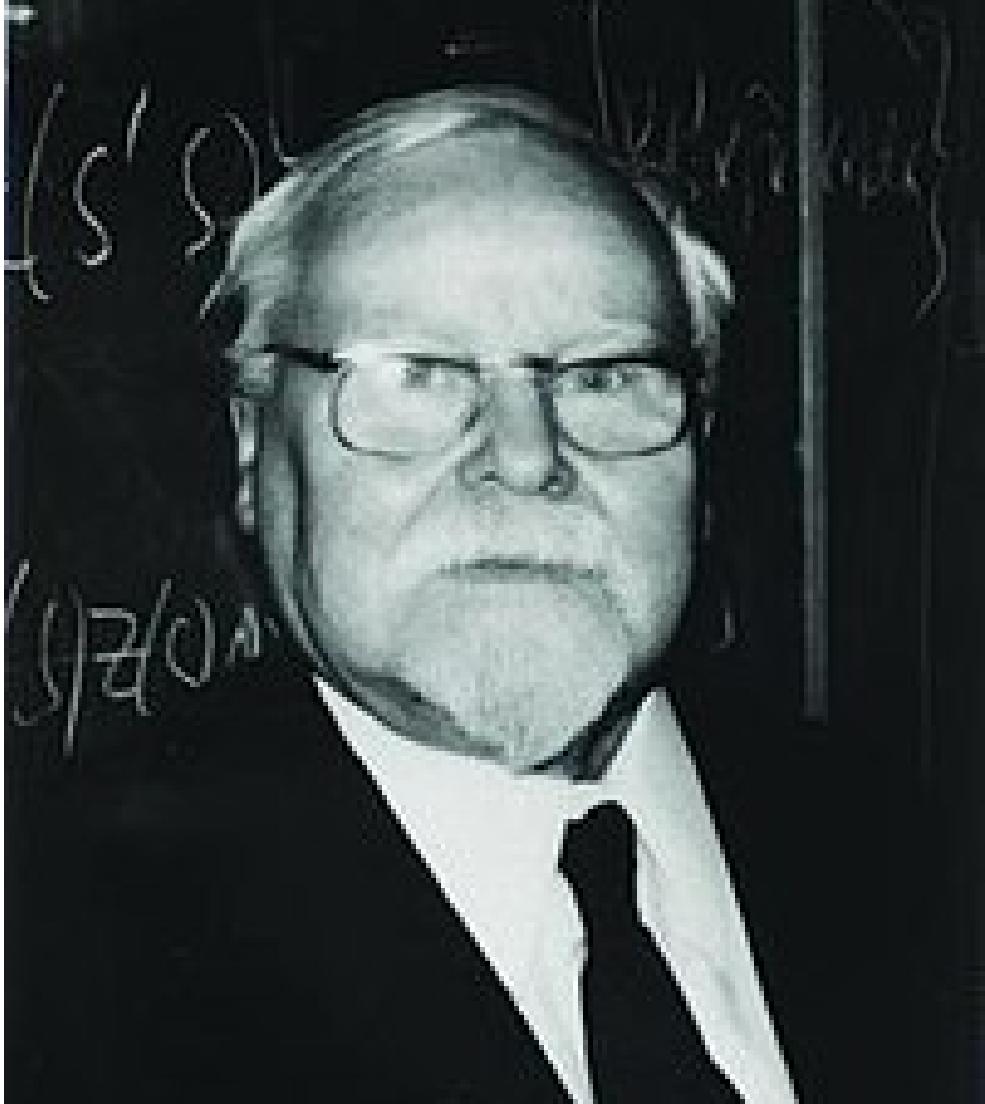
# Andrey Tikhonov (1906-1993)

- Soviet mathematician



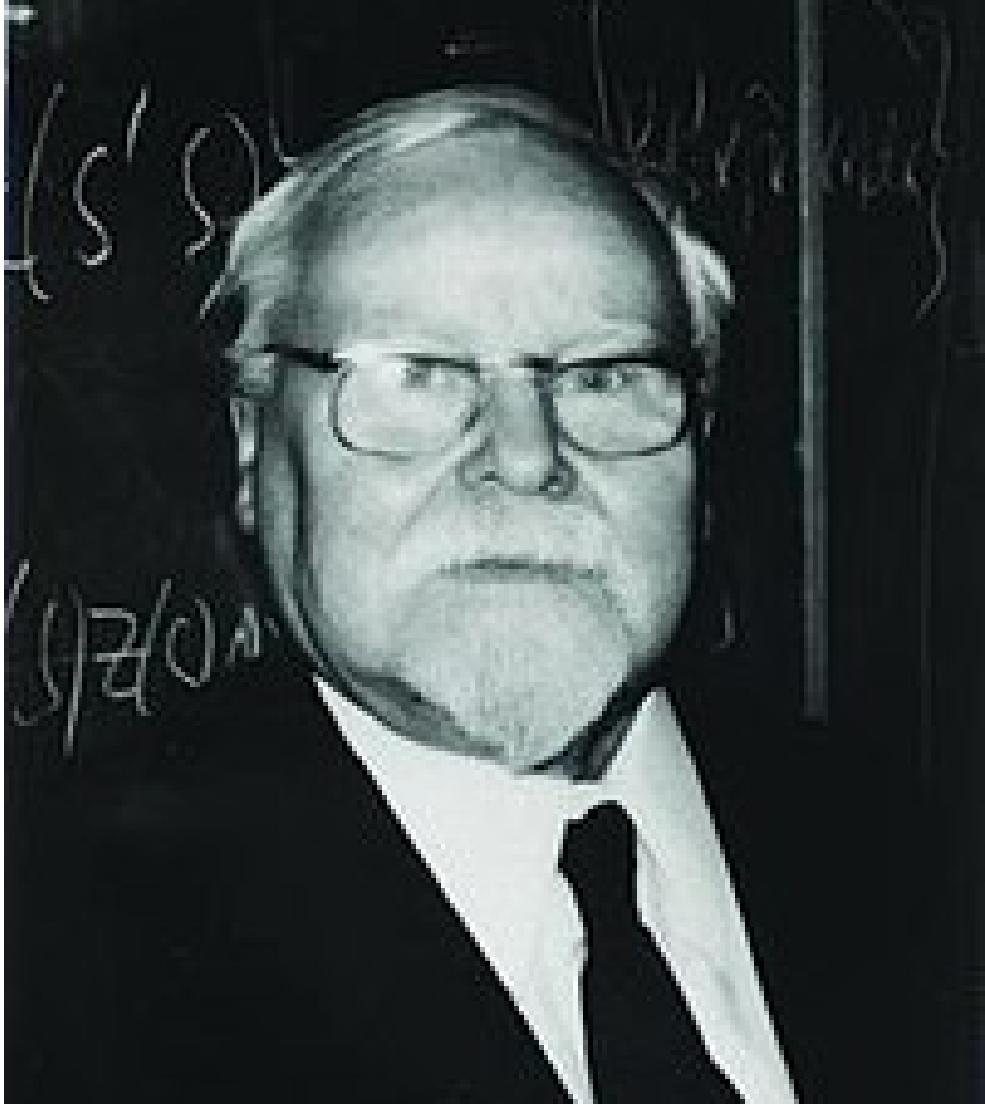
# Andrey Tikhonov (1906-1993)

- Soviet mathematician
- Active in topology and mathematical physics  
(among other things)



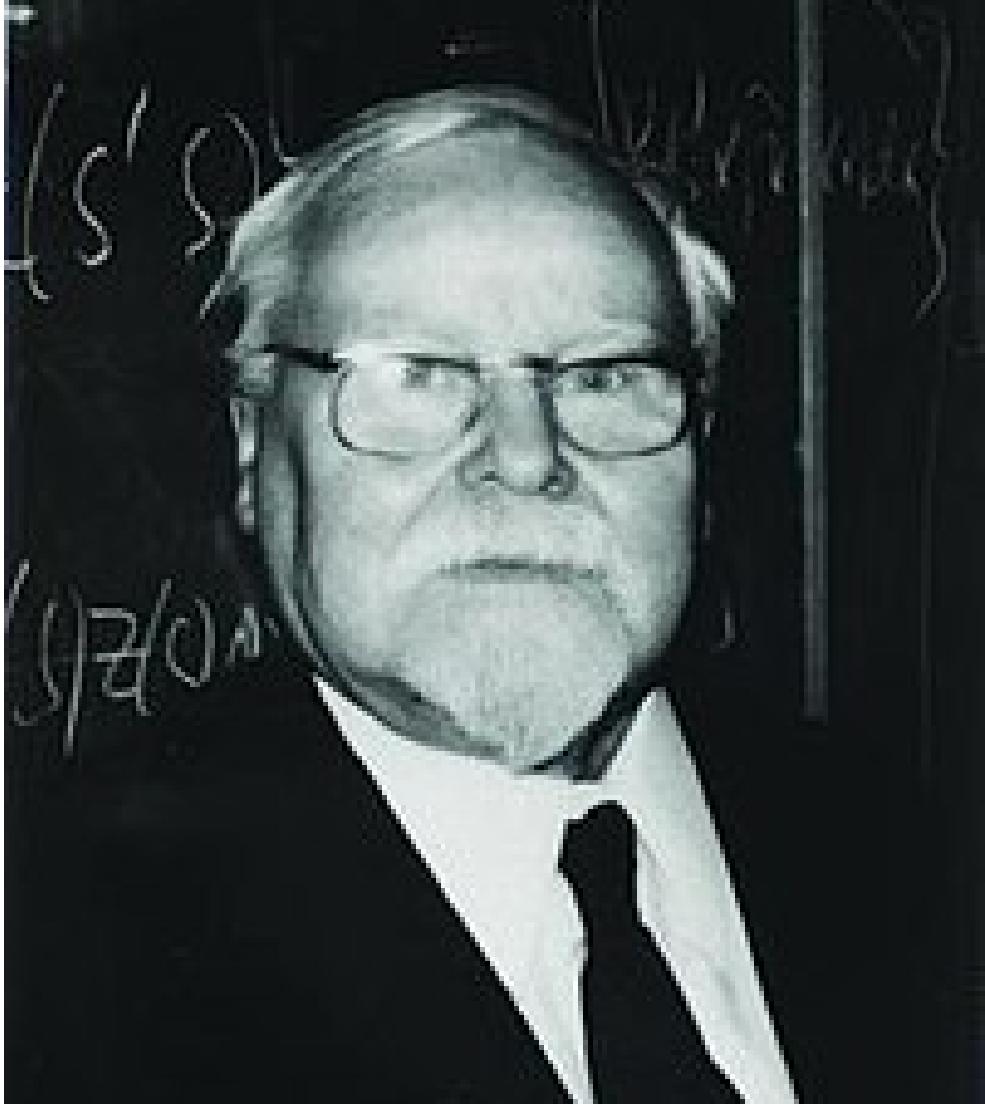
# Andrey Tikhonov (1906-1993)

