COM SCI M148, Lecture 7:
Cross-Validation & Hypothesis Testing

Prof. Nathan Szymanski
nszymanski@seas.ucla.edu
Department of Materials Science

UCLA

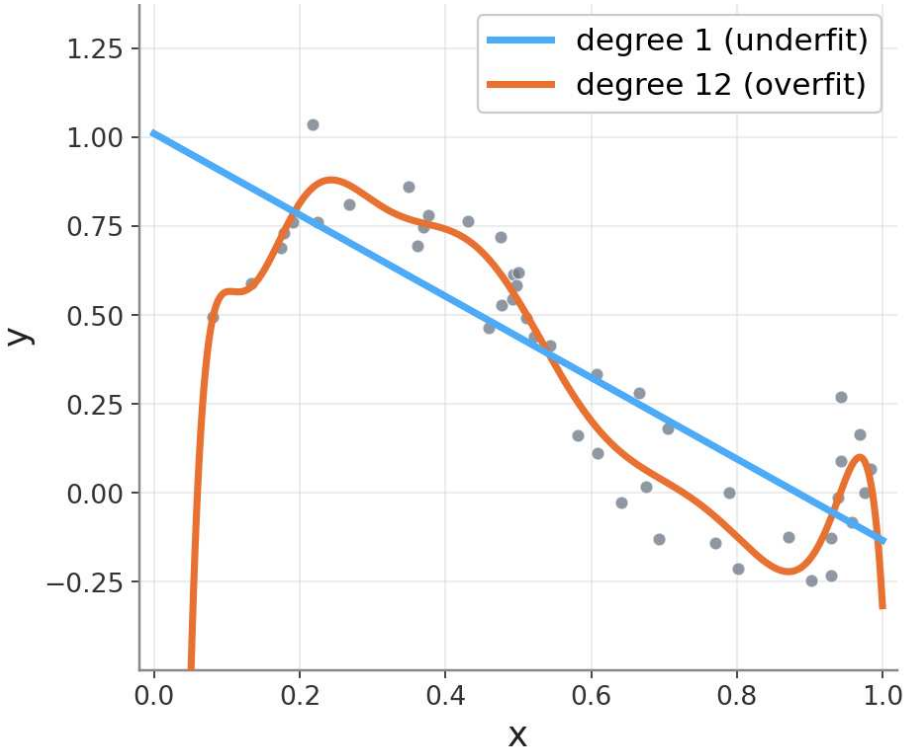
Samueli
School of Engineering

Recap: Models for regression

Linear regression:

$$\hat{y} = \beta_0 + \beta_1 x + \beta_2 x^2 + \cdots$$

Degree (x^n) controls model complexity



Recap: Models for regression

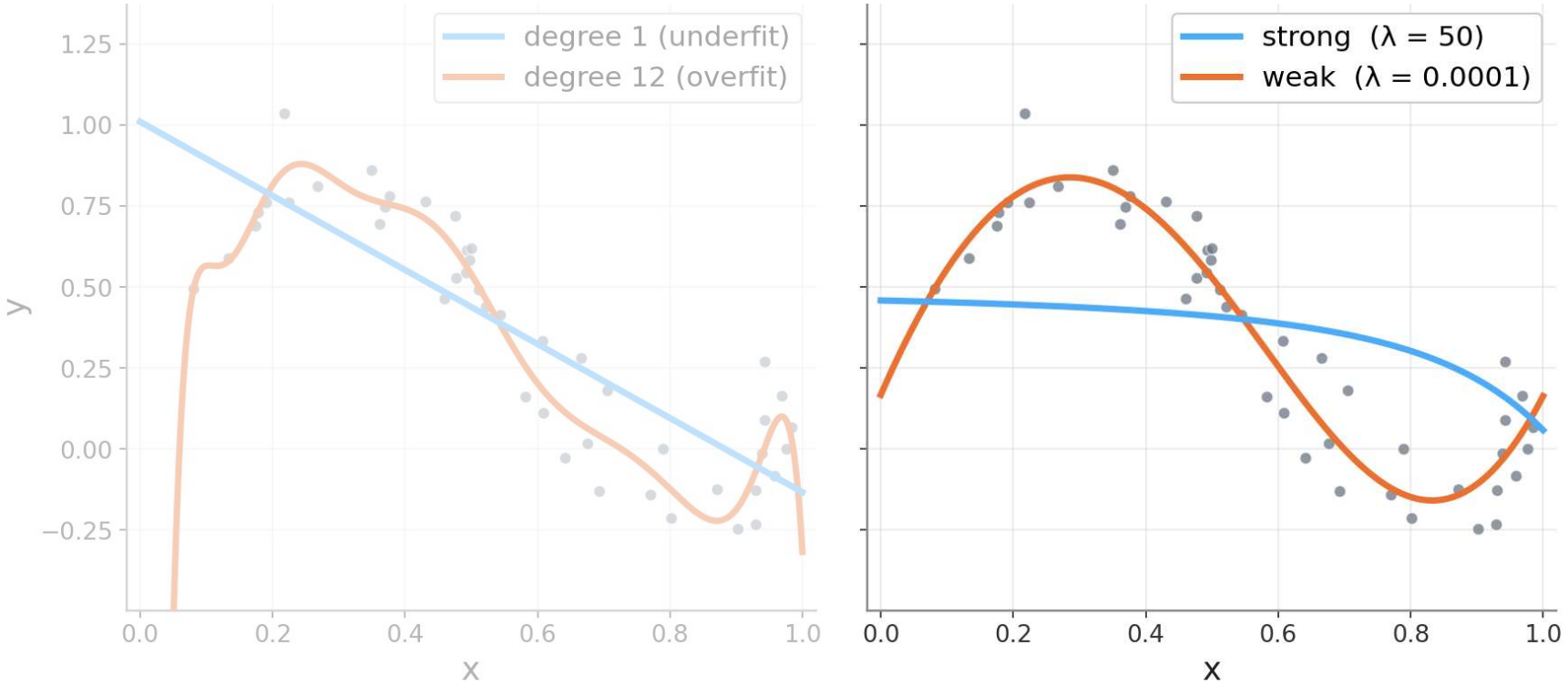
Linear regression:

$$\hat{y} = \beta_0 + \beta_1 x + \beta_2 x^2 + \cdots$$

Ridge and Lasso :

$$\mathcal{L} = \mathrm{MSE}(\beta_i) + \lambda R(\beta_i)$$

Regularization strength (λ) controls model complexity



Recap: Models for regression

Linear regression:

$$\hat{y} = \beta_0 + \beta_1 x + \beta_2 x^2 + \cdots$$

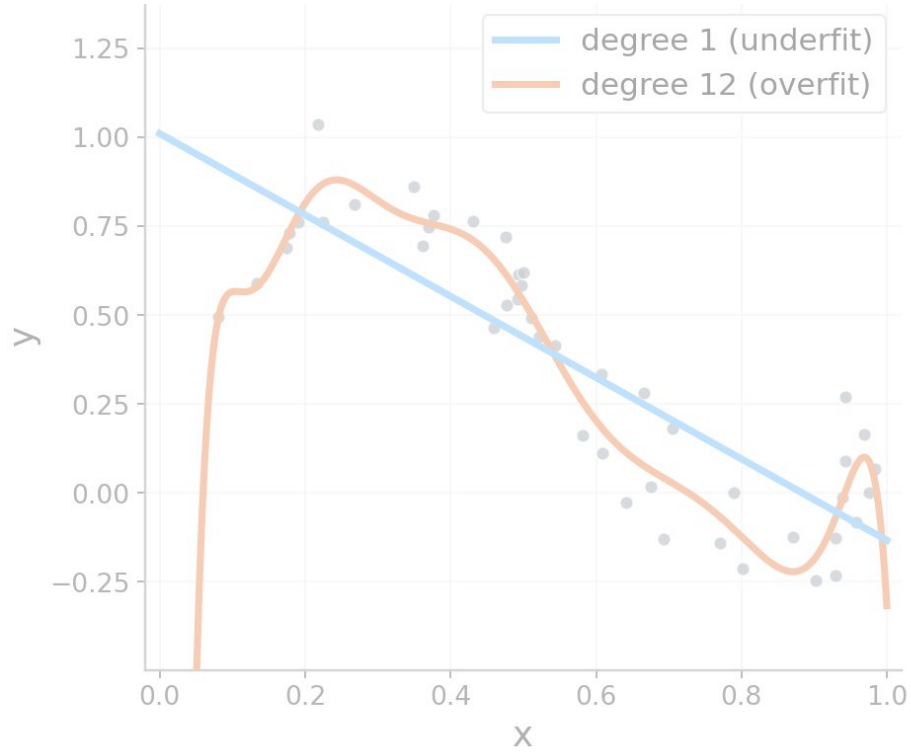
Ridge and Lasso :

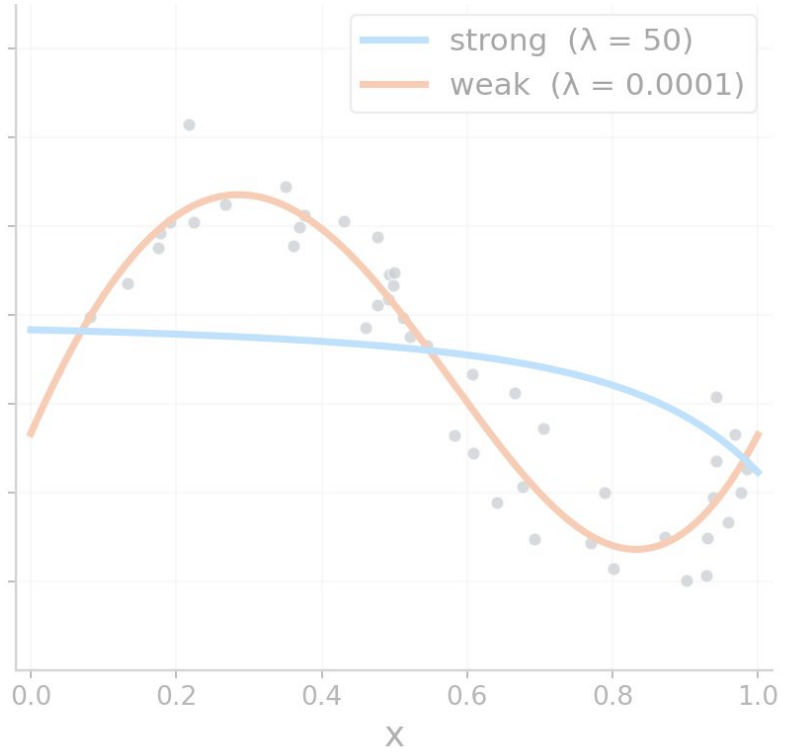
$$\mathcal{L} = \mathrm{MSE}(\beta_i) + \lambda R(\beta_i)$$

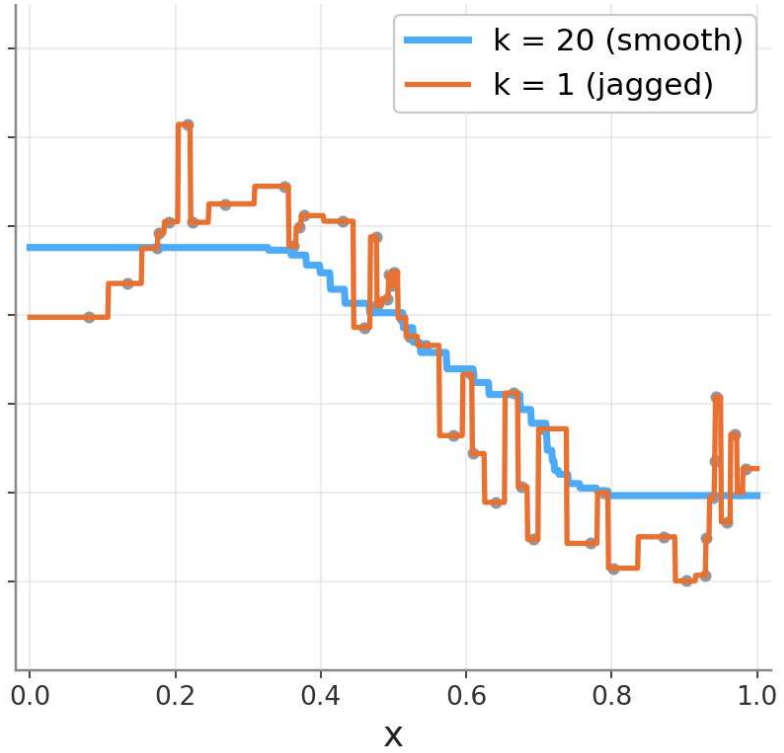
K-Nearest Neighbors:

$$\hat{y} = \frac{1}{k}\sum_i y_i \text{ for } i \in N_k(x')$$

Number of nearest neighbors (k) controls model complexity



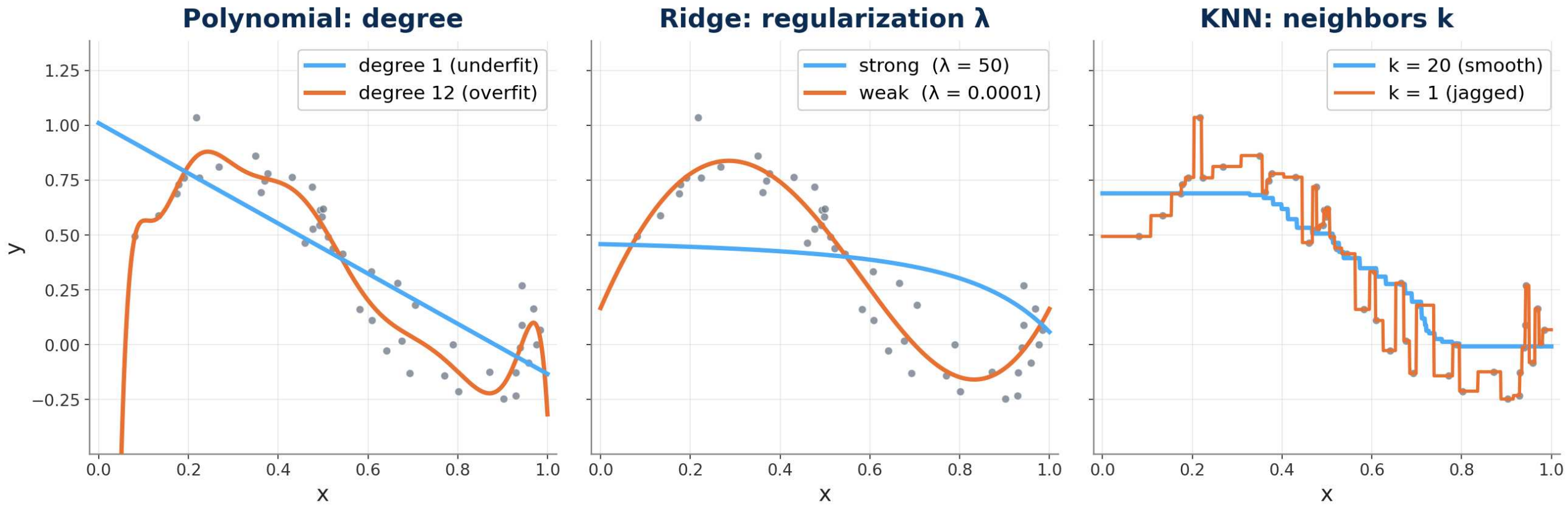




Recap: Models for regression

Two questions we'd like to answer in the 1st half of today's lecture:

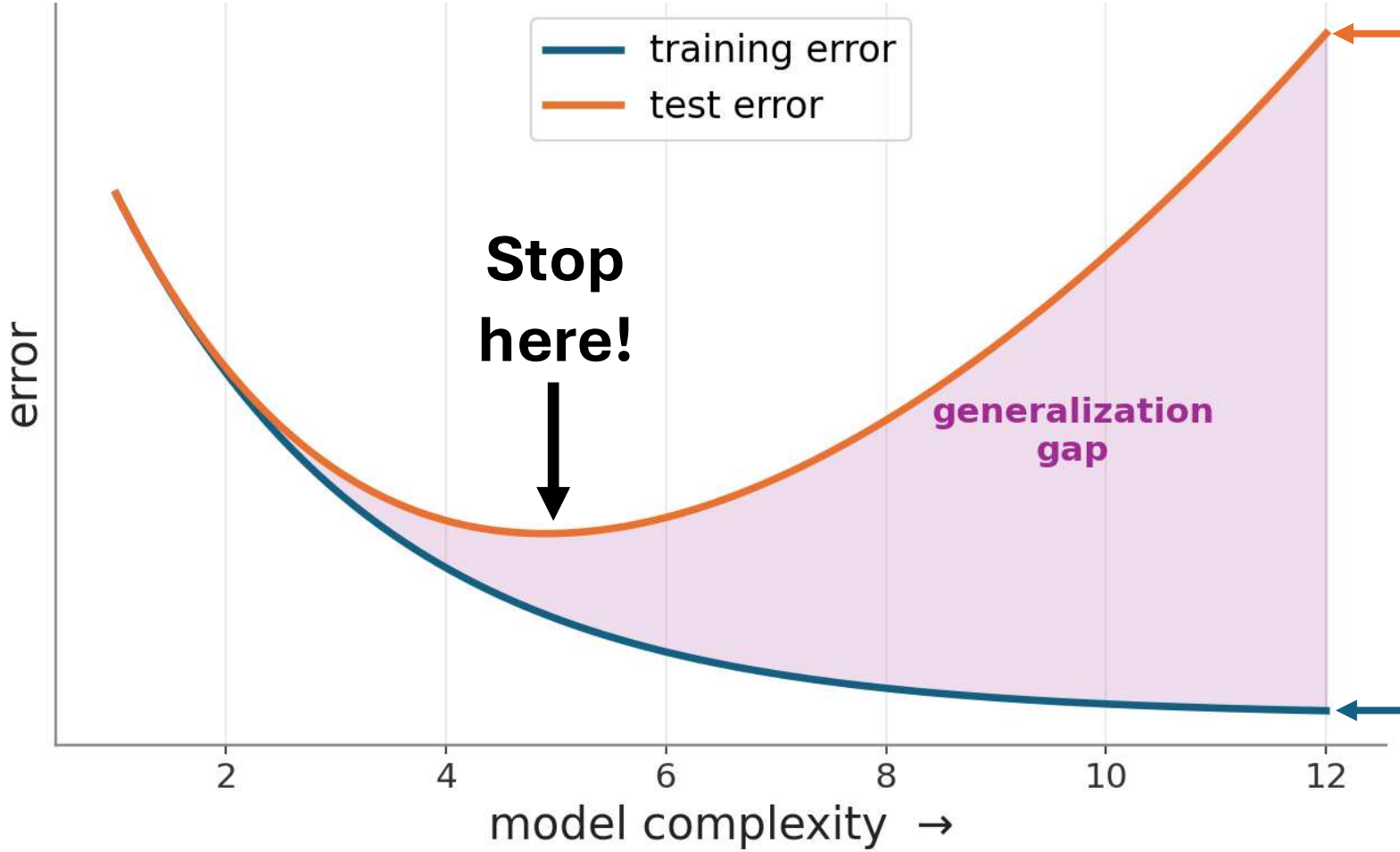
1) **Which model should we use on a given dataset?**
2) **How should we tune the complexity of that model?**



We’d like a model that can generalize well

We care most about performing well on **new data**, not the data we trained on

The generalization gap



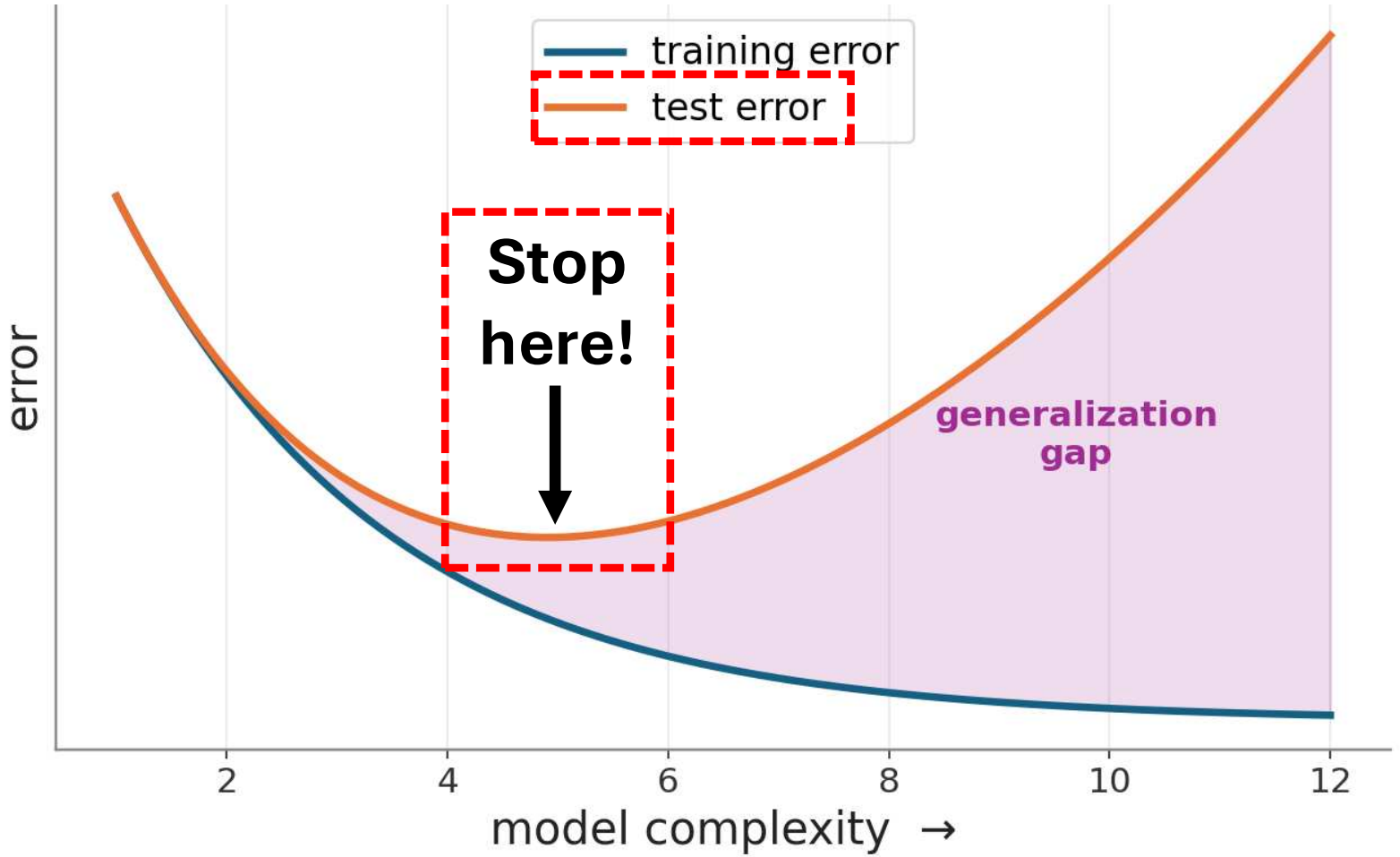
But it is the **test error** we want to be as low as possible

The difference between training and test error: **"generalization gap"**

Remember: we can always **reduce training error by increasing model complexity** (*e.g.*, $k = 1$ for KNNs)

The gap between training and test error grows as the model overfits

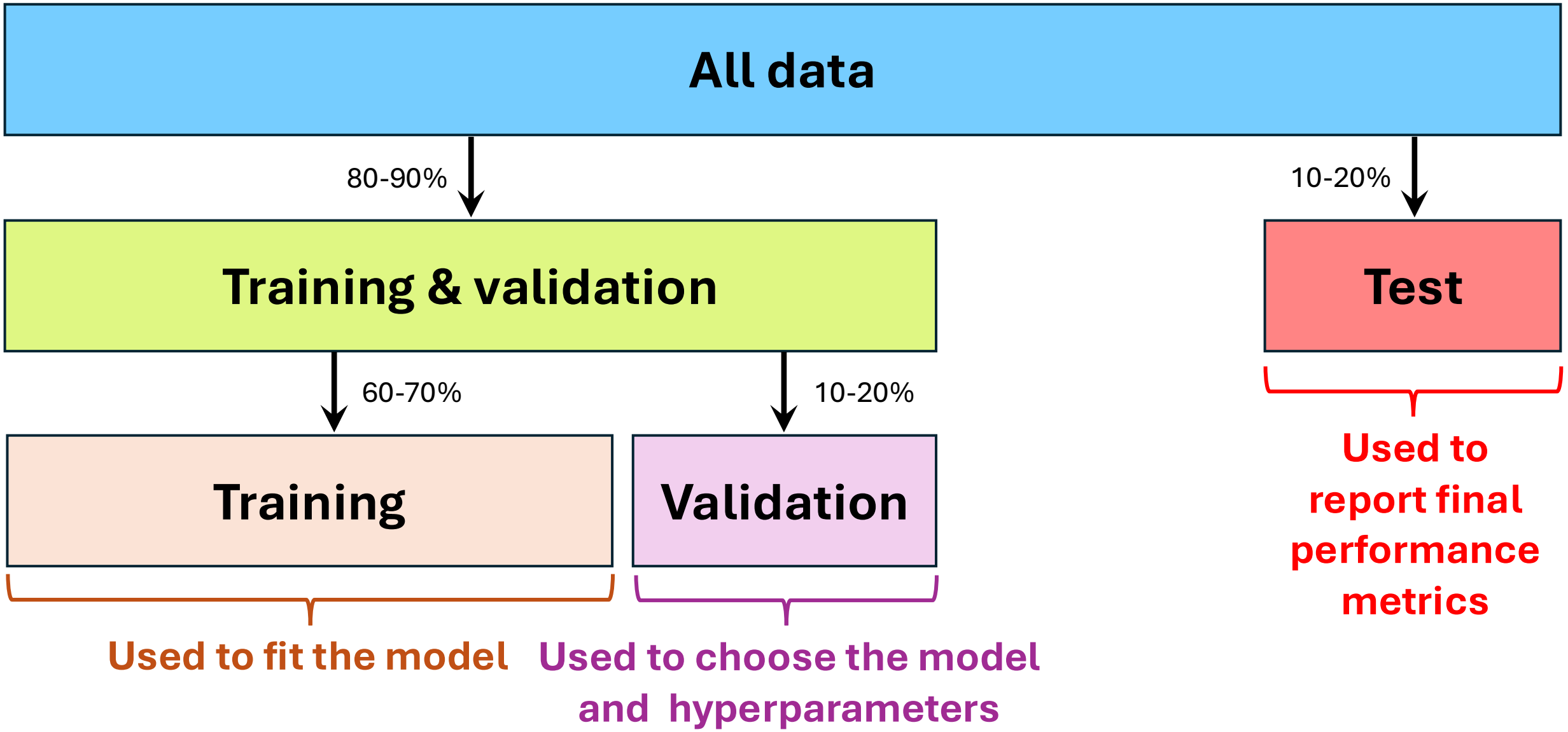
The generalization gap



But keep in mind we *cannot* use the test error directly for model and hyperparameter selection…

The gap between training and test error grows as the model overfits

Recall: Training, validation, and testing



Step 1: Train a variety of models

Train each candidate on TRAIN

Use enough data for training so that you can get a reasonable fit, but not so much that a statistically insignificant amount of data is left for validation & testing

TRAINING set

Fit model parameters (β, neighbors stored)

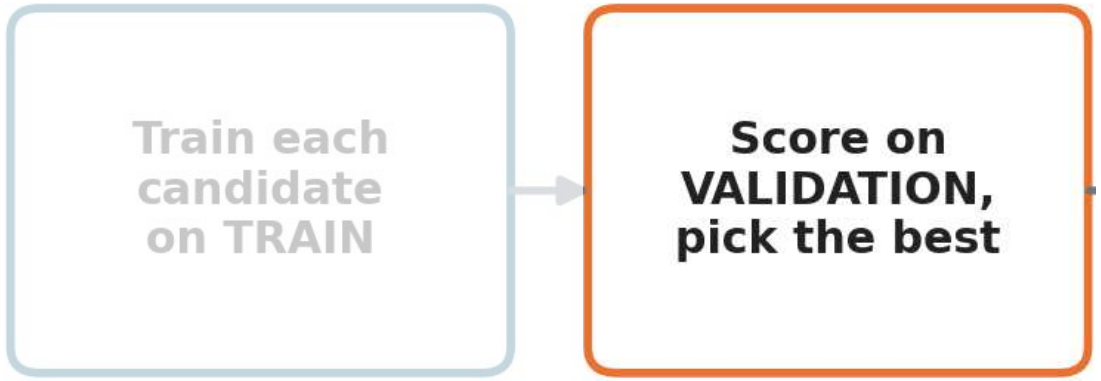
seen during fitting

Large datasets (100,000+ samples):
Can use **> 90% for training**

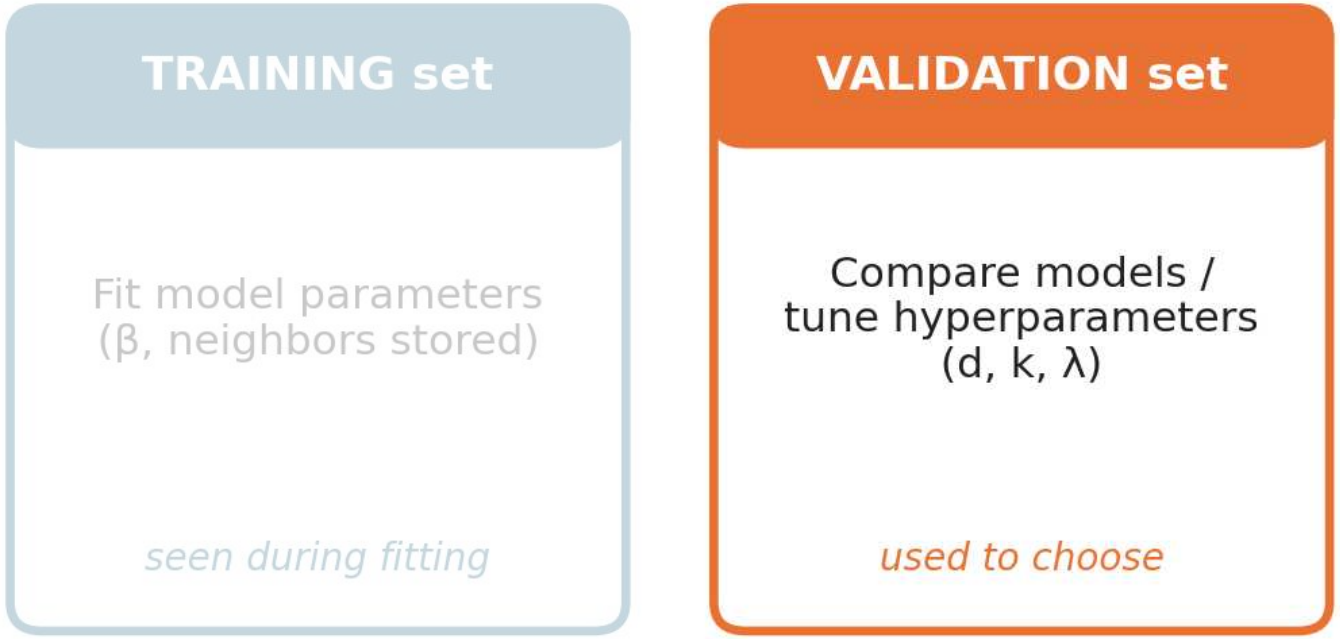
Medium datasets (few thousand samples):
Can use ≈ **80% for training**