- Soviet mathematician
- Active in topology and mathematical physics  
(among other things)
- Recipient of the Lenin Prize (1966)



# Andrey Tikhonov (1906-1993)

- Soviet mathematician
- Active in topology and mathematical physics  
(among other things)
- Recipient of the Lenin Prize (1966)
- Twice named Hero of Socialist Labor (1954,  
1986)



# Ridge Regression

- **Core Idea:** We will conduct a linear regression, but add a penalty proportional to the  $L^2$  norm of the vector of coefficients,  $\beta$
- This type of penalty is called the Ridge Penalty, Ridge Regularization, or  $L^2$  Regularization
- Not just useful for linear regressions!
- Recall the OLS loss function that minimizes the squared error:

$$\hat{\beta} = \operatorname{argmin}_{\beta} \sum_{\mathbf{x}_i, y_i \in \mathbf{X}} (y_i - \mathbf{x}_i \beta)^2$$

# Ridge Regression

- **Core Idea:** We will conduct a linear regression, but add a penalty proportional to the  $L^2$  norm of the vector of coefficients,  $\beta$
- This type of penalty is called the Ridge Penalty, Ridge Regularization, or  $L^2$  Regularization
- Not just useful for linear regressions!
- Recall the OLS loss function that minimizes the squared error:

$$\hat{\beta} = \operatorname{argmin}_{\beta} \sum_{\mathbf{x}_i, y_i \in \mathbf{X}} (y_i - \mathbf{x}_i \beta)^2$$

- Now we add the Ridge penalty:

$$\hat{\beta} = \operatorname{argmin}_{\beta} \sum_{\mathbf{x}_i, y_i \in \mathbf{X}} (y_i - \mathbf{x}_i \beta)^2 + \underbrace{\lambda \|\beta\|_2^2}_{\text{Ridge Penalty}}$$

- Note that minimizing the SSE is the same as minimizing the MSE!

# Ridge Regression

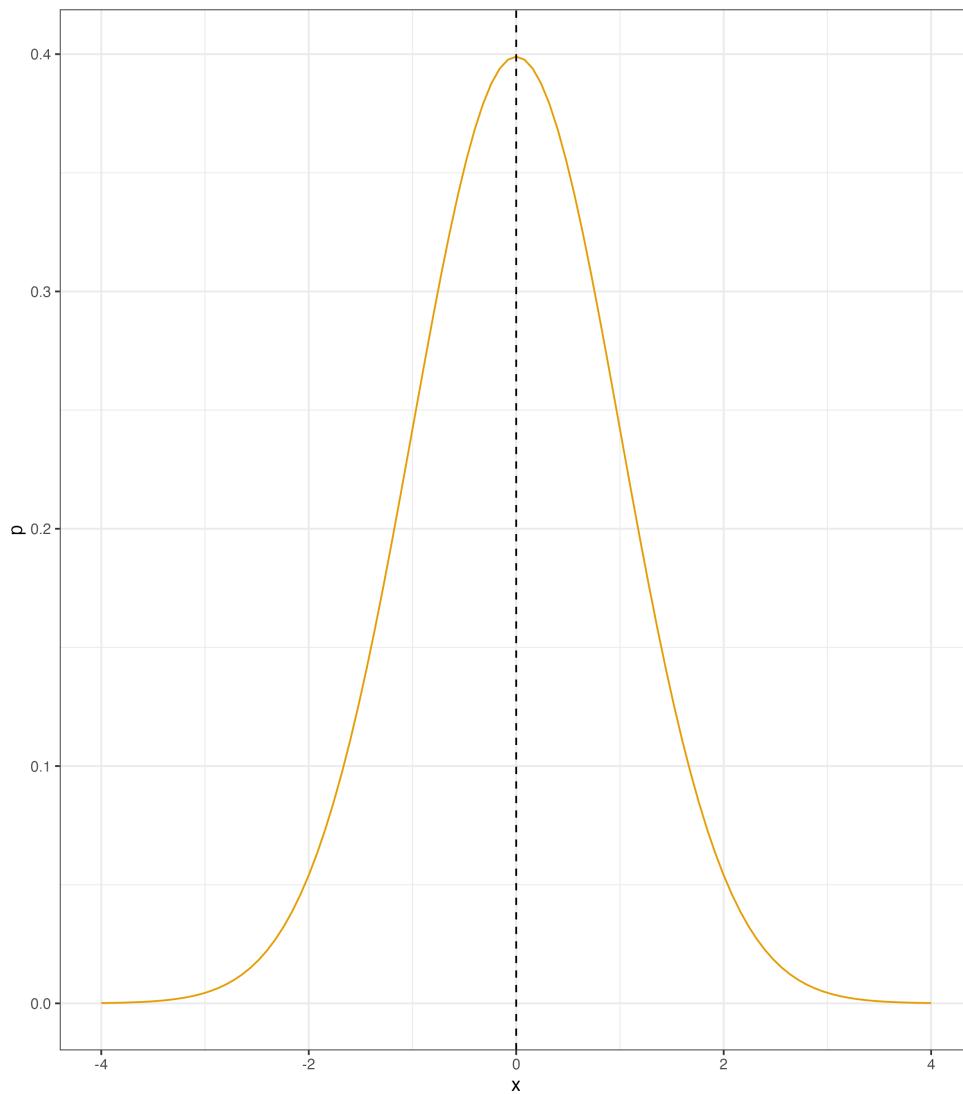
- Recall the loss function:

$$\hat{\beta} = \underset{\beta}{\operatorname{argmin}} \sum_{\mathbf{x}_i, y_i \in \mathbf{X}} (y_i - \mathbf{x}_i \beta)^2 + \underbrace{\lambda \|\beta\|_2^2}_{\text{Ridge Penalty}}$$

- So what happens here?
  - The left part of the argmin tries to minimize the MSE
  - The right part tries to minimize the length of the vector  $\beta$
- So who wins?
  - It depends on  $\lambda$ , often called the *regularization parameter*
  - If  $\lambda = 0$ , there is no regularization, and we recover OLS exactly
  - For  $\lambda > 0$ , the optimization problem has to consider the size of  $\beta$
  - As  $\lambda \rightarrow \infty$ , the only thing that matters is reducing the size of  $\beta$  until  $\|\beta\|_2 = 0$
  - The larger  $\lambda$  is, the more  $\beta$  is shrunk toward zero
  - This is how you control how much regularization (and how much bias) you induce

# The Ridge Penalty Puts a Normal Prior on $\beta$

- We think that the elements of  $\beta$  should be normally distributed
- They are *most likely* to be *nearly* zero
- The value of  $\lambda$  controls how much shrinkage toward zero is experienced
- Another way to think about  $\lambda$  is that it controls the *variance* of the normal prior
- When you get to multilevel models, think of random effects as similar to regularization!
- Typically, we pick  $\lambda$  to maximize OOS predictive performance, either by holding out a *test set* or through cross-validation



# One Quick Note on Maximizing a Penalized Likelihood

- This is how we minimize squared error:

$$\hat{\beta} = \underset{\beta}{\operatorname{argmin}} \sum_i (y_i - \mathbf{x}_i \beta)^2 + \lambda \|\beta\|_2^2$$

- This is how we maximize the log likelihood:

$$\hat{\beta} = \underset{\beta}{\operatorname{argmax}} - \sum_i (y_i - \mathbf{x}_i \beta)^2 - \lambda \|\beta\|_2^2$$

# What about other types of regularization?

- Remember that we said overfitting occurs if your parameters are too big or if you have too many
- Ridge only solves the first problem. What about the second?
- Turns out, we can do this using Least Absolute Shrinkage and Selection Operator (LASSO)
- This is implemented through another penalized loss function

# Robert Tibshirani (born 1956)

- Professor of Statistics at Stanford University
- Student of Bradley Efron
- Invented Generalized Additive Models (GAMs)
- Wrote *The Elements of Statistical Learning*
- Wrote *An Introduction to Statistical Learning*



# LASSO Regression

- **Core Idea:** We will conduct a linear regression, but add a penalty proportional to the  $L^1$  norm of the vector of coefficients,  $\beta$
- This type of penalty is called the LASSO Penalty, LASSO Regularization, or  $L^1$  Regularization
- Still not just useful for linear regressions!
- Recall the OLS loss function that minimizes the squared error:

$$\hat{\beta} = \operatorname{argmin}_{\beta} \sum_{\mathbf{x}_i, y_i \in \mathbf{X}} (y_i - \mathbf{x}_i \beta)^2$$

- Now we add the LASSO penalty:

$$\hat{\beta} = \operatorname{argmin}_{\beta} \sum_{\mathbf{x}_i, y_i \in \mathbf{X}} (y_i - \mathbf{x}_i \beta)^2 + \underbrace{\lambda \|\beta\|_1}_{\text{LASSO Penalty}}$$