Small datasets (< 1,000 samples):
Only use ≈ **60% for training**

Step 2: Validate those models

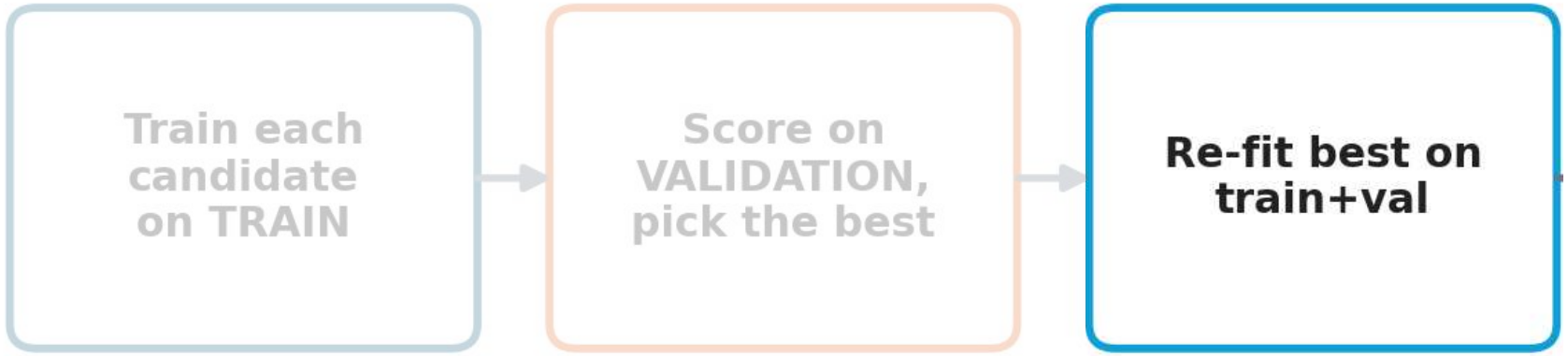


Aim to have at least a few hundred samples for validation (may not always be possible)

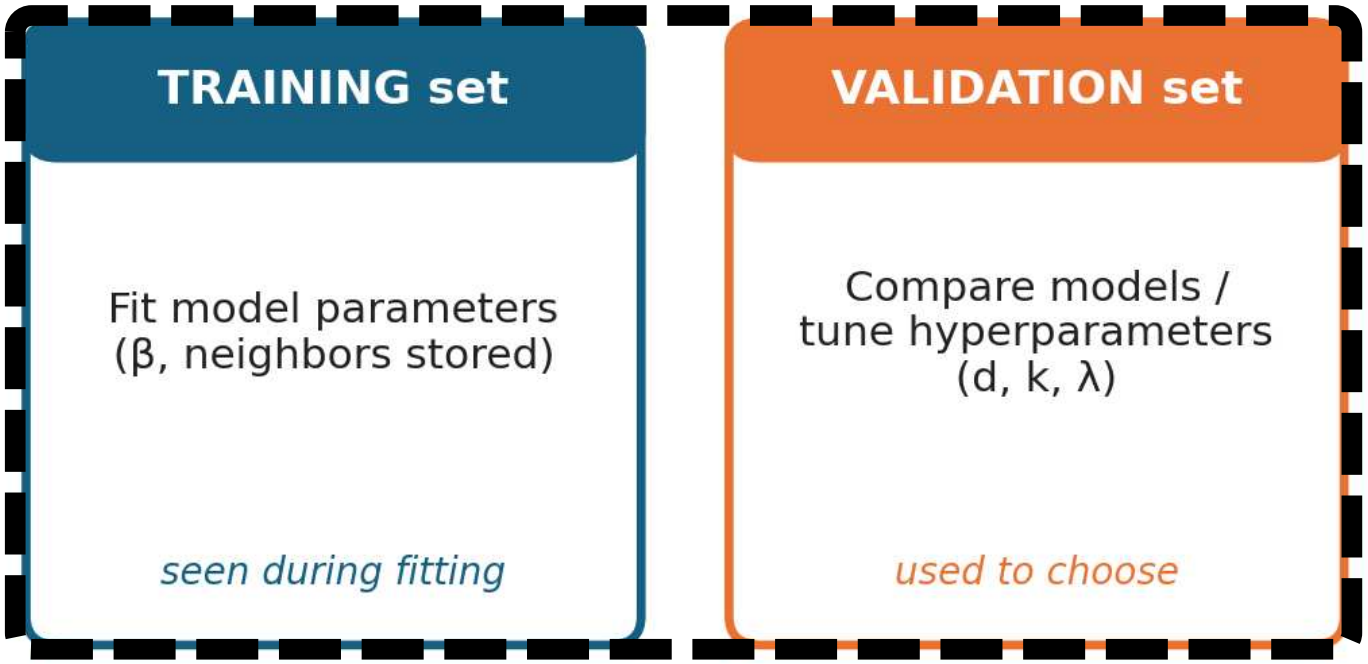


Try many different models and hyperparameters – **choose the one that gives minimal validation error**

Recall: Training, validation, and testing

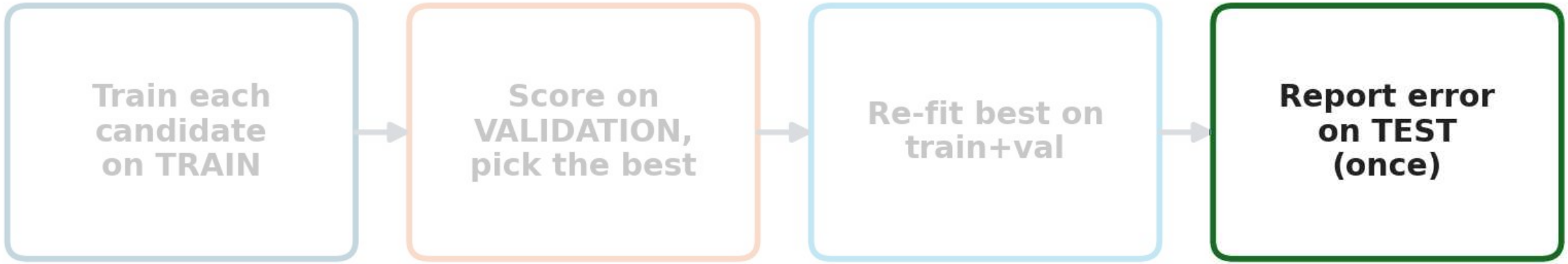


We might as well use all the data we can for the final fitting of the best model!



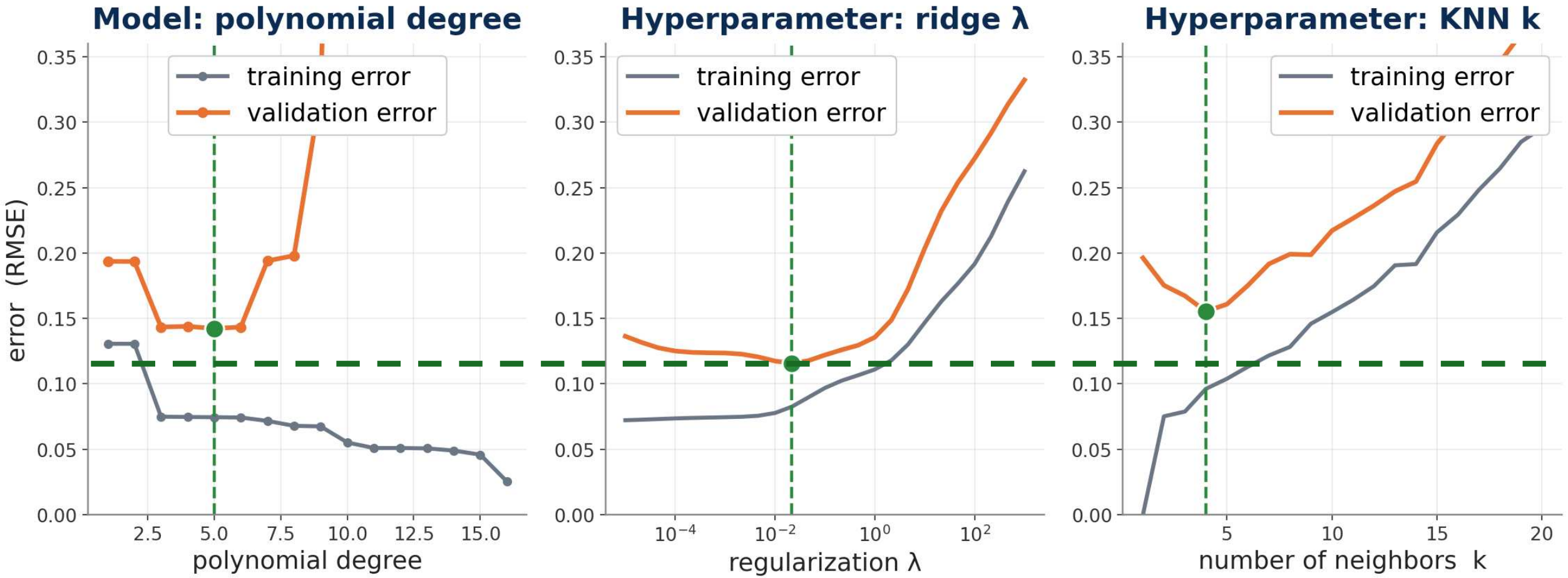
Use both datasets to optimize how well your model generalizes

Recall: Training, validation, and testing



TRAINING set	VALIDATION set	TEST set
Fit model parameters (β, neighbors stored)	Compare models / tune hyperparameters (d, k, λ)	Estimate true generalization error ONCE, at the end
seen during fitting	*used to choose*	*locked away*

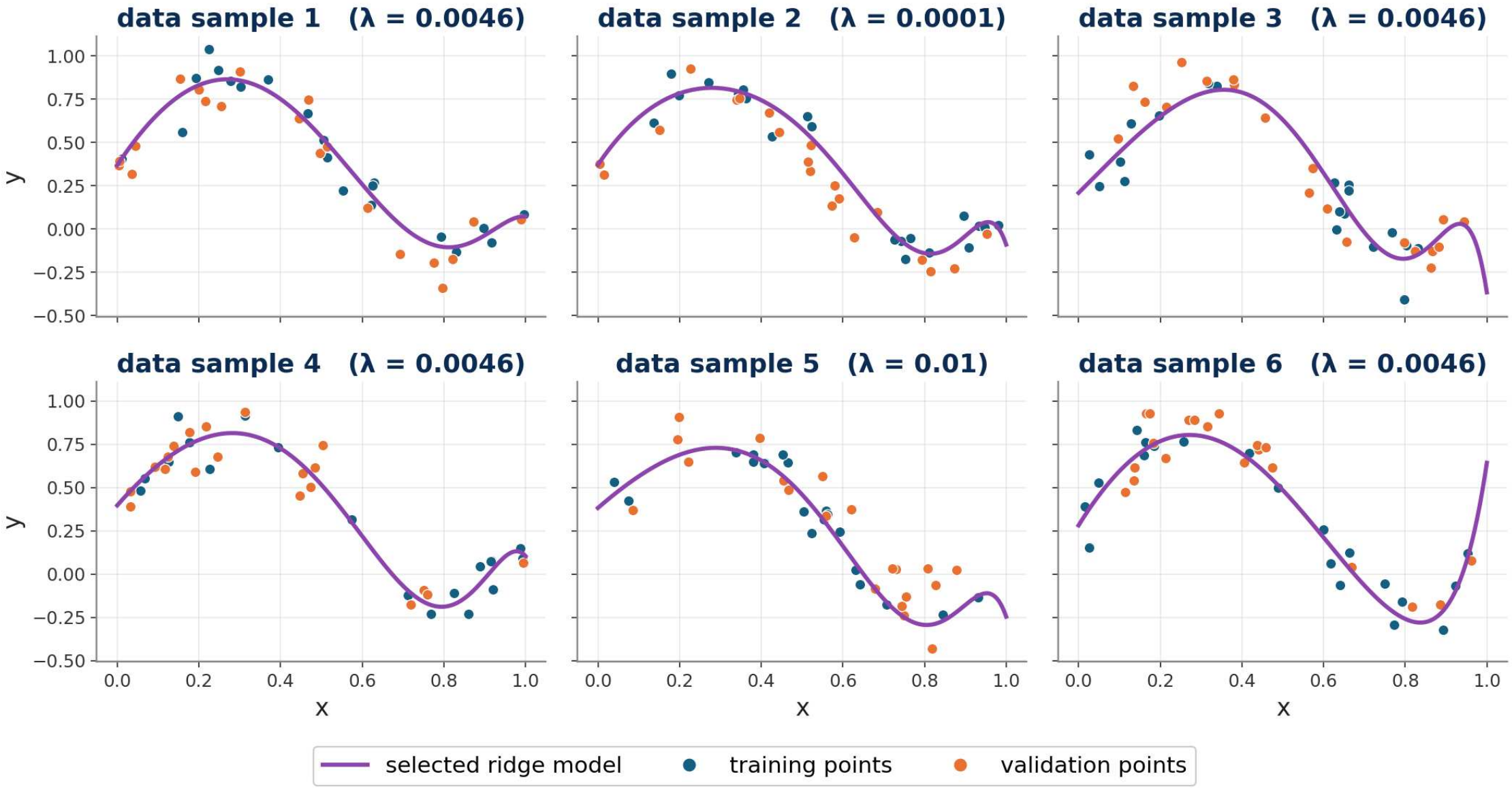
Model *and* hyperparameter selection





Ridge regression with $\lambda \sim 0.01$ is the winner!