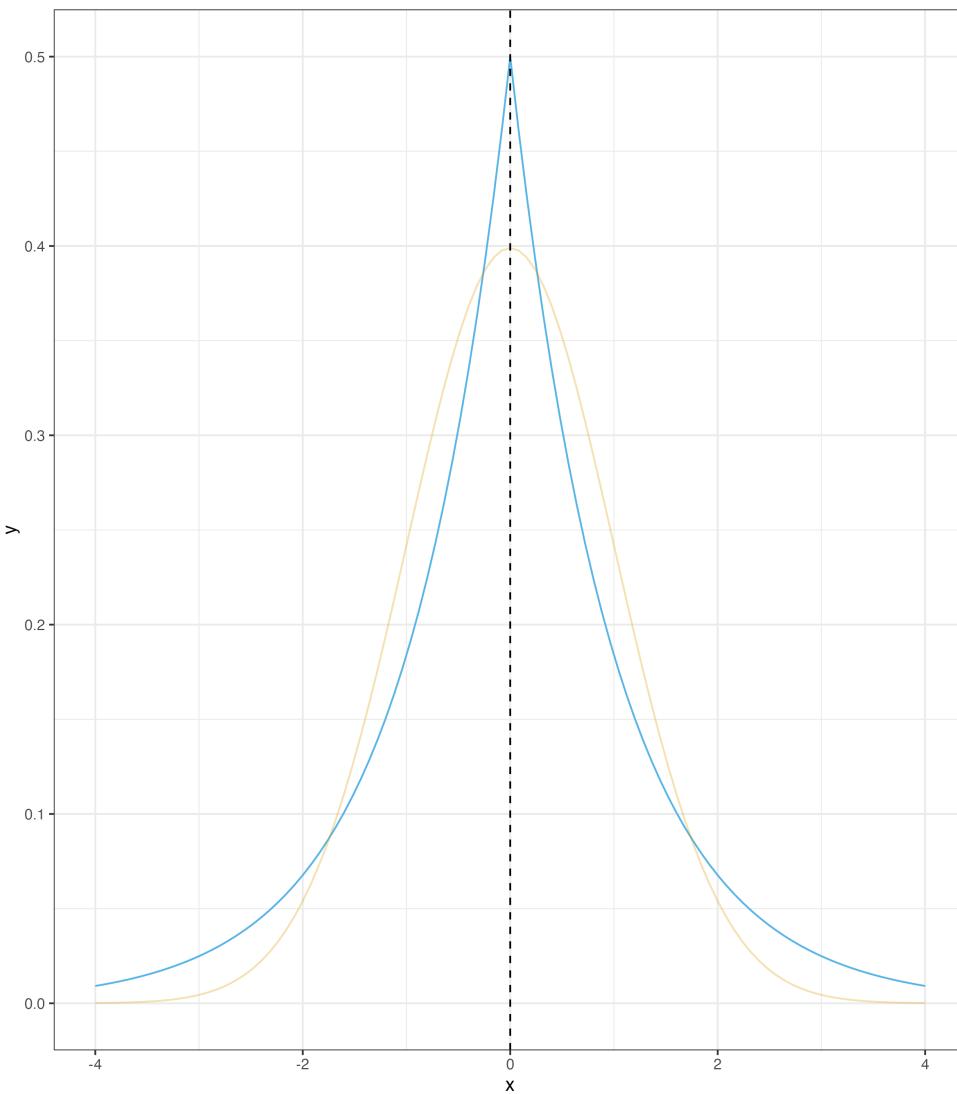
# LASSO Regression

$$\hat{\beta} = \underset{\beta}{\operatorname{argmin}} \sum_{\mathbf{x}_i, y_i \in \mathbf{X}} (y_i - \mathbf{x}_i \beta)^2 + \underbrace{\lambda \|\beta\|_1}_{\text{LASSO Penalty}}$$

- So what happens here?
  - The left part of the argmin tries to minimize the MSE
  - The right part tries to minimize the length of the vector  $\beta$ , but in kind of a different way from the Ridge penalty!
- So what changes?
  - Note that as  $\beta, \epsilon \rightarrow 0$ ,  $|\beta + \epsilon| - |\beta| > (\beta + \epsilon)^2 - \beta^2$
  - This gives a loss function with a LASSO penalty extra incentive to force some coefficients to be zero!
  - The Ridge penalty will uniformly shrink coefficients toward zero, while the LASSO penalty will force some number of coefficients to be *precisely* zero
  - This effectively drops variables with less predictive value from your model (which is the "selection" part of LASSO)
  - Higher values of  $\lambda$  will drop more variables

# LASSO Penalty Puts a Laplace Prior on $\beta$

- This means we think that the elements of  $\beta$  should be Laplace distributed
- This means they are *most likely* to be *exactly* zero, but with enough evidence they can take on any value
- The value of  $\lambda$  controls how aggressively coefficients are forced to be zero
- Again, we pick the  $\lambda$  value to maximize OOS predictive performance, either by holding out a *test set* of some data or through cross-validation



# More Resampling Methods

# The Central Limit Theorem

- Core to statistics, because it says "normal distributions are worth understanding" even though most data is not actually normally distributed!
- $\{X_1, X_2, \dots, X_n\}$  are i.i.d. random variables with:
  - $E[X_i] = \mu$
  - $\text{Var}[X_i] = \sigma^2$
- We define the *sample mean*:  $\bar{X}_n = \frac{1}{n} \sum_{i=1}^n X_i$

# The Central Limit Theorem

- As  $n$  grows large, the *distribution of sample means* converges to:

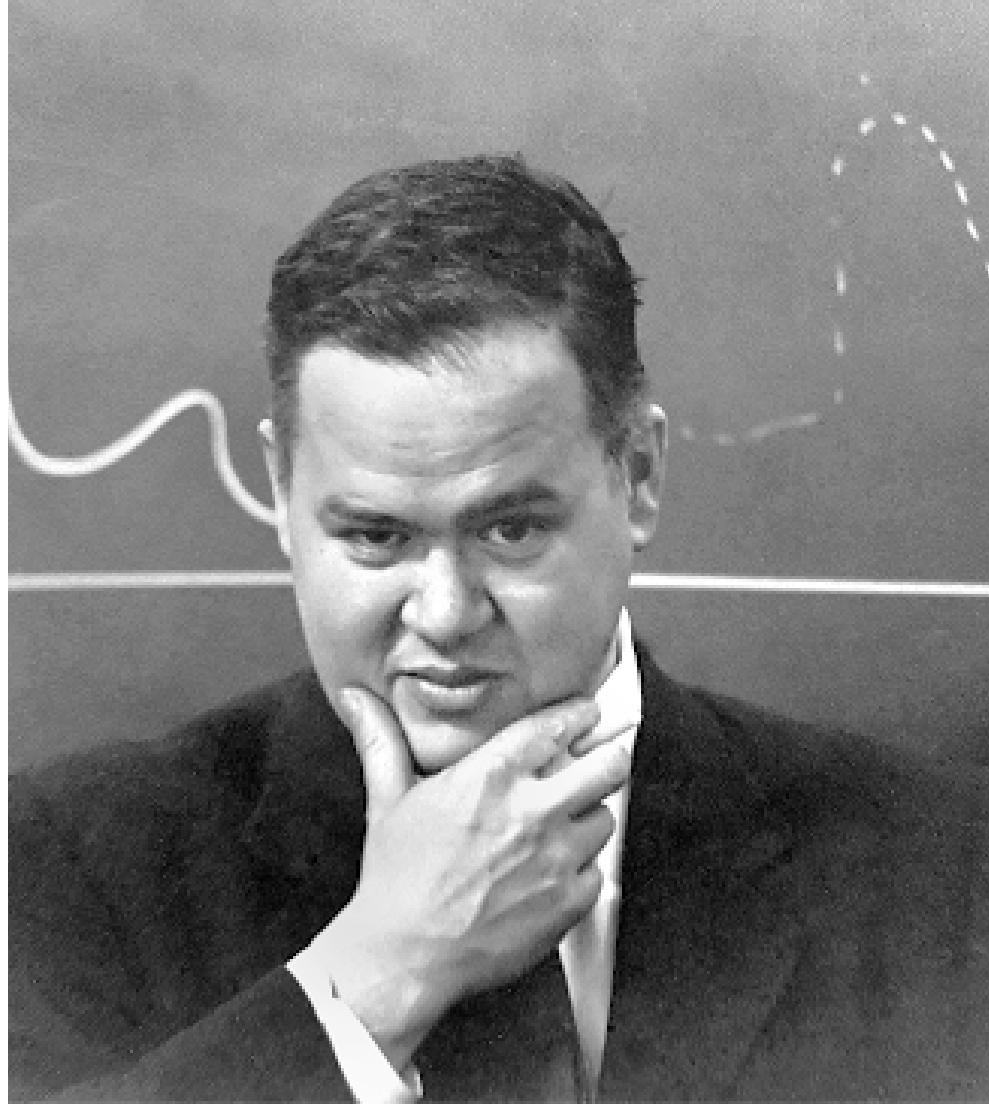
$$\bar{X}_i \sim \mathcal{N}\left(\mu, \frac{\sigma^2}{n}\right)$$

- This holds even if  $X_i$  is not normally distributed!
- This only applies to sample means!

# Resampling Methods

- We want to reason about populations from samples
- The CLT helps a lot, but it doesn't do everything!
- **Core idea:** The distribution of the data you have is your best guess at the distribution of the population
- But what do we do with this?

John Tukey (1915-  
2000)



# John Tukey (1915-2000)

- His ideas about “exploratory data analysis” are the foundations of modern data science



# John Tukey (1915-2000)

- His ideas about “exploratory data analysis” are the foundations of modern data science
- His book “Exploratory Data Analysis” is still one of the best sources for data visualization and exploratory analysis



# John Tukey (1915-2000)

- His ideas about “exploratory data analysis” are the foundations of modern data science
- His book “Exploratory Data Analysis” is still one of the best sources for data visualization and exploratory analysis
- Invented the term “bit” at Bell Labs



# John Tukey (1915-2000)

- His ideas about “exploratory data analysis” are the foundations of modern data science
- His book “Exploratory Data Analysis” is still one of the best sources for data visualization and exploratory analysis
- Invented the term “bit” at Bell Labs
- Invented the box plot