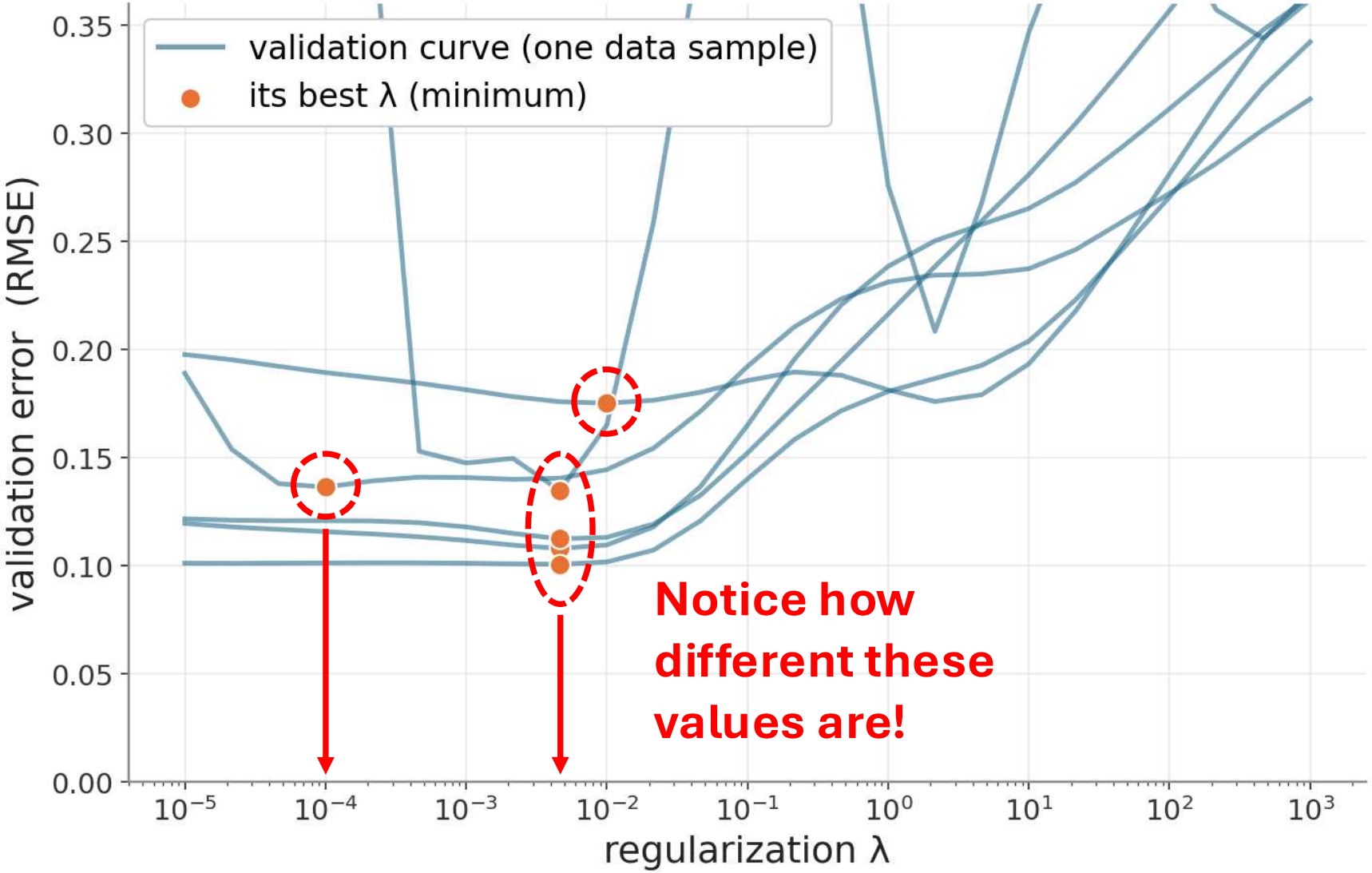
The problem with using just *one* validation set



In the small data regime, the best model & hyperparam may vary with the choice of training vs. validation data

The problem with using just *one* validation set

Each validation curve is from one random split of the same dataset



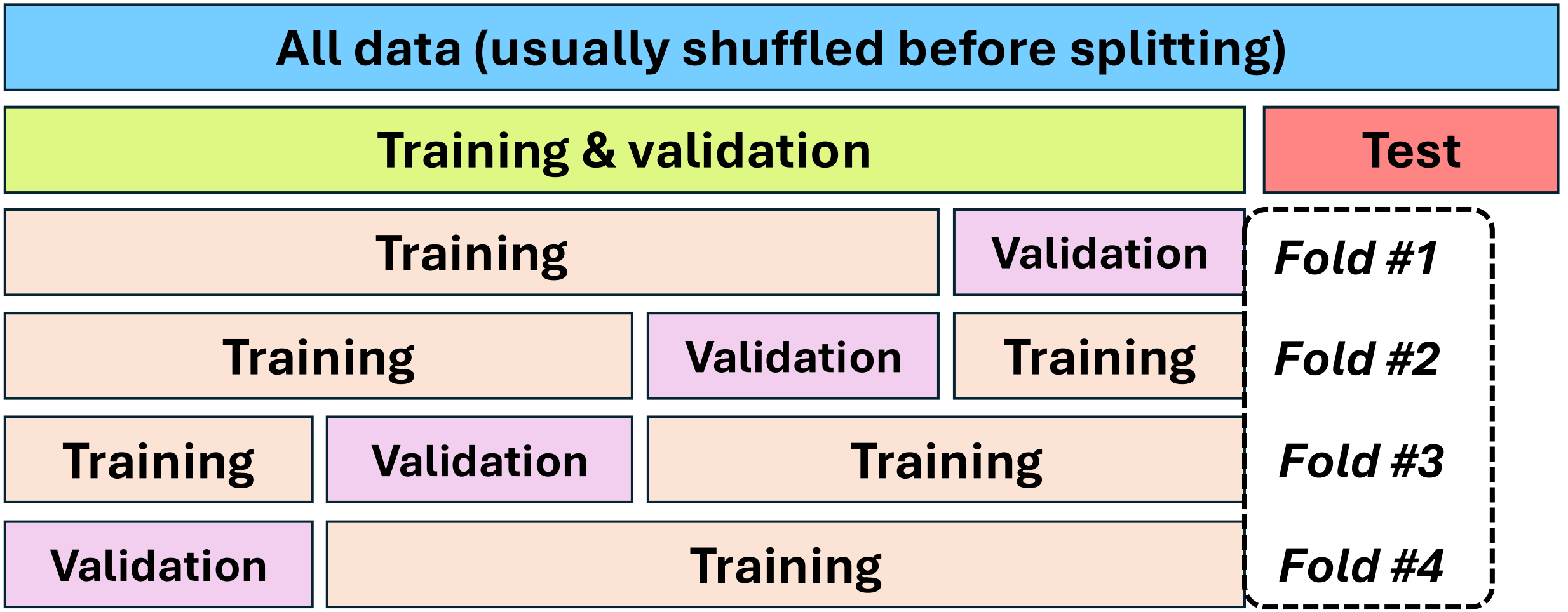
When the dataset is small (< 100 samples), we see **large changes to the validation curve**, which leads to changes in the "optimal" model and hyperparameters, **depending on our training/validation split**

What if we averaged over these?

Cross-Validation (CV)

Use **many different training/validation splits** and average over them

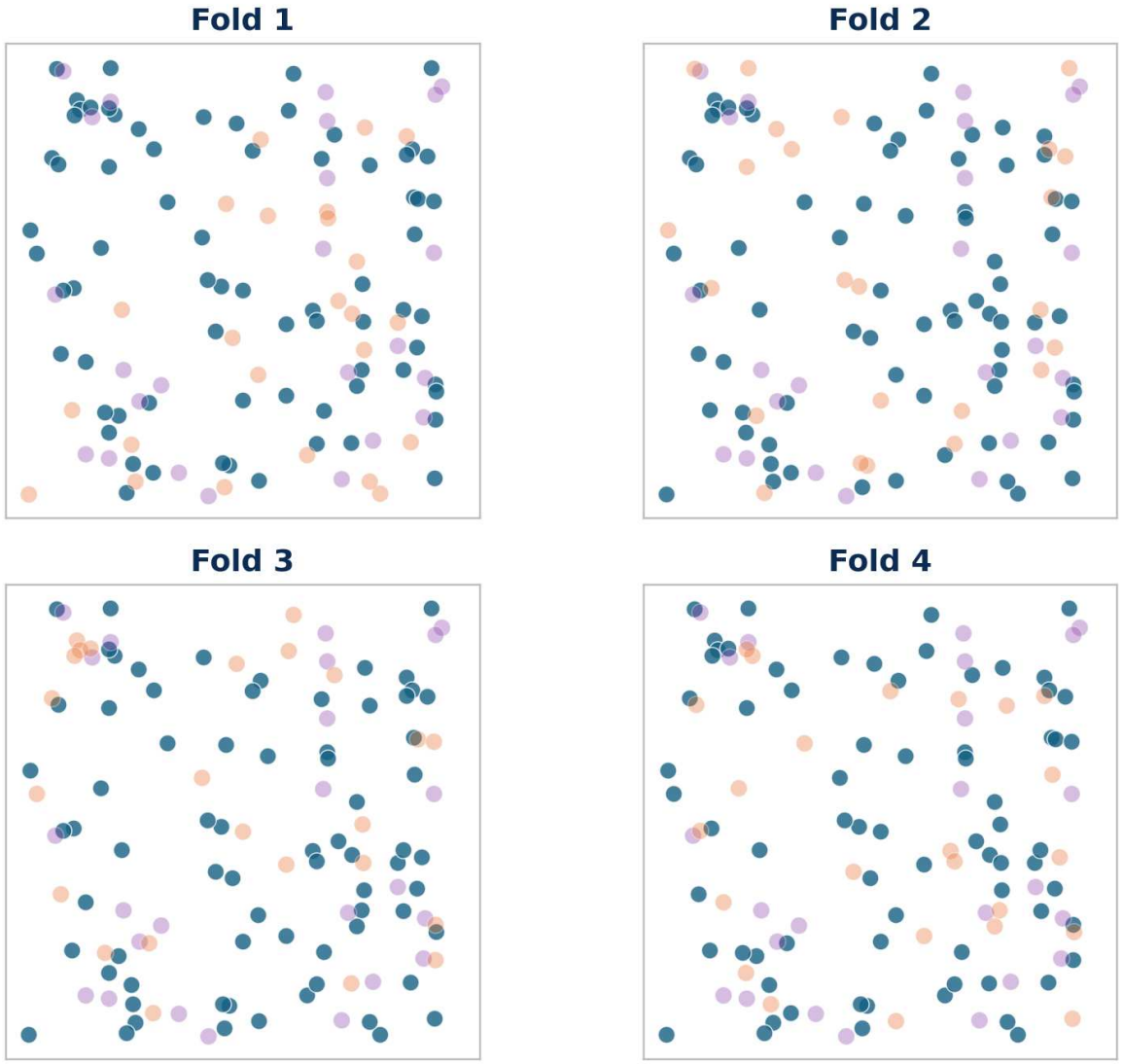
k-fold Cross-validation

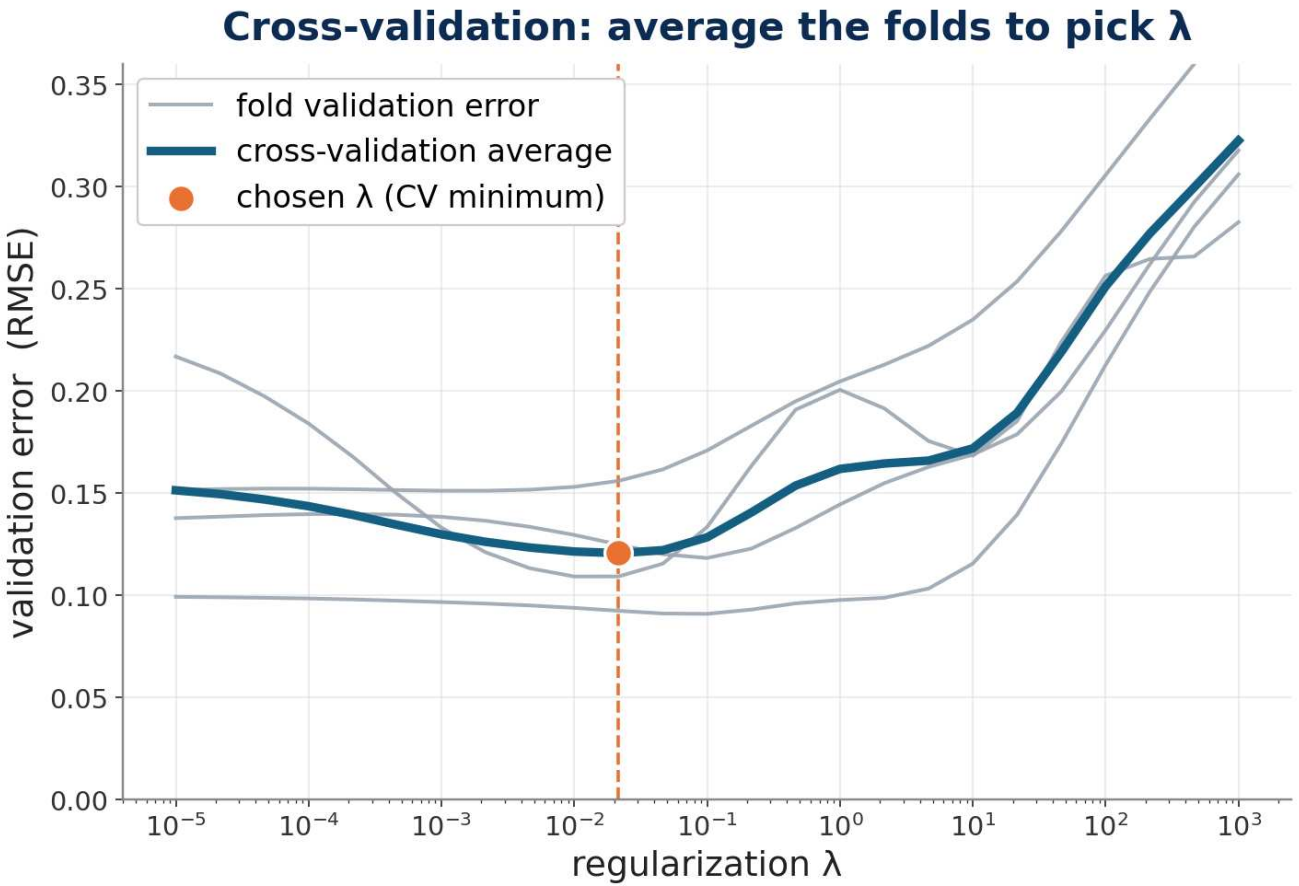


$$\text{CV error} = \text{mean}(\text{error } \#1, \#2, \#3, \#4)$$

4-fold CV

k-fold Cross-validation





Averaged CV curve is much smoother than any individual curve

Test set does not change!