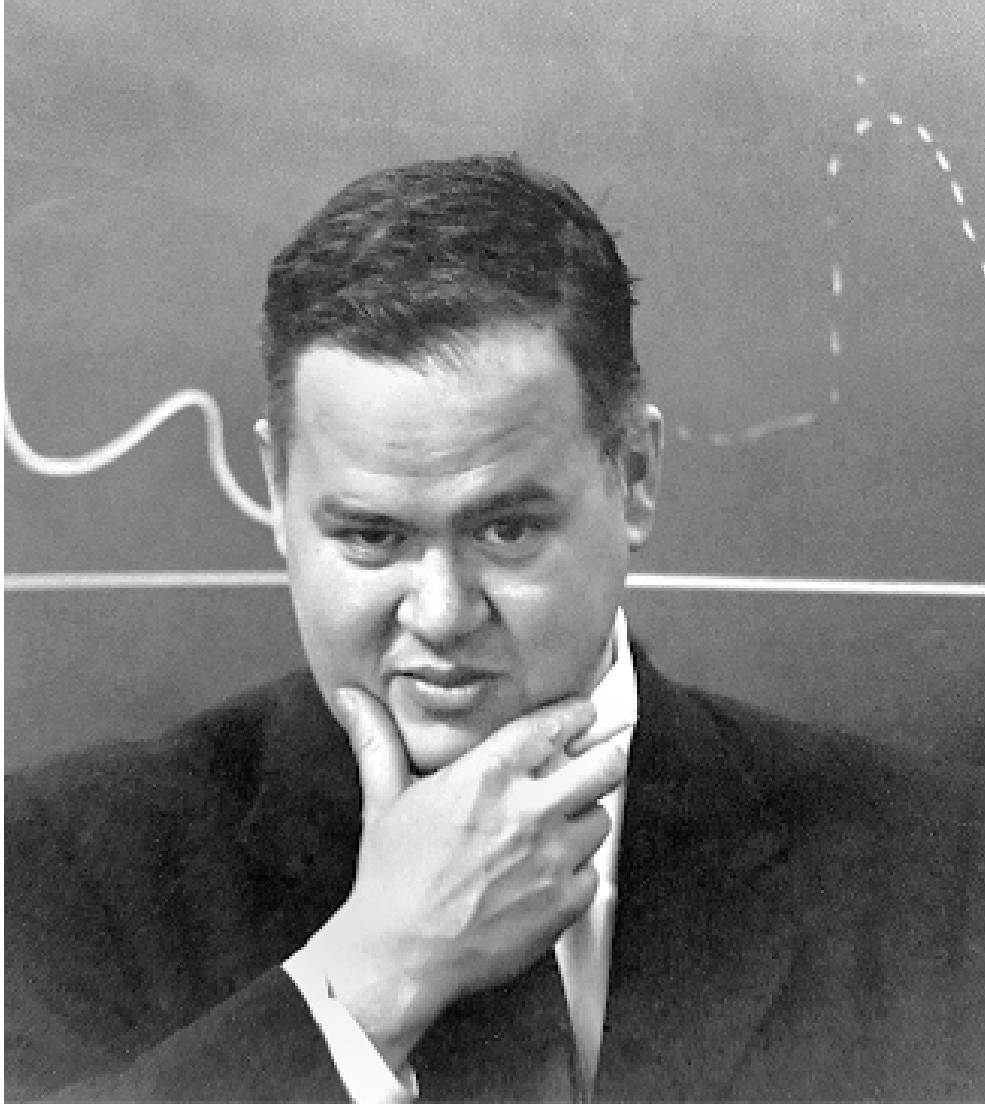
# John Tukey (1915-2000)

- His ideas about “exploratory data analysis” are the foundations of modern data science
- His book “Exploratory Data Analysis” is still one of the best sources for data visualization and exploratory analysis
- Invented the term “bit” at Bell Labs
- Invented the box plot
- Invented the Fast Fourier Transform



# John Tukey (1915-2000)

- His ideas about “exploratory data analysis” are the foundations of modern data science
- His book “Exploratory Data Analysis” is still one of the best sources for data visualization and exploratory analysis
- Invented the term “bit” at Bell Labs
- Invented the box plot
- Invented the Fast Fourier Transform
- Certified badass



# What did Tukey give us today?



# The Jackknife

- Thanks Tukey!
- "Rough and ready" tool to solve statistical problems when the exact solution might be hard or unknown
- **Core idea:** We take our data, make new datasets by dropping single observations, compute our statistic of interest, and then look at the distribution

# The Jackknife

- We have  $\{X_1, X_2, \dots, X_n\}$ , which are i.i.d. samples from some population
- We care about some estimator or statistic in the population,  $f(X_i)$  We often think that:  $E[f(X_i)] = f(\{X_1, X_2, \dots, X_n\})$
- But this tells us nothing about the bias or variance of that estimator!

# The Jackknife

- From your dataset of length  $n$ , create  $n$  new resampled datasets of length  $n - 1$ , each missing one observation
- Compute your estimator or statistic on each resampled dataset
- Aggregate these statistics to get your jackknife estimate
  - The mean of the estimator in jackknife samples can be used to understand bias
  - The variance of the estimator in jackknife samples can be used to quantify uncertainty
  - Does not require the CLT to apply to the estimator!



# The Bootstrap

# The Bootstrap

# The Bootstrap

- **Resampling's Core Idea:** The distribution of the data you have is your best guess at the distribution of the population

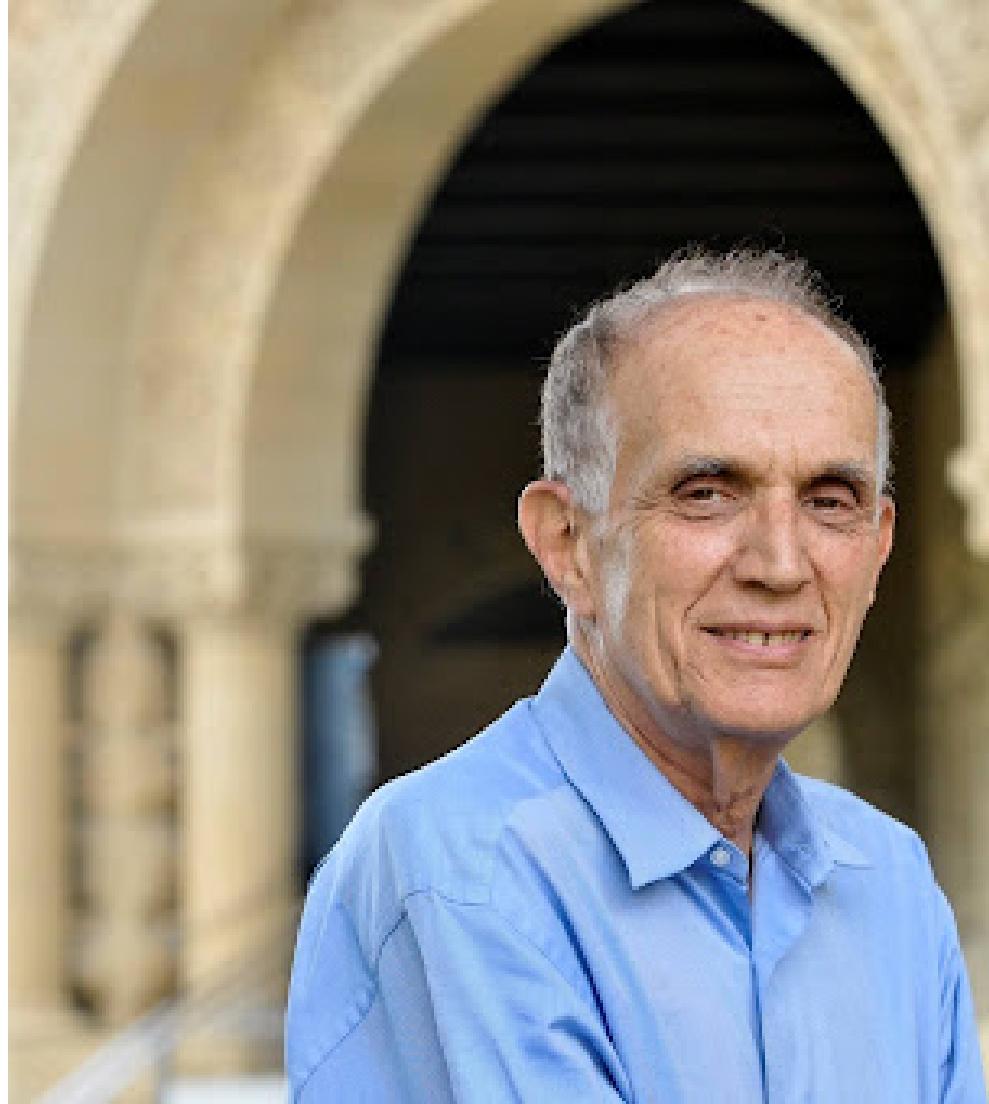
# The Bootstrap

- **Resampling's Core Idea:** The distribution of the data you have is your best guess at the distribution of the population
- **Jackknife's Core Idea:** We take our data, make new datasets by dropping single observations, compute our statistic of interest, and then look at the distribution

# The Bootstrap

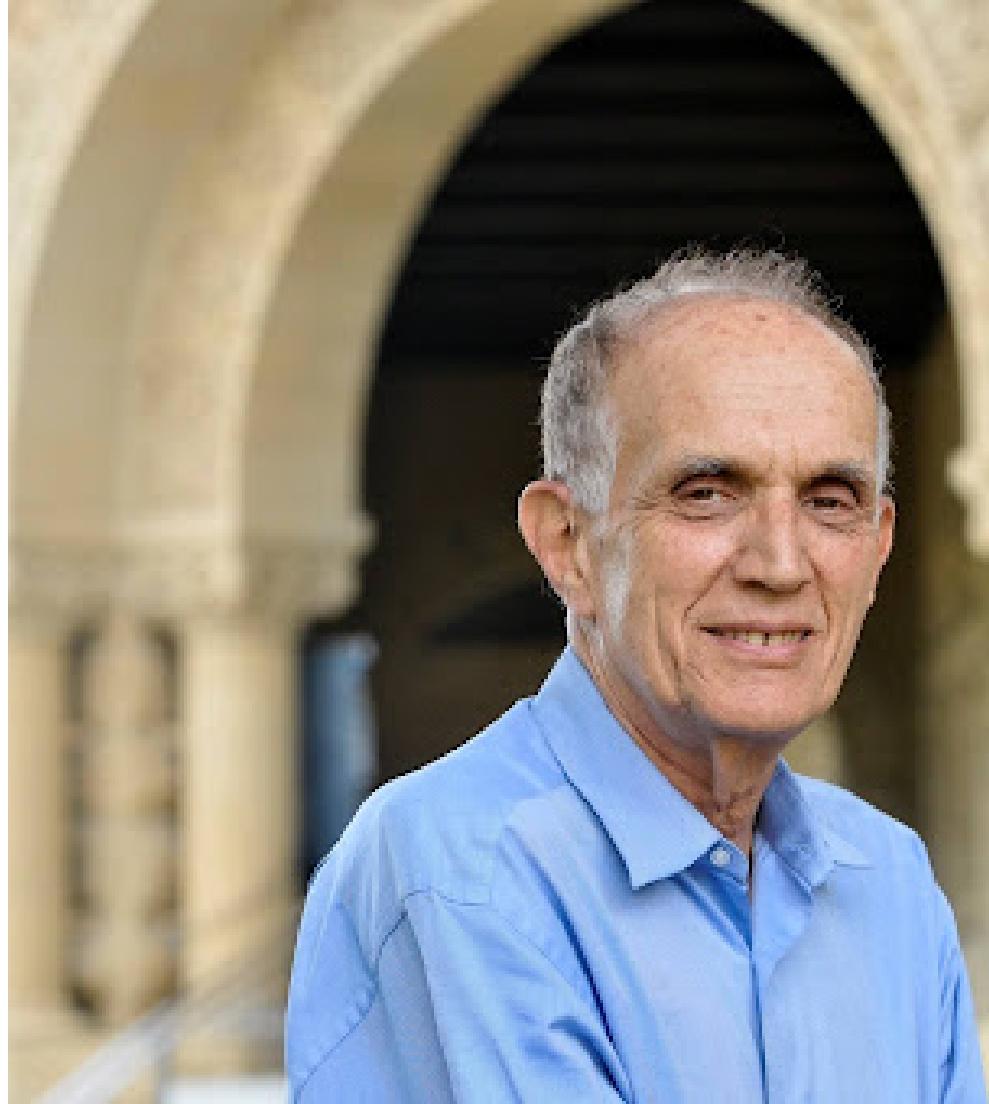
- **Resampling's Core Idea:** The distribution of the data you have is your best guess at the distribution of the population
- **Jackknife's Core Idea:** We take our data, make new datasets by dropping single observations, compute our statistic of interest, and then look at the distribution
- **Bootstrap's Core Idea:** We take our data and make new datasets by resampling *with replacement* from our original dataset to approximate the population. Then we compute our statistic of interest on each resampled dataset and look at the distribution

Bradley Efron (born  
1938)



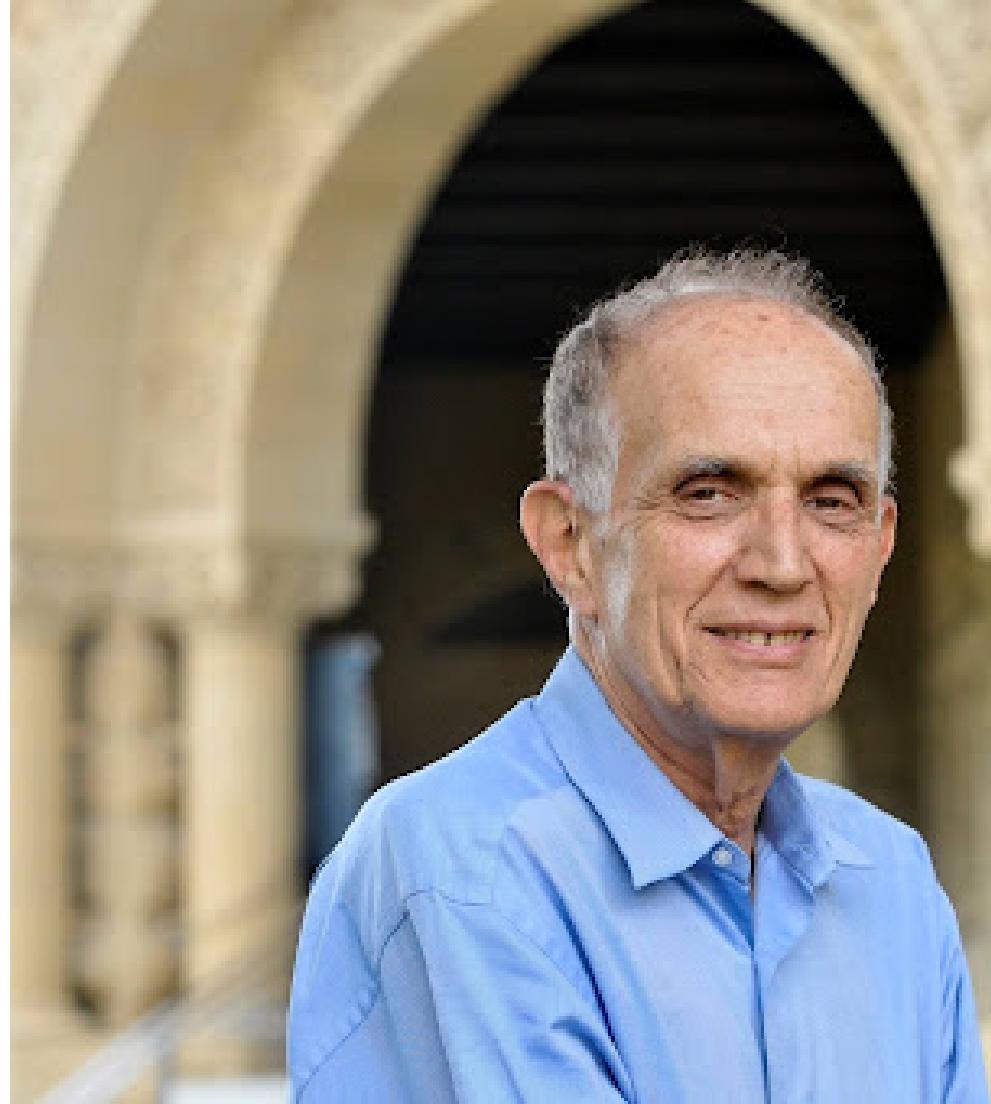
# Bradley Efron (born 1938)

- Invented the bootstrap in 1982



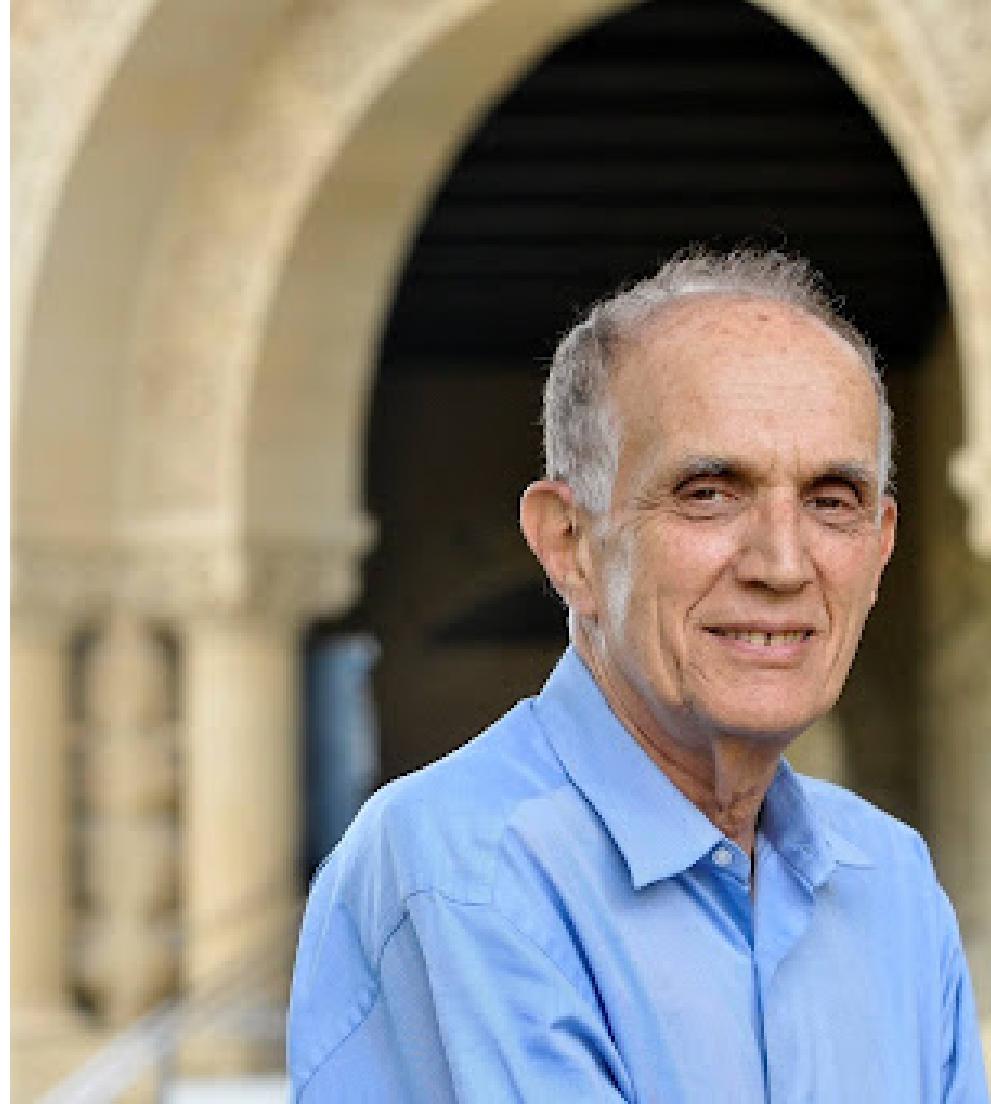
# Bradley Efron (born 1938)

- Invented the bootstrap in 1982
- One of the first major computationally intensive statistical techniques