The *k*-fold algorithm

$$\{(x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4), (x_5, y_5), (x_6, y_6), (x_7, y_7), (x_8, y_8), (x_9, y_9), (x_{10}, y_{10})\}$$

An example dataset containing 10 samples ($x = \text{input feature}, y = \text{output}$)

Assume a test set has already been set aside – let's focus on training/validation

The *k*-fold algorithm

$$\{(x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4), (x_5, y_5), (x_6, y_6), (x_7, y_7), (x_8, y_8), (x_9, y_9), (x_{10}, y_{10})\}$$

1) Randomly shuffle your data

$$\{(x_3, y_3), (x_9, y_9), (x_4, y_4), (x_{10}, y_{10}), (x_1, y_1), (x_8, y_8), (x_2, y_2), (x_6, y_6), (x_3, y_3), (x_7, y_7)\}$$

We do this to **avoid bias** in the data
Sometimes data is ordered in a specific way based on how it was collected

Make sure each output stays with its input

The *k*-fold algorithm

$\{(x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4), (x_5, y_5), (x_6, y_6), (x_7, y_7), (x_8, y_8), (x_9, y_9), (x_{10}, y_{10})\}$

1) **Randomly shuffle your data**

$\{(x_3, y_3), (x_9, y_9), (x_4, y_4), (x_{10}, y_{10}), (x_1, y_1), (x_8, y_8), (x_2, y_2), (x_6, y_6), (x_3, y_3), (x_7, y_7)\}$

2) **Split into k folds approximately equal in size ($k = 5$ is a standard choice)**

$\{(x_3, y_3), (x_9, y_9), (x_4, y_4), (x_{10}, y_{10}), (x_1, y_1), (x_8, y_8), (x_2, y_2), (x_6, y_6), (x_3, y_3), (x_7, y_7)\}$

These need not be *exactly* equal in size

For example, if we had 11 samples, then one of these folds would have three instead of two

The *k*-fold algorithm

$\{(x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4), (x_5, y_5), (x_6, y_6), (x_7, y_7), (x_8, y_8), (x_9, y_9), (x_{10}, y_{10})\}$

1) **Randomly shuffle your data**

$\{(x_3, y_3), (x_9, y_9), (x_4, y_4), (x_{10}, y_{10}), (x_1, y_1), (x_8, y_8), (x_2, y_2), (x_6, y_6), (x_3, y_3), (x_7, y_7)\}$

2) **Split into k folds approximately equal in size ($k = 5$ is a standard choice)**

$\{(x_3, y_3), (x_9, y_9), (x_4, y_4), (x_{10}, y_{10}), (x_1, y_1), (x_8, y_8), (x_2, y_2), (x_6, y_6), (x_3, y_3), (x_7, y_7)\}$

3) **Each fold is used *once* for validation while the others are used for training**

$\{(x_3, y_3), (x_9, y_9), (x_4, y_4), (x_{10}, y_{10}), (x_1, y_1), (x_8, y_8), (x_2, y_2), (x_6, y_6), (x_3, y_3), (x_7, y_7)\}$

$\{(x_3, y_3), (x_9, y_9), (x_4, y_4), (x_{10}, y_{10}), (x_1, y_1), (x_8, y_8), (x_2, y_2), (x_6, y_6), (x_3, y_3), (x_7, y_7)\}$

$\{(x_3, y_3), (x_9, y_9), (x_4, y_4), (x_{10}, y_{10}), (x_1, y_1), (x_8, y_8), (x_2, y_2), (x_6, y_6), (x_3, y_3), (x_7, y_7)\}$

...

The *k*-fold algorithm

$$\{(x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4), (x_5, y_5), (x_6, y_6), (x_7, y_7), (x_8, y_8), (x_9, y_9), (x_{10}, y_{10})\}$$

1) **Randomly shuffle your data**

$$\{(x_3, y_3), (x_9, y_9), (x_4, y_4), (x_{10}, y_{10}), (x_1, y_1), (x_8, y_8), (x_2, y_2), (x_6, y_6), (x_3, y_3), (x_7, y_7)\}$$

2) **Split into k folds approximately equal in size ($k = 5$ is a standard choice)**

$$\{(x_3, y_3), (x_9, y_9), (x_4, y_4), (x_{10}, y_{10}), (x_1, y_1), (x_8, y_8), (x_2, y_2), (x_6, y_6), (x_3, y_3), (x_7, y_7)\}$$

3) **Each fold is used *once* for validation while the others are used for training**

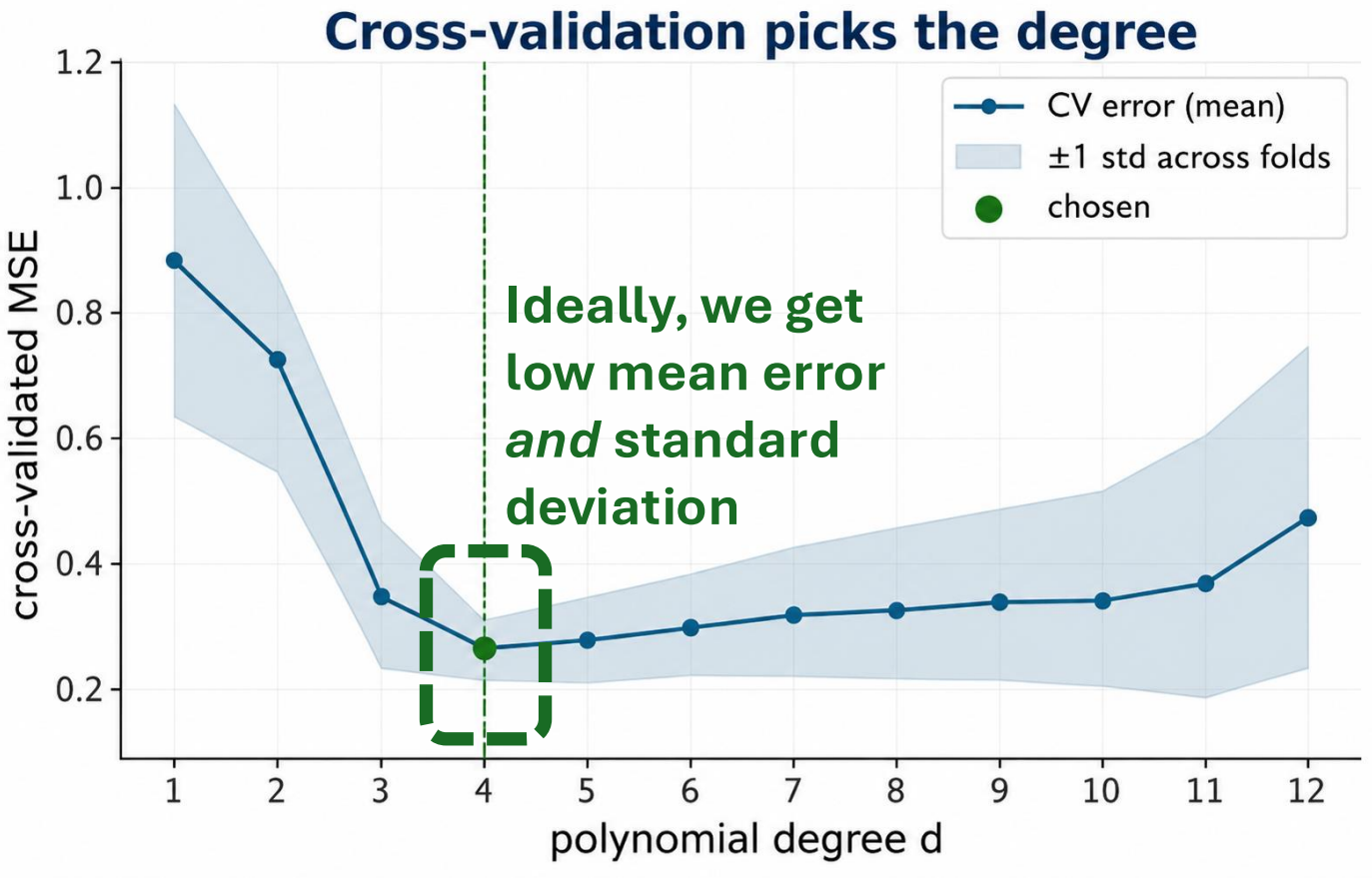
$$\{(x_3, y_3), (x_9, y_9), (x_4, y_4), (x_{10}, y_{10}), (x_1, y_1), (x_8, y_8), (x_2, y_2), (x_6, y_6), (x_3, y_3), (x_7, y_7)\}$$

4) **Compute the average cross-validation (CV) error**

$$CV_k = \frac{1}{k} \sum_{i=0}^{k} \mathrm{MSE}_{k-\mathrm{fold}}$$

And report the standard deviation in MSE across the folds

Choosing your model based on CV error



$$\hat{y} = \beta_0 + \beta_1 x + \beta_2 x_2 + \cdots + \beta_d x^d$$

$$CV_k = \frac{1}{k}\sum_{i=1}^{k} \text{MSE}_{k-\text{fold}}$$

$$SD_k = \sqrt{\frac{1}{K-1}\sum_{i=1}^{k} (\text{MSE}_k - CV_k)^2}$$

Error of each fold

Mean error of all folds

A **large standard deviation (SD_k) may suggest the model is unnecessary complex.**

Though sometimes it is unavoidable if you have relatively little data…

Pros and cons of cross-validation

1. Use *all* data (minus test) for *both* training and validation
2. Get a more reliable estimate of generalization error
3. Model tuning is much less sensitive to one “unlucky” split
4. Standard deviation provides insight: Is model too complex?



1. Computationally more **expensive** since we have to train each model k times instead of just one time. Not a big deal for our simple models, but can become a problem for deep NNs…

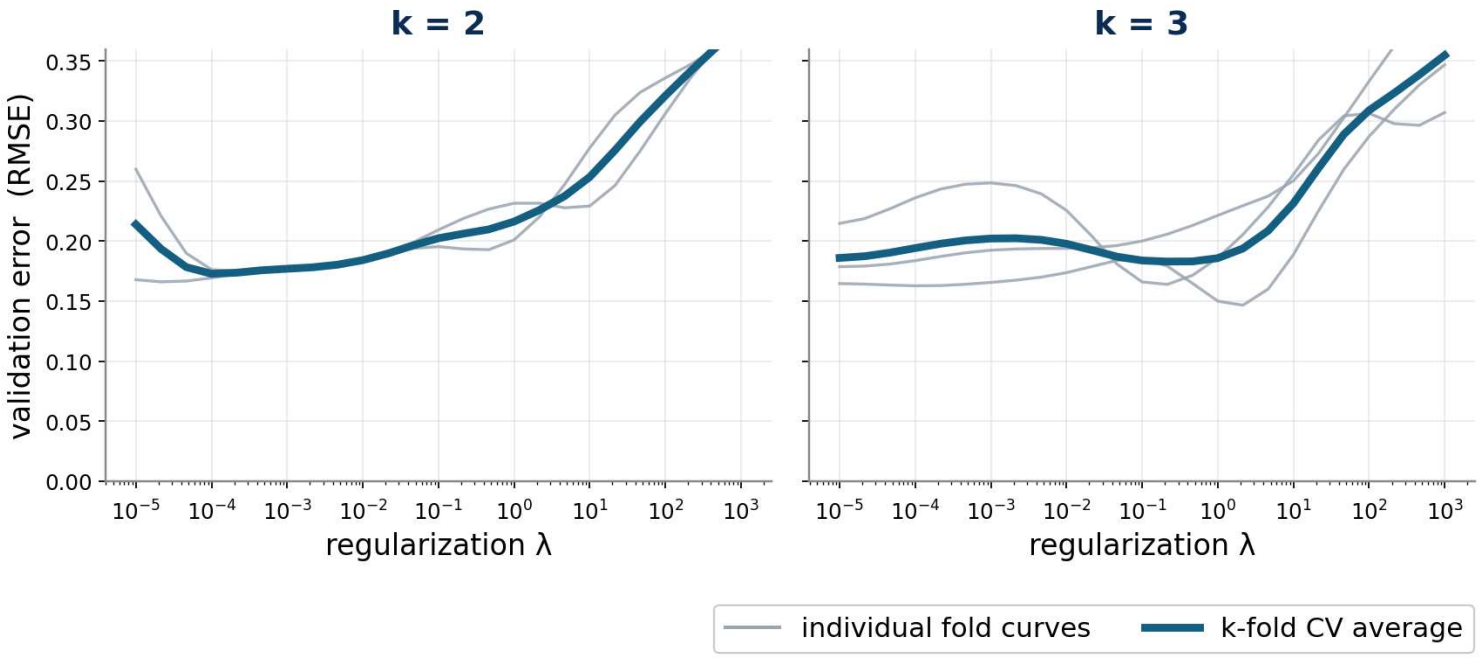


Questions?

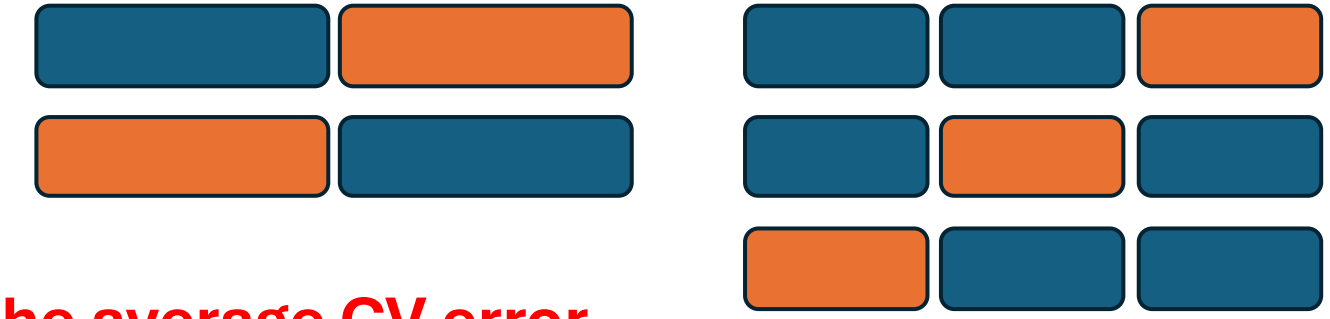
Some more details on cross-validation

How many folds to use? Leave-one-out CV and data leakage

The effects of increasing k folds

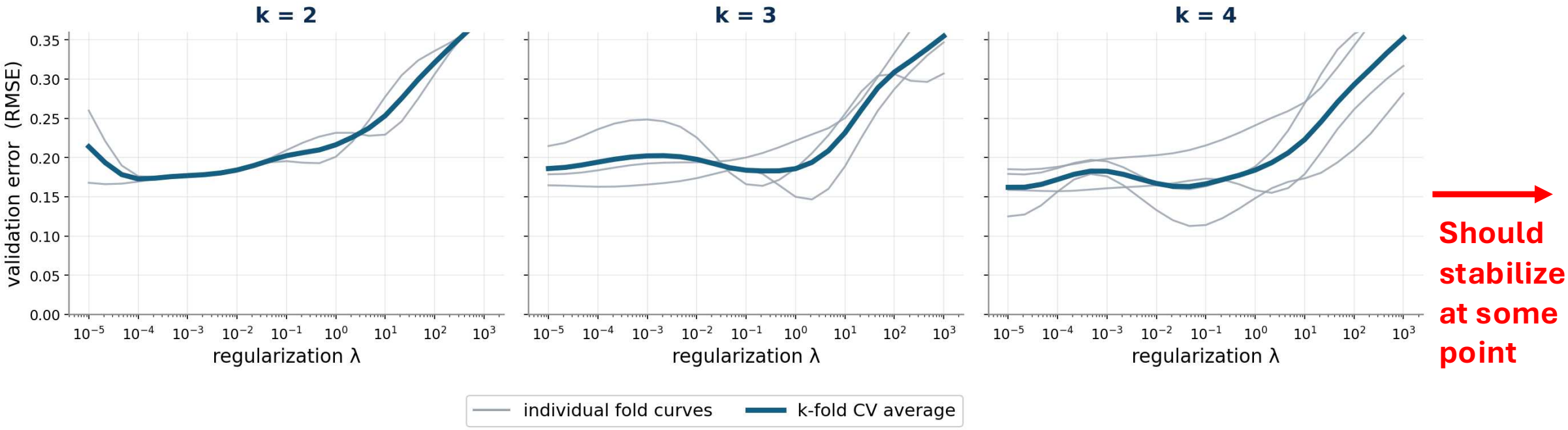


Notice these two average curves are not exactly the same!