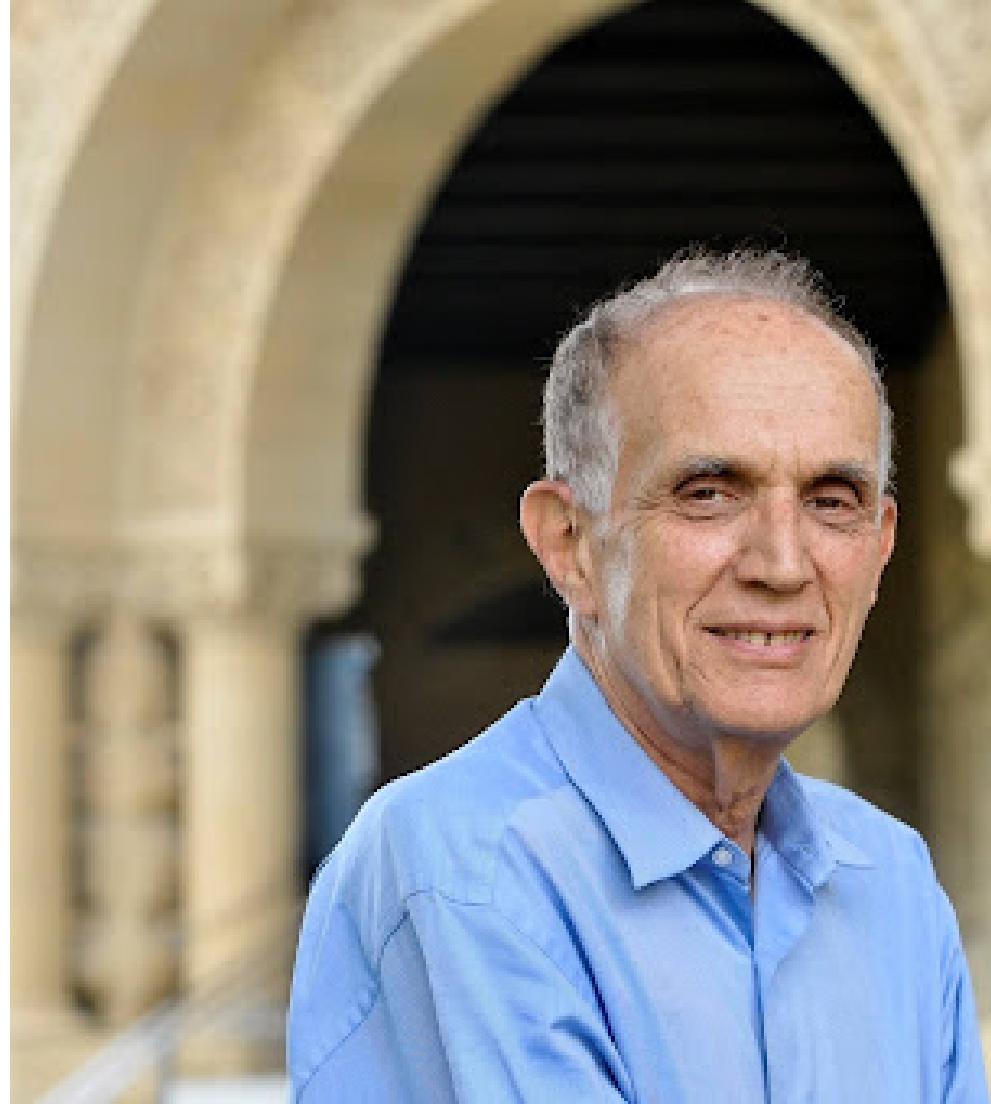
# Bradley Efron (born 1938)

- Invented the bootstrap in 1982
- One of the first major computationally intensive statistical techniques
- His list of awards is insane
  - MacArthur Prize Fellowship
  - Membership in the National Academy of Sciences and the American Academy of Arts and Sciences
  - Fellowship in the Institute of Mathematical Statistics (IMS) and the American Statistical Association (ASA)
  - The Lester R. Ford Award, the Wilks Medal, the Parzen Prize, and the Rao Prize



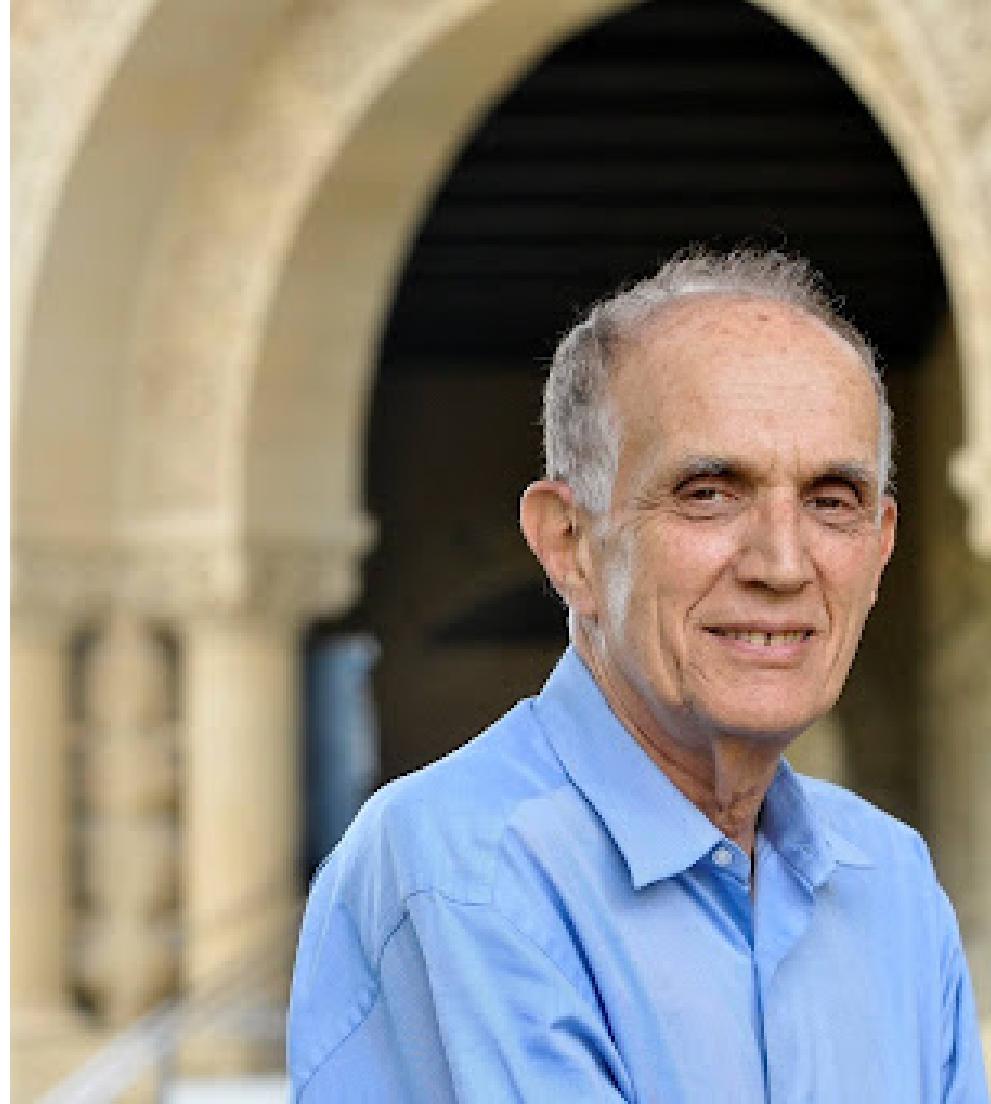
# Bradley Efron (born 1938)

- Invented the bootstrap in 1982
- One of the first major computationally intensive statistical techniques
- His list of awards is insane
  - MacArthur Prize Fellowship
  - Membership in the National Academy of Sciences and the American Academy of Arts and Sciences
  - Fellowship in the Institute of Mathematical Statistics (IMS) and the American Statistical Association (ASA)
  - The Lester R. Ford Award, the Wilks Medal, the Parzen Prize, and the Rao Prize



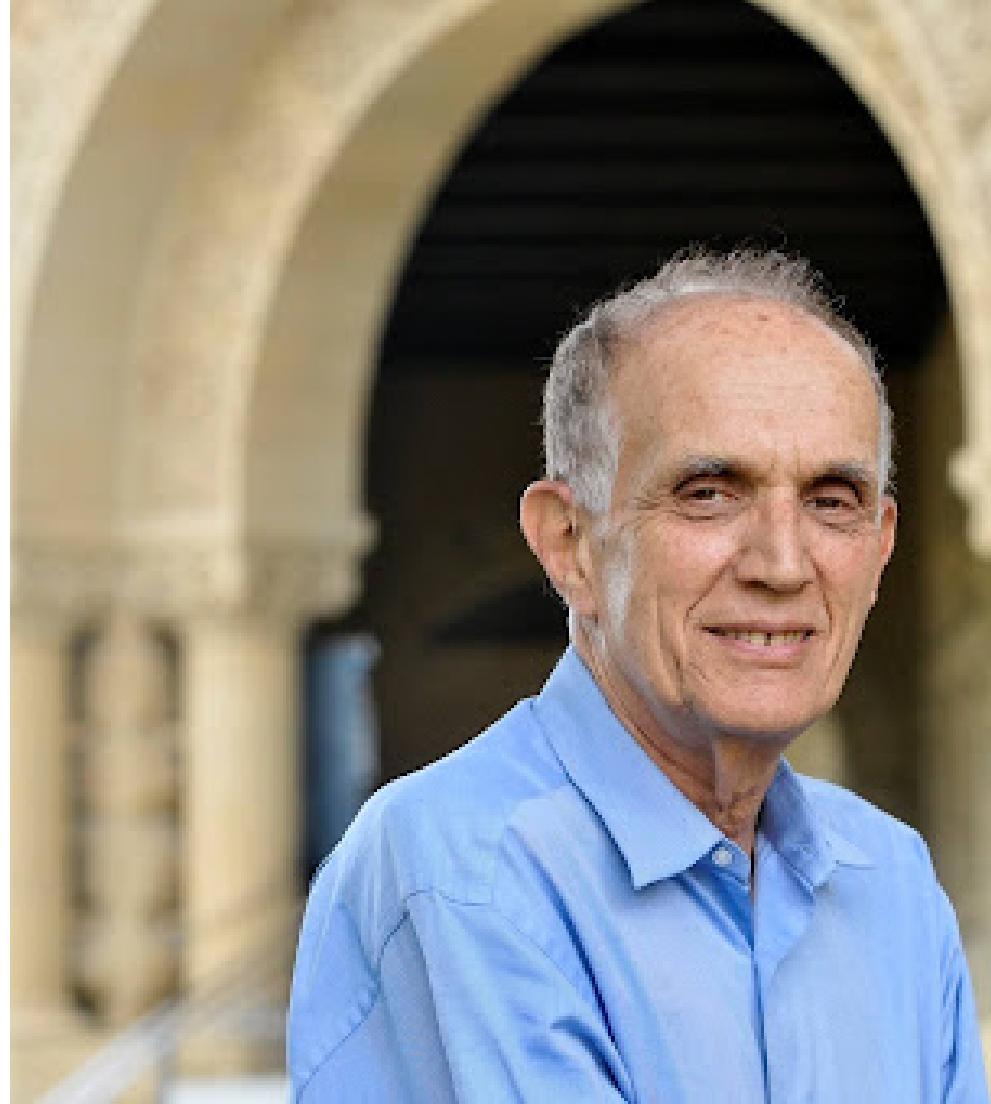
# Bradley Efron (born 1938)

- Invented the bootstrap in 1982
- One of the first major computationally intensive statistical techniques
- His list of awards is insane
  - MacArthur Prize Fellowship
  - Membership in the National Academy of Sciences and the American Academy of Arts and Sciences
  - Fellowship in the Institute of Mathematical Statistics (IMS) and the American Statistical Association (ASA)
  - The Lester R. Ford Award, the Wilks Medal, the Parzen Prize, and the Rao Prize



# Bradley Efron (born 1938)

- Invented the bootstrap in 1982
- One of the first major computationally intensive statistical techniques
- His list of awards is insane
  - MacArthur Prize Fellowship
  - Membership in the National Academy of Sciences and the American Academy of Arts and Sciences
  - Fellowship in the Institute of Mathematical Statistics (IMS) and the American Statistical Association (ASA)
  - The Lester R. Ford Award, the Wilks Medal, the Parzen Prize, and the Rao Prize



# The Bootstrap

- It works for basically anything!
- Your data describes an *empirical distribution function* - this is an approximation of the cumulative distribution function (CDF)
- The empirical distribution function (eCDF) converges to the CDF
- The bootstrap uses resampling with replacement to build new eCDFs
- Because we resample with replacement, we can have many more eCDFs *with the same sample size as the original data* than the jackknife
- Data are guaranteed to be drawn from the population of interest!
- Often used for bias, variance, and constructing confidence intervals
- Does not require the CLT to apply to your estimator!

# How to Bootstrap

1. Compute your estimate in the full sample with  $N$  observations
2. Draw  $M$  bootstrap samples, each of length  $N$ , from the original dataset *with replacement*
3. Compute your estimate on each bootstrap sample
4. Use the distribution of bootstrap estimates to:
  - Check the original estimate for bias (using the mean value)
  - Find the variance of a standard error of the estimate (using the variance or sd)
  - Construct a confidence interval (by looking at the end points of the middle 95% of the distribution of bootstrap estimates)

# Applied Bootstrapping

```
1 d <- data.frame(x = rnorm(1e2))
2 d$y <- -1.5 + 3*d$x + rnorm(1e2)
3
4 m <- lm(y~x, d)
5
6
7 summary(m)$coefficients
```

# Applied Bootstrapping

```
1 d <- data.frame(x = rnorm(1e2))
2 d$y <- -1.5 + 3*d$x + rnorm(1e2)
3
4 m <- lm(y~x, d)
5
6
7 summary(m)$coefficients
8
9 #             Estimate Std. Error    t value   Pr(>|t|)
10 # (Intercept) -1.486122  0.09547956 -15.56482 3.078629e-28
11 # x            3.101451  0.09144205  33.91712 5.913505e-56
```

# Applied Bootstrapping

```
1 d <- data.frame(x = rnorm(1e2))
2 d$y <- -1.5 + 3*d$x + rnorm(1e2)
3
4 m <- lm(y~x, d)
5
6
7 summary(m)$coefficients
8
9 #             Estimate Std. Error    t value   Pr(>|t|)
10 # (Intercept) -1.486122  0.09547956 -15.56482 3.078629e-28
11 # x            3.101451  0.09144205  33.91712 5.913505e-56
12
13 coef(m)[2]
```

# Applied Bootstrapping

```
1 d <- data.frame(x = rnorm(1e2))
2 d$y <- -1.5 + 3*d$x + rnorm(1e2)
3
4 m <- lm(y~x, d)
5
6
7 summary(m)$coefficients
8
9 #             Estimate Std. Error    t value   Pr(>|t|)
10 # (Intercept) -1.486122 0.09547956 -15.56482 3.078629e-28
11 # x            3.101451 0.09144205  33.91712 5.913505e-56
12
13 coef(m)[2]
14
15 #           x
16 # 3.101451
```

# Applied Bootstrapping

1. Write a function that extracts the coefficient we care about
2. Then we add resampling
3. Run it a bunch of times

```
1 Beta1Boot <- function(d){  
2  
3 }
```

# Applied Bootstrapping

1. Write a function that extracts the coefficient we care about
2. Then we add resampling
3. Run it a bunch of times

```
1 Beta1Boot <- function(d){  
2   m <- lm(y~x, d)  
3 }
```

# Applied Bootstrapping

1. Write a function that extracts the coefficient we care about
2. Then we add resampling
3. Run it a bunch of times

```
1 Beta1Boot <- function(d){  
2   m <- lm(y~x, d)  
3   beta1 <- unname(coef(m)[2])  
4 }
```

# Applied Bootstrapping

1. Write a function that extracts the coefficient we care about
2. Then we add resampling
3. Run it a bunch of times

```
1 Beta1Boot <- function(d){  
2   m <- lm(y~x, d)  
3   beta1 <- unname(coef(m)[2])  
4   return(beta1)  
5 }
```

# Applied Bootstrapping

1. Write a function that extracts the coefficient we care about
2. Then we add resampling
3. Run it a bunch of times

```
1 Beta1Boot <- function(d){  
2   idx <- sample(nrow(d), replace=TRUE)  
3   m <- lm(y~x, d)  
4   beta1 <- unname(coef(m)[2])  
5   return(beta1)  
6 }
```

# Applied Bootstrapping

1. Write a function that extracts the coefficient we care about
2. Then we add resampling
3. Run it a bunch of times

```
1 Beta1Boot <- function(d){  
2   idx <- sample(nrow(d), replace=TRUE)  
3   m <- lm(y~x, d[idx,])  
4   beta1 <- unname(coef(m)[2])  
5   return(beta1)  
6 }
```

# Applied Bootstrapping

1. Write a function that extracts the coefficient we care about
2. Then we add resampling
3. Run it a bunch of times

```
1 Beta1Boot <- function(d){  
2   idx <- sample(nrow(d), replace=TRUE)  
3   m <- lm(y~x, d[idx,])  
4   beta1 <- unname(coef(m)[2])  
5   return(beta1)  
6 }  
7  
8 replicate(5, Beta1Boot(d))
```

# Applied Bootstrapping

1. Write a function that extracts the coefficient we care about
2. Then we add resampling
3. Run it a bunch of times

```
1 Beta1Boot <- function(d){  
2   idx <- sample(nrow(d), replace=TRUE)  
3   m <- lm(y~x, d[idx,])  
4   beta1 <- unname(coef(m)[2])  
5   return(beta1)  
6 }  
7  
8 replicate(5, Beta1Boot(d))  
9  
10 # [1] 2.930935 3.264935 3.177743 3.027659 2.913476
```

# Applied Bootstrapping

1. Write a function that extracts the coefficient we care about
2. Then we add resampling
3. Run it a bunch of times

```
1 Beta1Boot <- function(d){  
2   idx <- sample(nrow(d), replace=TRUE)  
3   m <- lm(y~x, d[idx,])  
4   beta1 <- unname(coef(m)[2])  
5   return(beta1)  
6 }  
7  
8 replicate(5, Beta1Boot(d))  
9  
10 # [1] 2.930935 3.264935 3.177743 3.027659 2.913476  
11  
12 N_boot <- 1e6
```