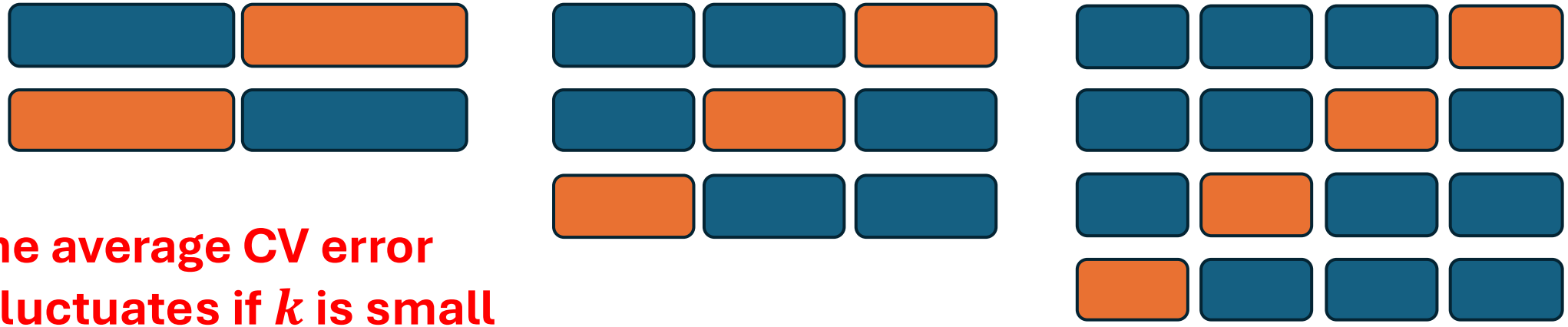


Even the average CV error curve fluctuates if k is small

The effects of increasing k folds

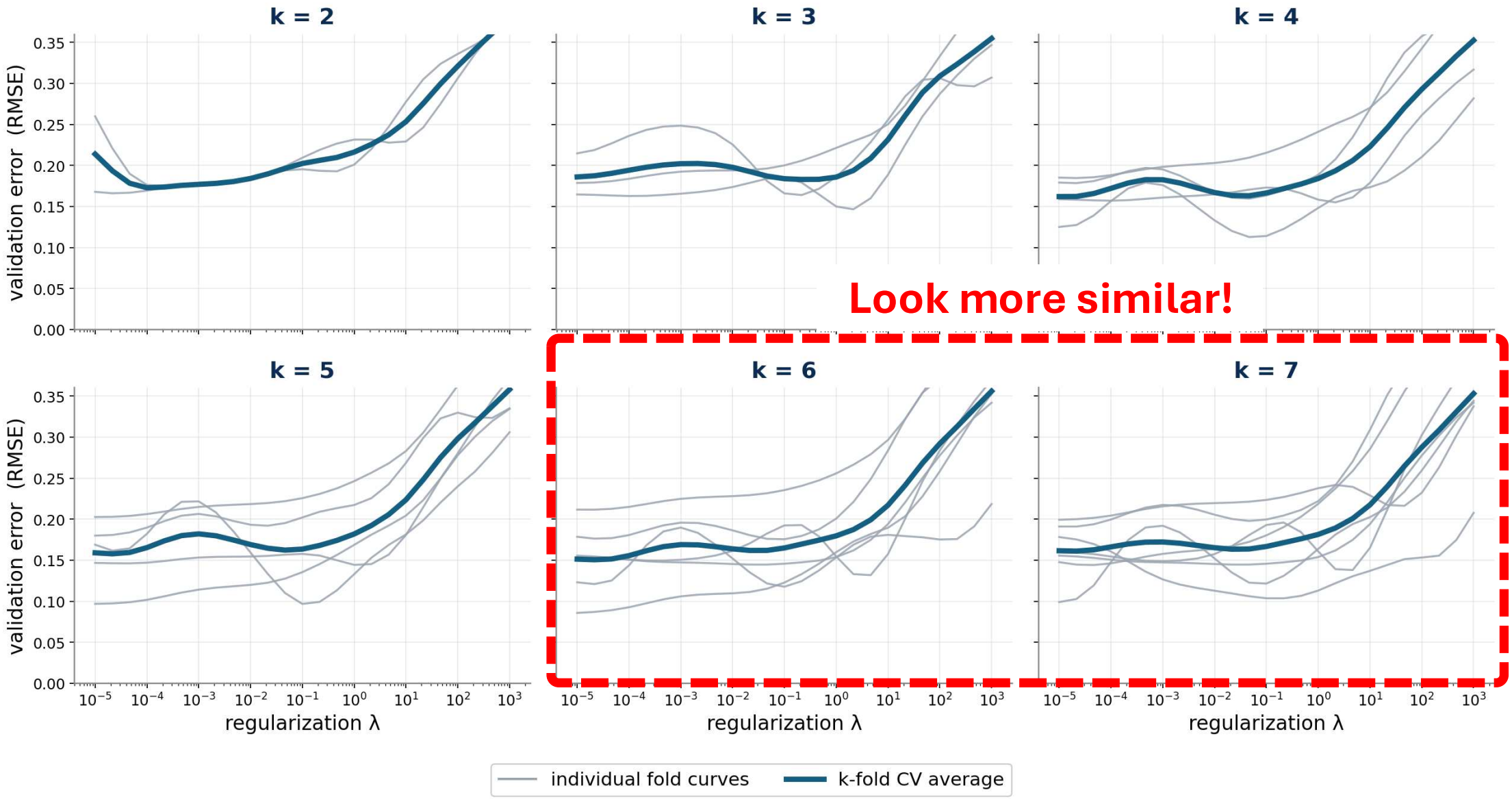


Should stabilize at some point

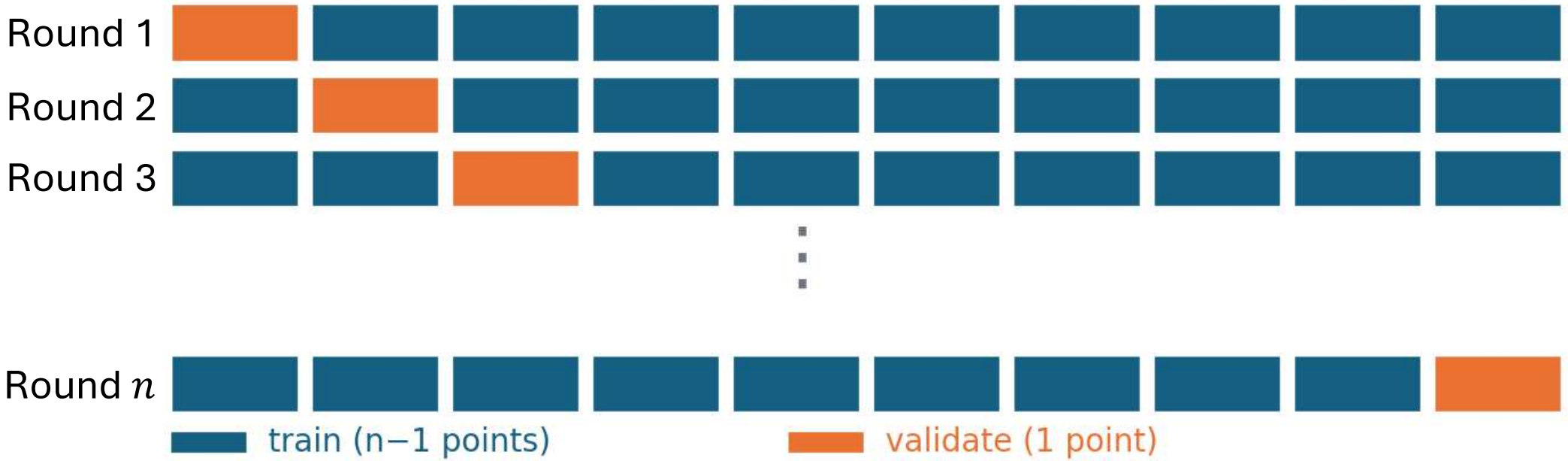


Even the average CV error curve fluctuates if k is small

The effects of increasing k folds

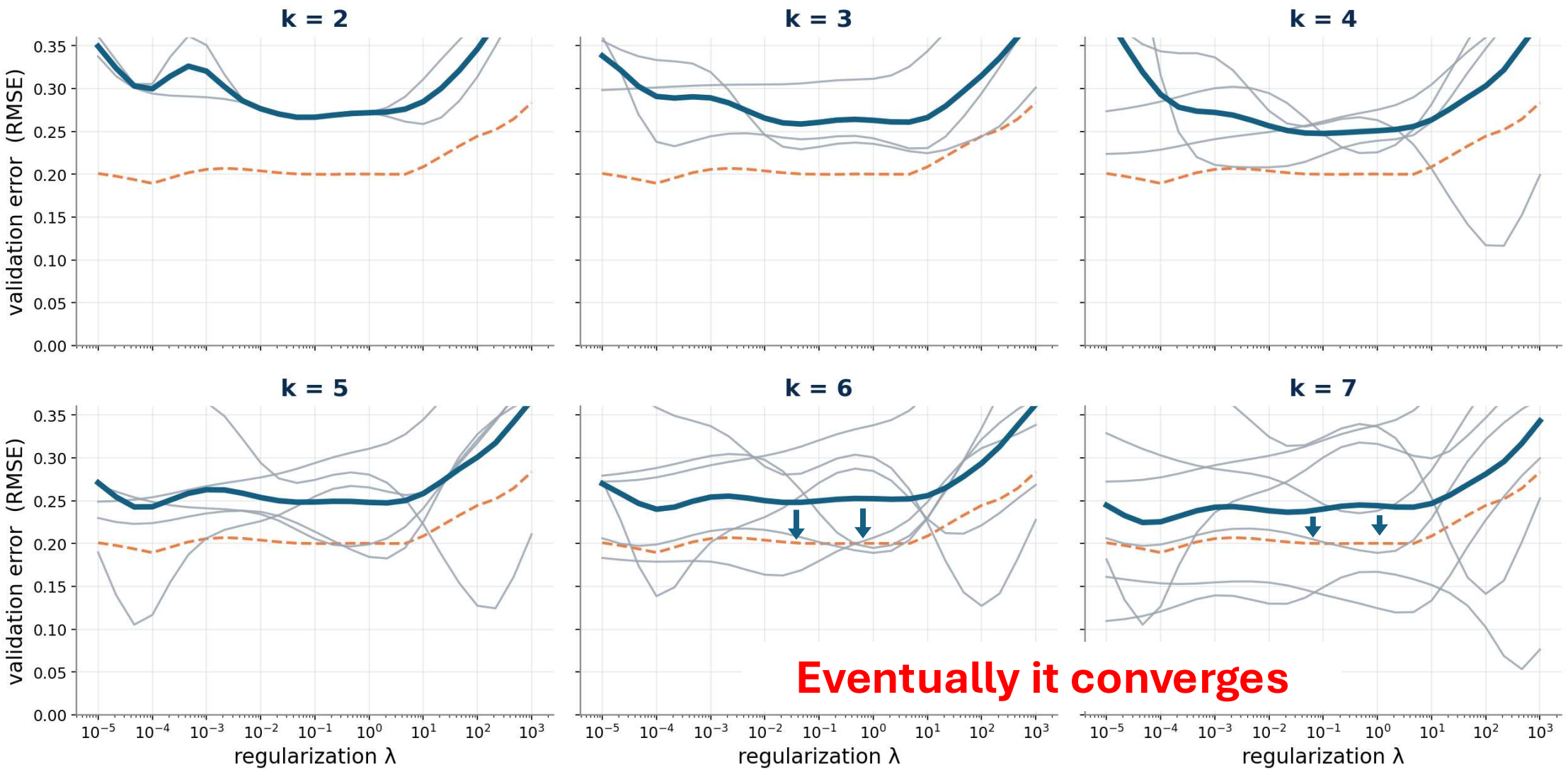


The extreme case: Leave-One-Out CV



- **Each single datapoint is its own validation fold** – train on *all other* datapoints
- So, if we have $\boldsymbol{n}$ **points**, we **train our model** $\boldsymbol{n}$ **times** – each time on the entire dataset with just one point held out. Hence the name: **LOOCV**

Increasing k converges to LOOCV curve



LOOCV: Pros and Cons

1. You use (basically) the entire dataset to train, so there should be minimal bias due to “missing” data

2. The splitting is now deterministic – no randomness, as the order of samples doesn’t matter

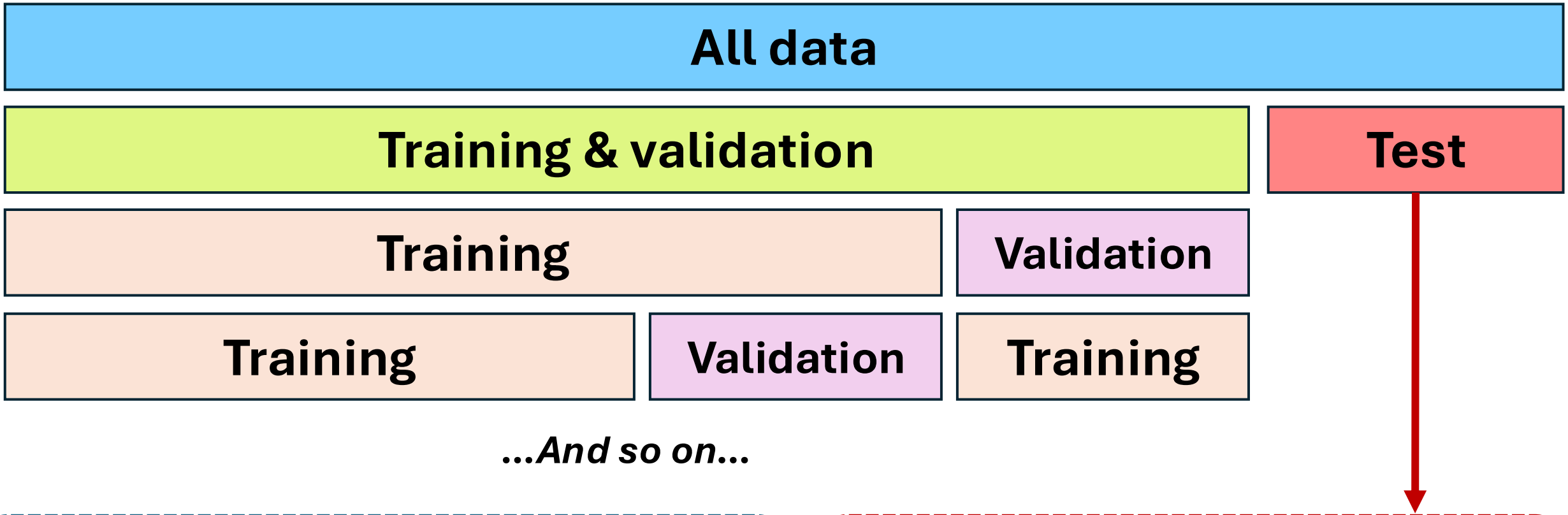
3. Many folds → **good statistics, less variance in the avg.**



1. **Computationally even *more* expensive** since we have to train each model n times…This is impractical when n is very large (say, > 100 samples)



Another variant: Nested cross-validation

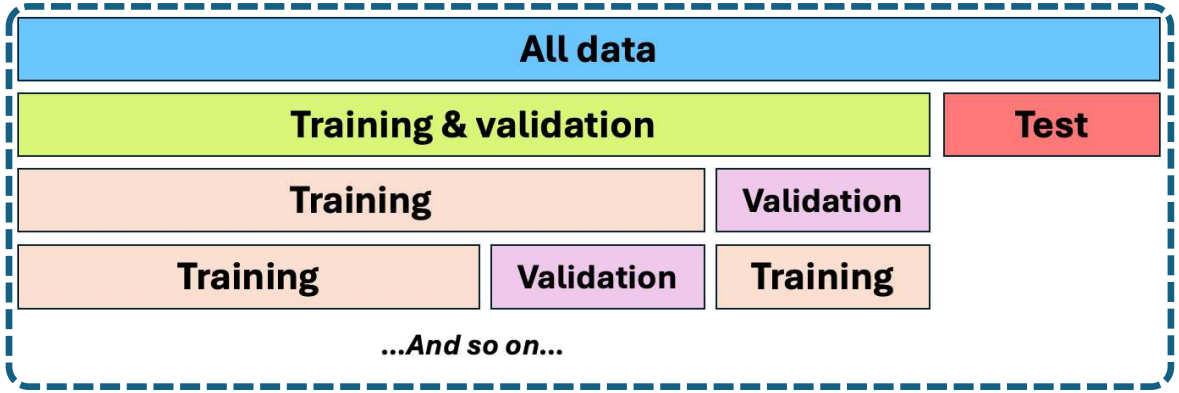


But what if our dataset is small, and we'd like to use the test data, without cheating?

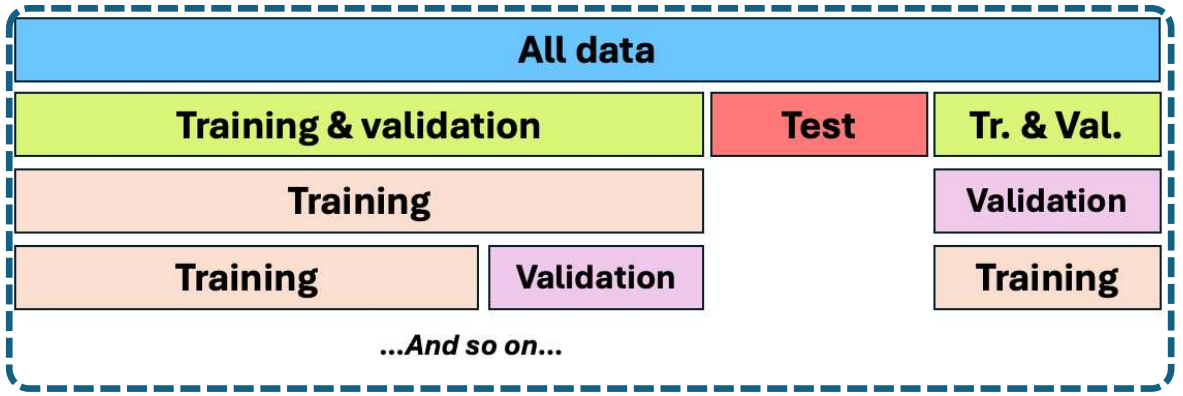
Remember: we evaluate on the test set only once. It has **no role in the hyperparameter selection.**

Another variant: Nested cross-validation

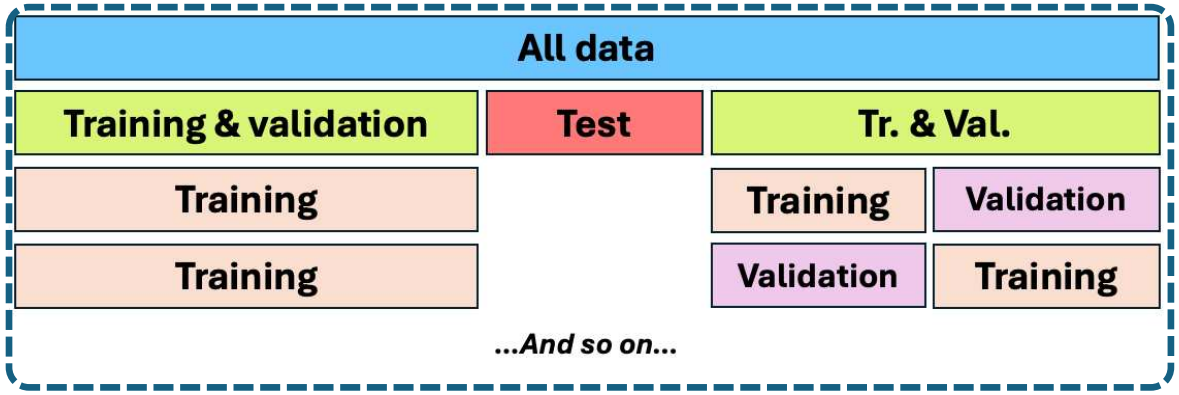
Outer fold #1



Outer fold #2



Outer fold #3

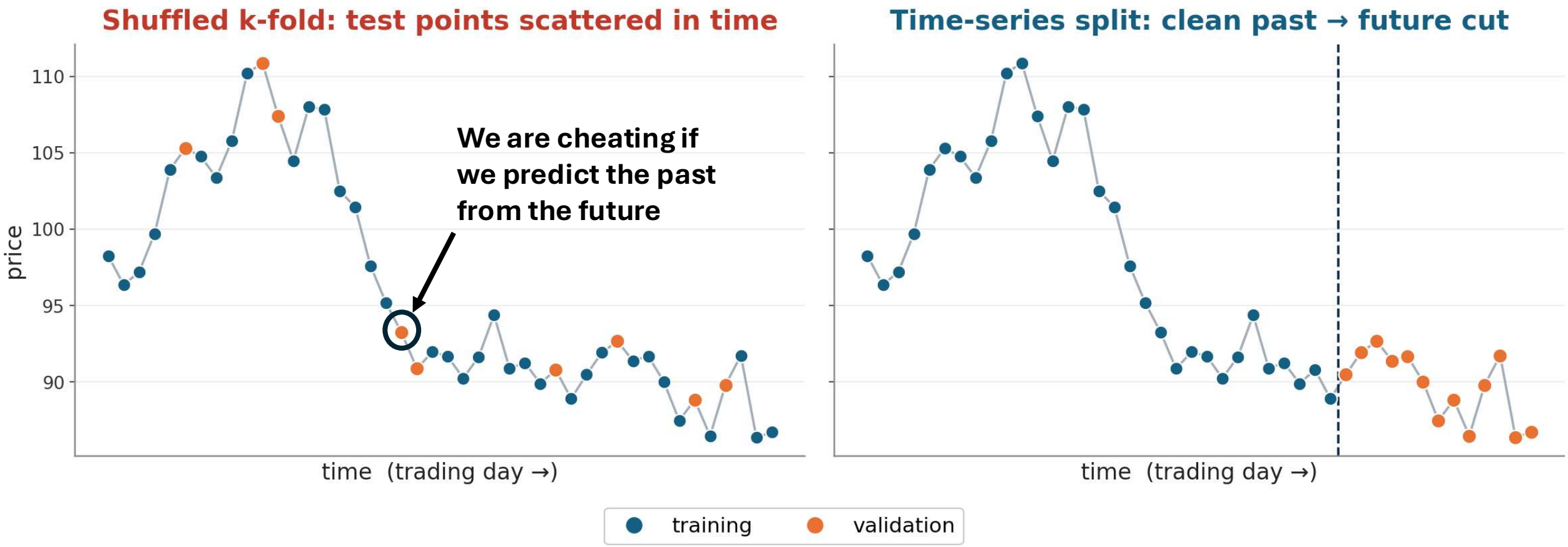


...And so on...

Outer folds are used *only* to estimate generalization error, **not to choose hyperparameters!** Those are chosen at the end, doing CV on *all* the data

Generally used for **small datasets** where you can't afford to sacrifice the test data

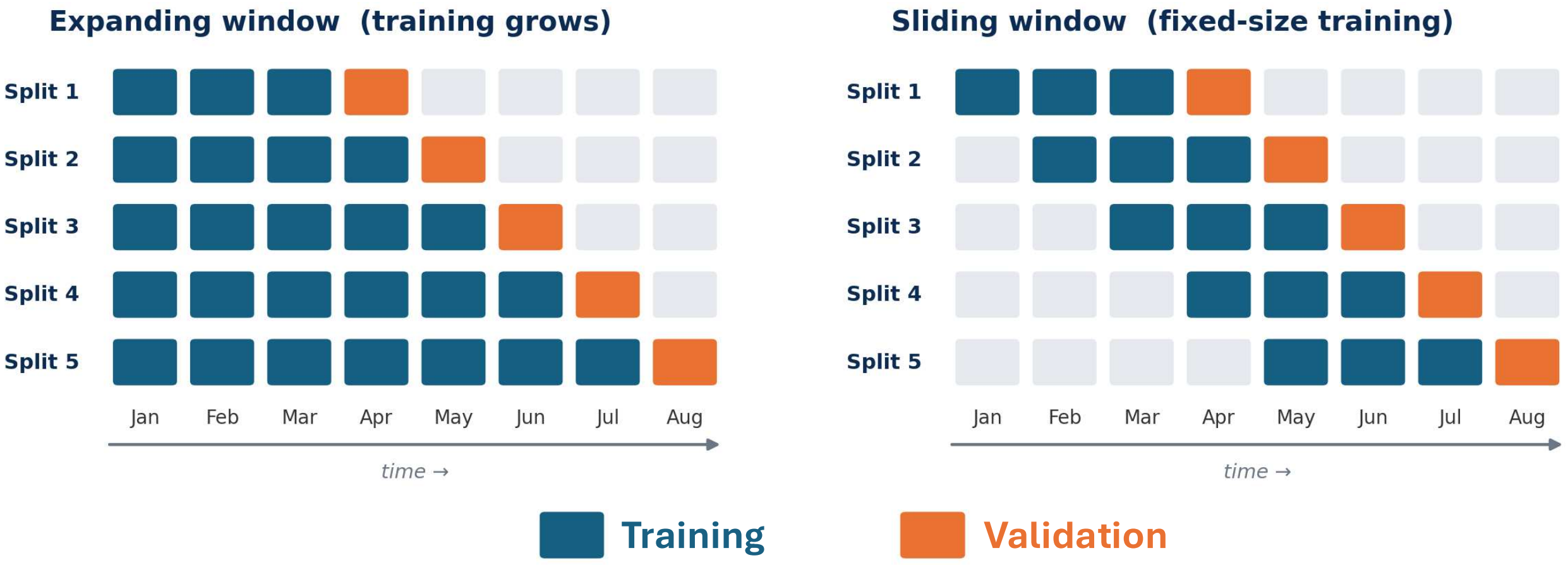
One more variant: Time series split



A **time series** is a sequence of observations collected in **chronological order**, where the order of the data carries meaningful information.

One more variant: Time series split

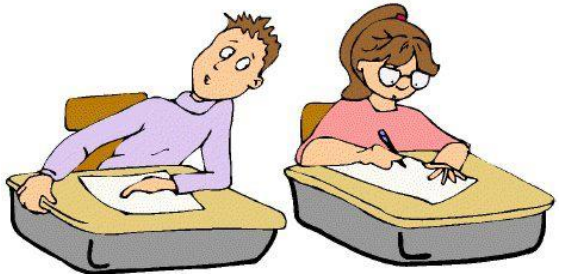
You should always train on the past, then validate/test on the future

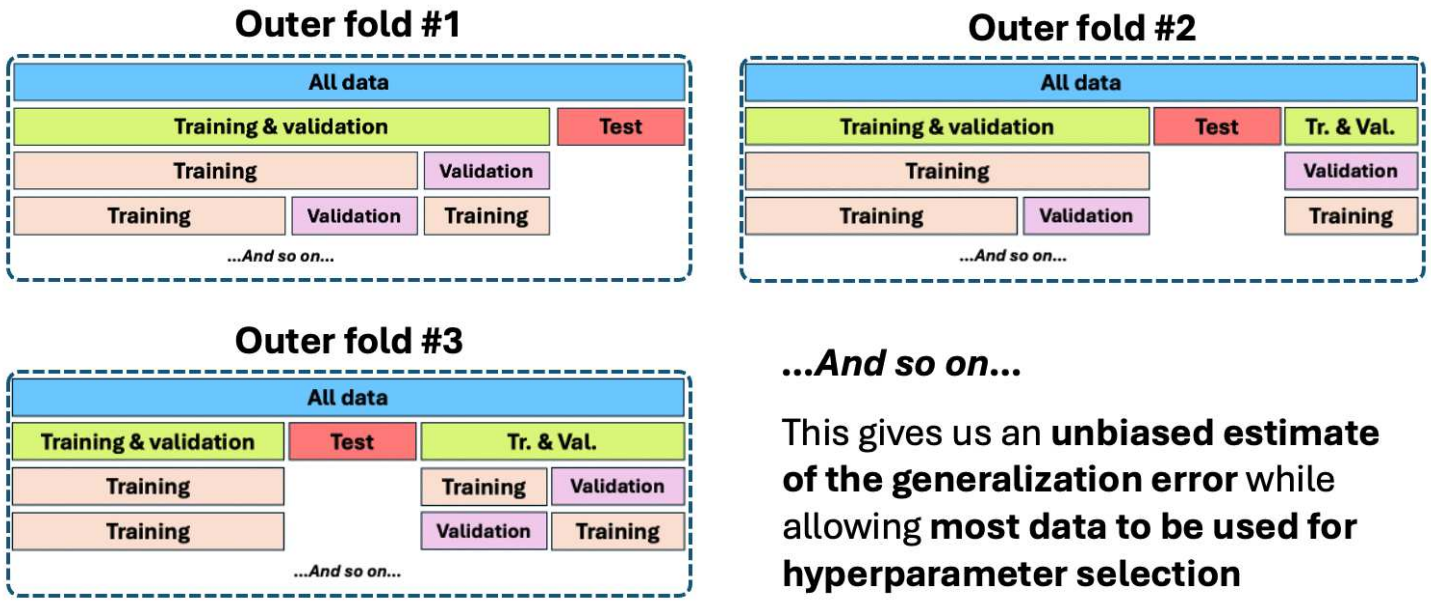


Common pitfalls when doing cross-validation

1) Using the test set during model selection

No matter which cross-validation method you use, ***never* use the test set to choose the best model or hyperparameters**





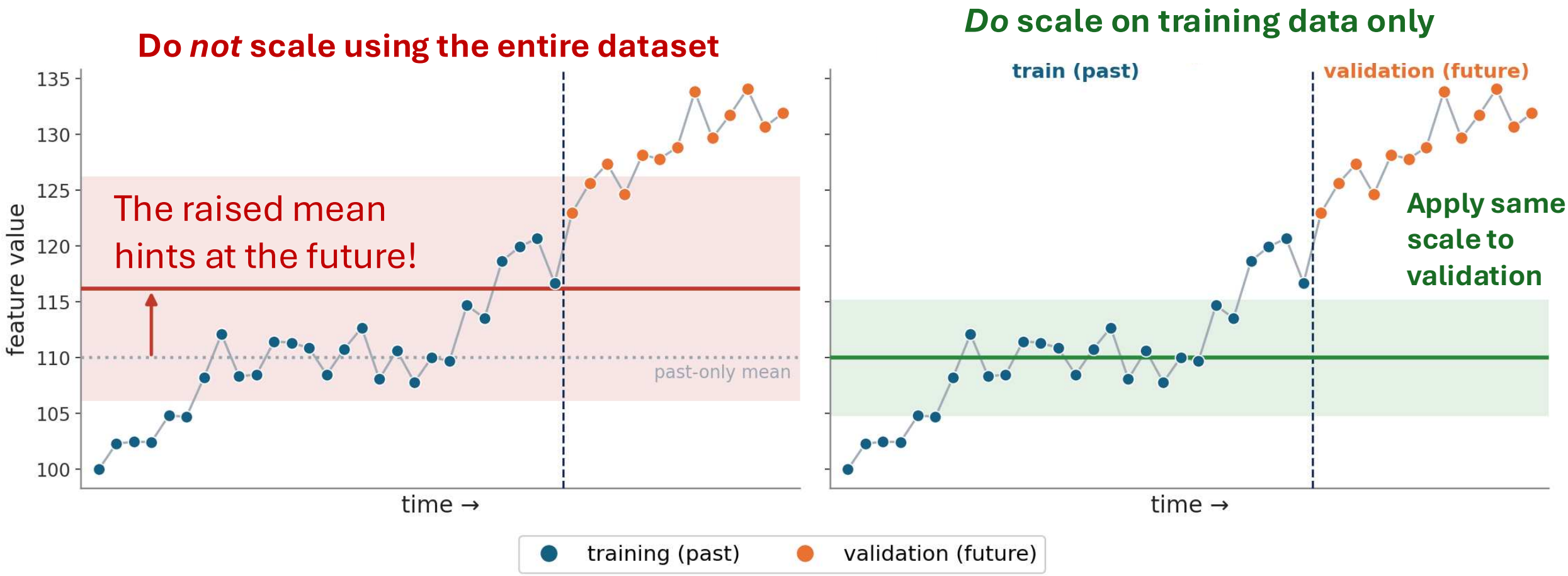
...And so on...