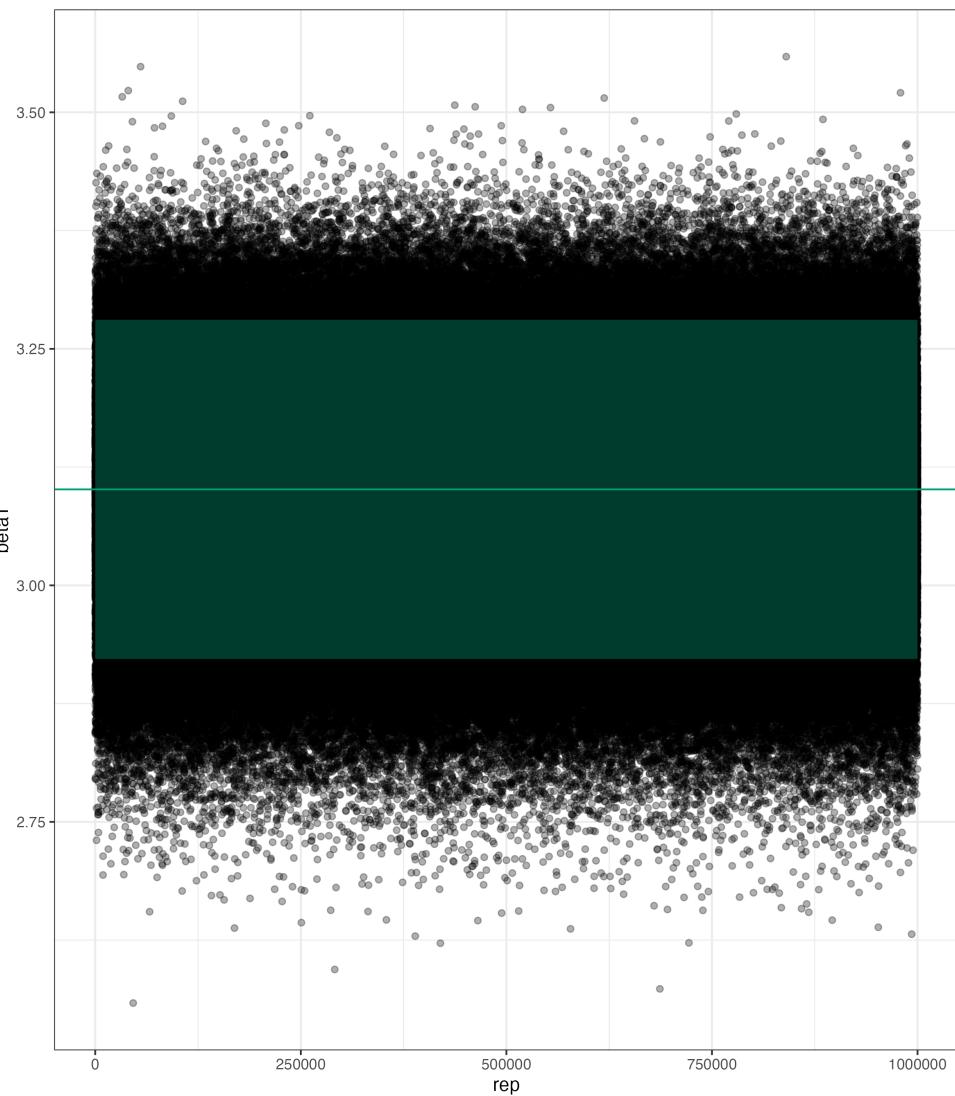
# Applied Bootstrapping

1. Write a function that extracts the coefficient we care about
2. Then we add resampling
3. Run it a bunch of times

```
1 Beta1Boot <- function(d){  
2   idx <- sample(nrow(d), replace=TRUE)  
3   m <- lm(y~x, d[idx,])  
4   beta1 <- unname(coef(m)[2])  
5   return(beta1)  
6 }  
7  
8 replicate(5, Beta1Boot(d))  
9  
10 # [1] 2.930935 3.264935 3.177743 3.027659 2.913476  
11  
12 N_boot <- 1e6  
13 boot <- data.frame(rep = 1:N_boot,  
14   beta1 = replicate(N_boot, Beta1Boot(d)))
```

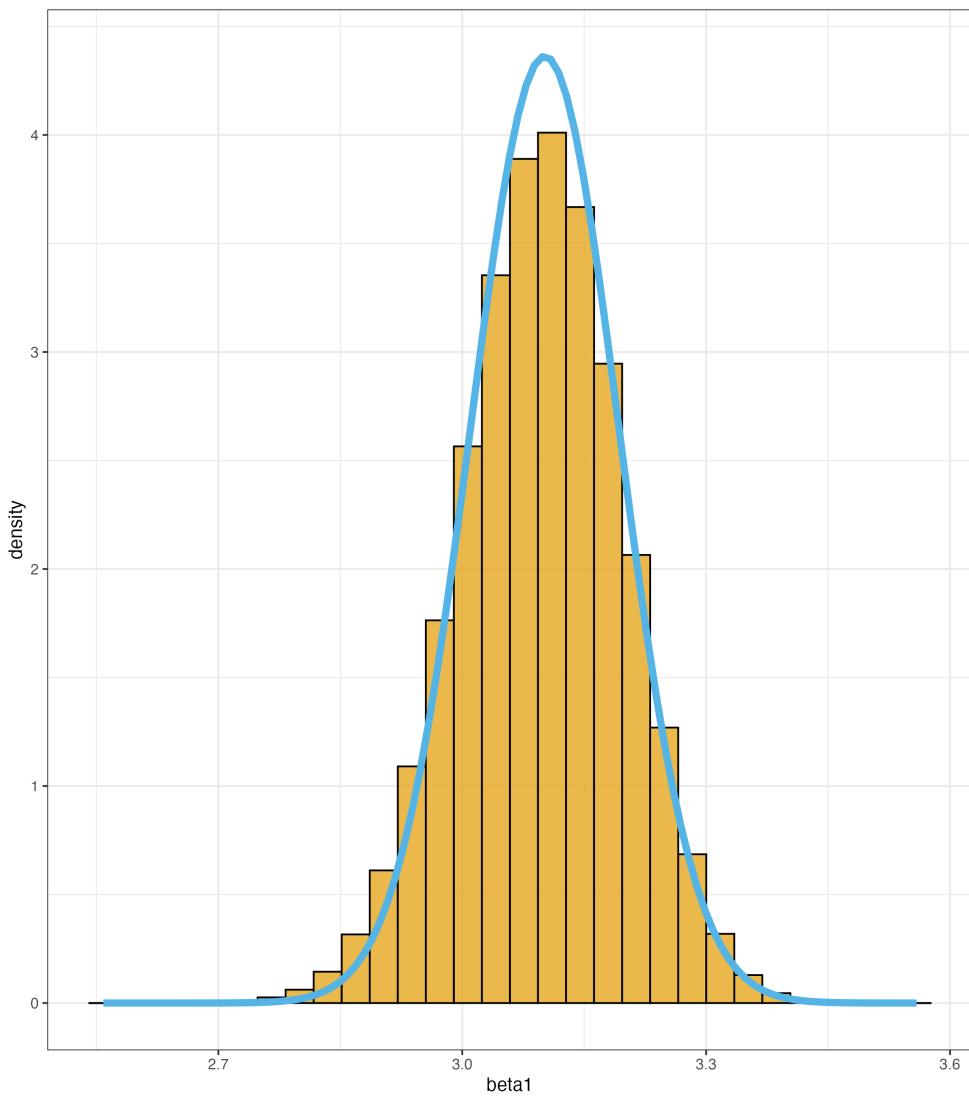
# Applied Bootstrapping

```
1 ggplot(boot, aes(x = rep, y = beta1)) +  
2   geom_point(alpha = 0.3) +  
3   geom_hline(  
4     aes(  
5       yintercept =  
6         summary(m)$coefficients[2, 1]),  
7       color = okabeito_colors(3)  
8     ) +  
9   geom_ribbon(  
10    aes(  
11      ymin =  
12        summary(m)$coefficients[2, 1] -  
13        1.96*summary(m)$coefficients[2, 2],  
14      ymax =  
15        summary(m)$coefficients[2, 1] +  
16        1.96*summary(m)$coefficients[2, 2]  
17    ),  
18    fill = okabeito_colors(3),  
19    alpha = 0.4  
20  ) +  
21  theme_bw()
```



# Applied Bootstrapping

```
1 ggplot(boot, aes(x = beta1)) +
2   geom_histogram(
3     aes(y = after_stat(density)),
4     bins = 30,
5     color = 'black',
6     fill = okabeito_colors(1),
7     alpha = 0.7
8   ) +
9   geom_function(
10     fun = dnorm,
11     size = 2,
12     color = okabeito_colors(2),
13     args = list(
14       mean = summary(m)$coefficients[2, 1],
15       sd = summary(m)$coefficients[2, 2]
16     )
17   ) +
18   theme_bw()
```



# Applied Bootstrapping

- `lm()` estimate, SE, and 95% CI:

```
1 summary(m)$coefficients[2,1:2]
2 c(summary(m)$coefficients[2,1] - 1.96*summary(m)$coefficients[2,2],
3   summary(m)$coefficients[2,1] + 1.96*summary(m)$coefficients[2,2])
```

- Bootstrapped estimate, SE, and 95% CI:

```
1 c(mean(boot$beta1), sd(boot$beta1))
2 quantile(boot$beta1, c(0.025, 0.975))
```

# Applied Bootstrapping

- `lm()` estimate, SE, and 95% CI:

```
1 summary(m)$coefficients[2,1:2]
2
3 #   Estimate Std. Error
4 # 3.10145067 0.09144205
5
6 c(summary(m)$coefficients[2,1] - 1.96*summary(m)$coefficients[2,2],
7   summary(m)$coefficients[2,1] + 1.96*summary(m)$coefficients[2,2])
```

- Bootstrapped estimate, SE, and 95% CI:

```
1 c(mean(boot$beta1), sd(boot$beta1))
2 quantile(boot$beta1, c(0.025, 0.975))
```

# Applied Bootstrapping

- `lm()` estimate, SE, and 95% CI:

```
1 summary(m)$coefficients[2,1:2]
2
3 #   Estimate Std. Error
4 # 3.10145067 0.09144205
5
6 c(summary(m)$coefficients[2,1] - 1.96*summary(m)$coefficients[2,2],
7   summary(m)$coefficients[2,1] + 1.96*summary(m)$coefficients[2,2])
8
9 # [1] 2.922224 3.280677
```

- Bootstrapped estimate, SE, and 95% CI:

```
1 c(mean(boot$beta1), sd(boot$beta1))
2 quantile(boot$beta1, c(0.025, 0.975))
```

# Applied Bootstrapping

- `lm()` estimate, SE, and 95% CI:

```
1 summary(m)$coefficients[2,1:2]
2
3 #   Estimate Std. Error
4 # 3.10145067 0.09144205
5
6 c(summary(m)$coefficients[2,1] - 1.96*summary(m)$coefficients[2,2],
7   summary(m)$coefficients[2,1] + 1.96*summary(m)$coefficients[2,2])
8
9 # [1] 2.922224 3.280677
```

- Bootstrapped estimate, SE, and 95% CI:

```
1 c(mean(boot$beta1), sd(boot$beta1))
2
3 # [1] 3.09708749 0.09906309
4
5 quantile(boot$beta1, c(0.025, 0.975))
```

# Applied Bootstrapping

- `lm()` estimate, SE, and 95% CI:

```
1 summary(m)$coefficients[2,1:2]
2
3 #   Estimate Std. Error
4 # 3.10145067 0.09144205
5
6 c(summary(m)$coefficients[2,1] - 1.96*summary(m)$coefficients[2,2],
7   summary(m)$coefficients[2,1] + 1.96*summary(m)$coefficients[2,2])
8
9 # [1] 2.922224 3.280677
```

- Bootstrapped estimate, SE, and 95% CI:

```
1 c(mean(boot$beta1), sd(boot$beta1))
2
3 # [1] 3.09708749 0.09906309
4
5 quantile(boot$beta1, c(0.025, 0.975))
6
7 #      2.5%    97.5%
8 # 2.897704 3.286801
```

# Wrap Up

# Recap

- Regularization puts a penalty on the likelihood proportional to the length of the parameter vector
  - This shrinks parameters toward zero
  - Ridge regression shrinks them slightly
  - LASSO regression tends to force parameters to be zero
- Resampling is a way to estimate things that are difficult to know analytically
  - The jackknife drops single observations to create new datasets of approximately the same size
  - The bootstrap resamples *with replacement* to create new datasets the same size as your old data
- Regularization and the bootstrap are probably the two most generalizable applied techniques you'll learn here!

# Final Thoughts

- [PollEv.com/klintkanopka](https://PollEv.com/klintkanopka)