This gives us an **unbiased estimate of the generalization error** while allowing **most data to be used for hyperparameter selection**

Even in Nested CV, we do not use the test set to choose hyperparameters. **The outer folds are used only to estimate generalization error.**

Common pitfalls when doing cross-validation

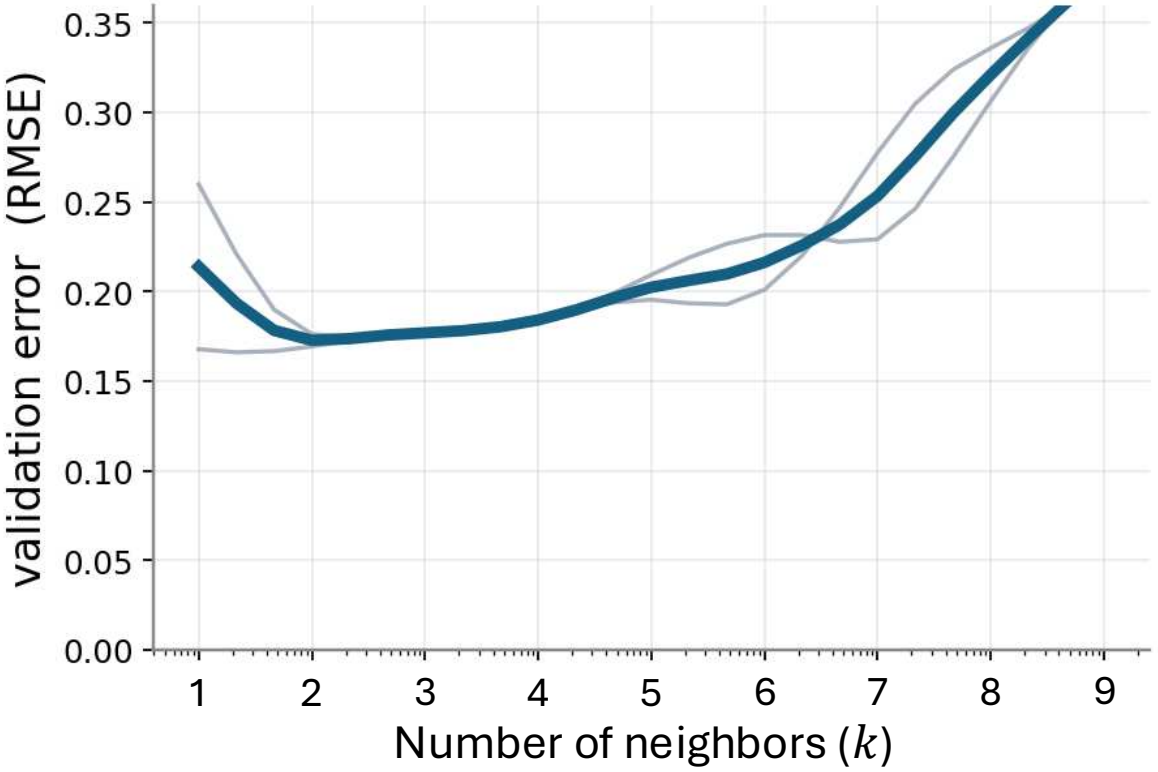
2) Performing data preprocessing before cross-validation



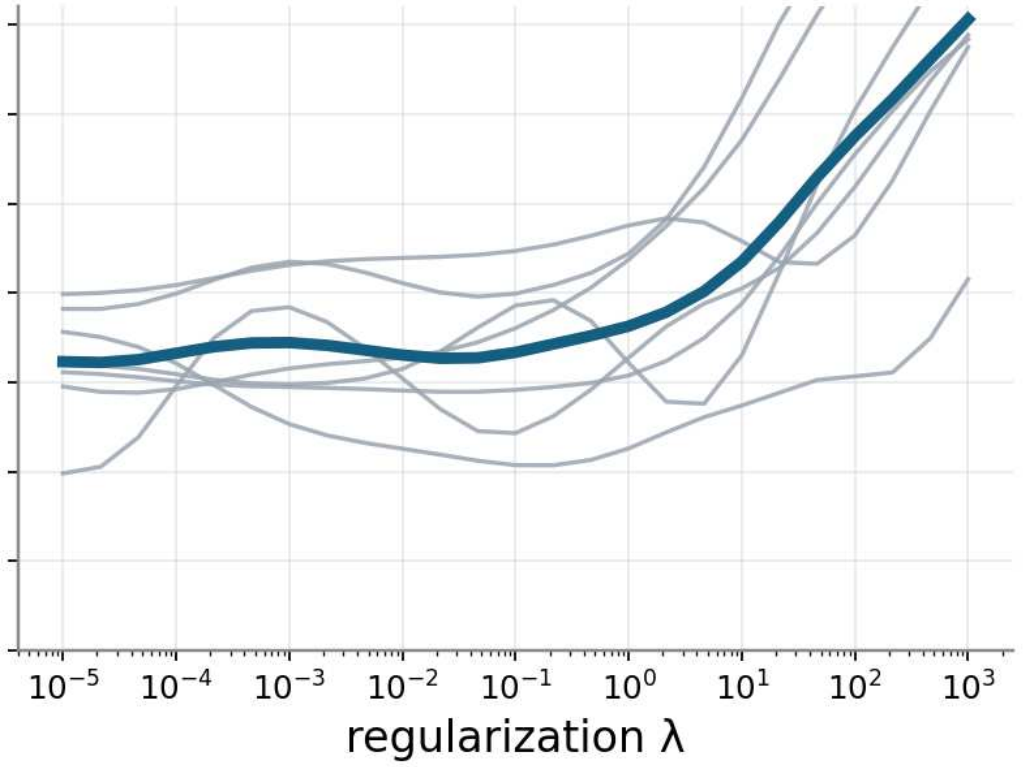
Common pitfalls when doing cross-validation

3) Using different cross-validation approaches for different models

KNN regression (2 folds)

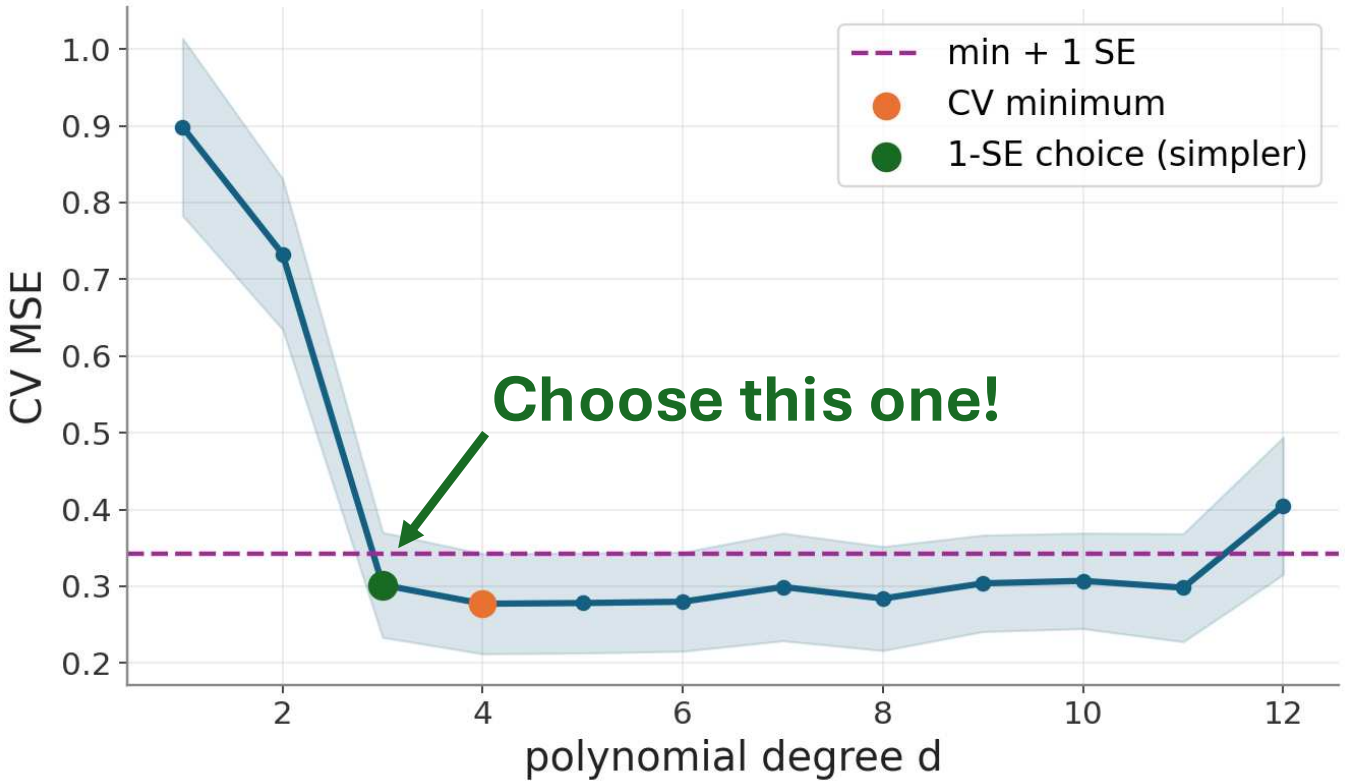


Ridge regression (7 folds)



Common pitfalls when doing cross-validation

4) Ignoring CV variability and choosing an unnecessarily complex model

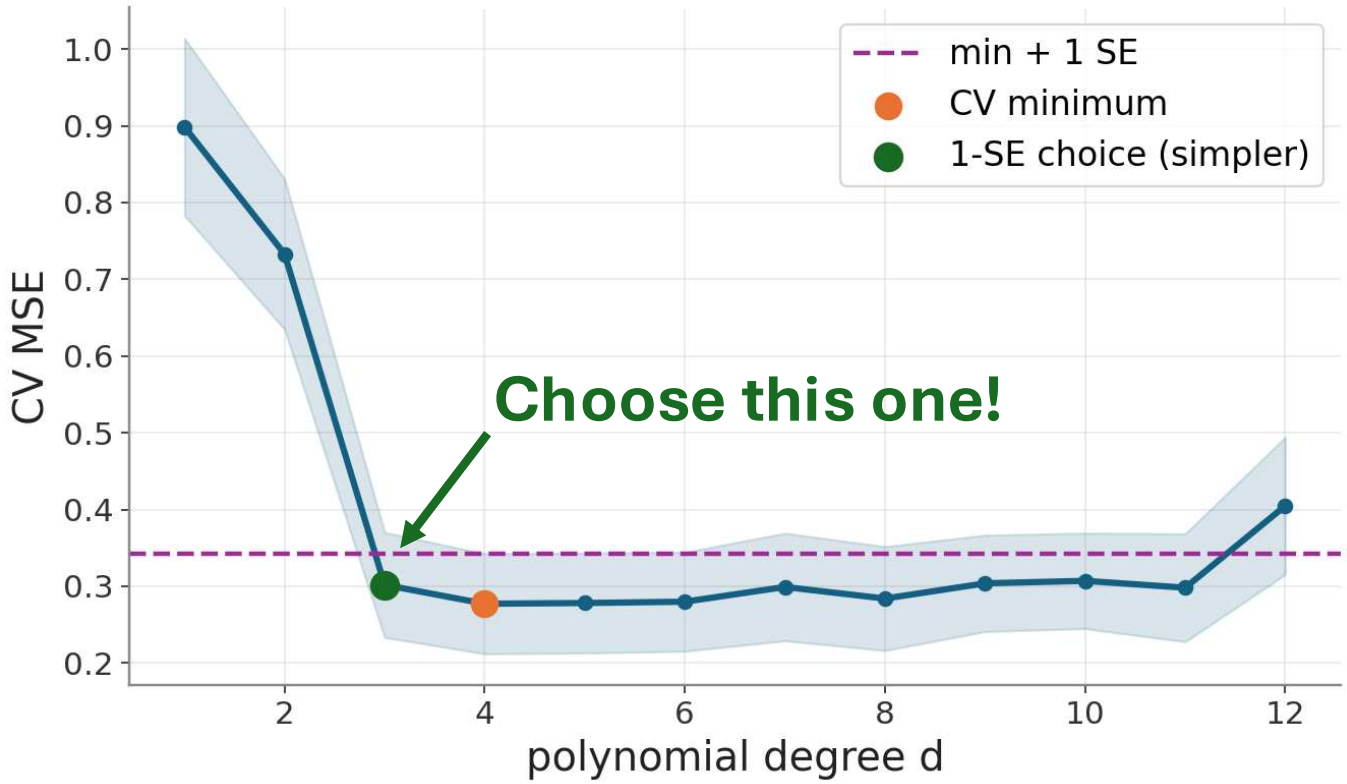


Among models within one standard deviation of the rest, **choose the simplest model.** This guards against overfitting. Plus, simpler models are usually more interpretable.

One-standard-error rule: Choose the simplest model whose CV error is within 1 SE of the minimum.

Common pitfalls when doing cross-validation

4) Ignoring CV variability and choosing an unnecessarily complex model



$$SE = \frac{SD_{\text{across folds}}}{\sqrt{K}}$$ ← Num folds

$$\sqrt{\frac{1}{K-1}\sum_{k=1}^{K}\left(MSE_k - MSE_{\text{avg}}\right)^2}$$

One-standard-error rule: Choose the simplest model whose CV error is within **1 SE** of the minimum.

Questions?

Prediction focuses on the model **output**, assessed through cross-validation and testing

From Prediction to Inference

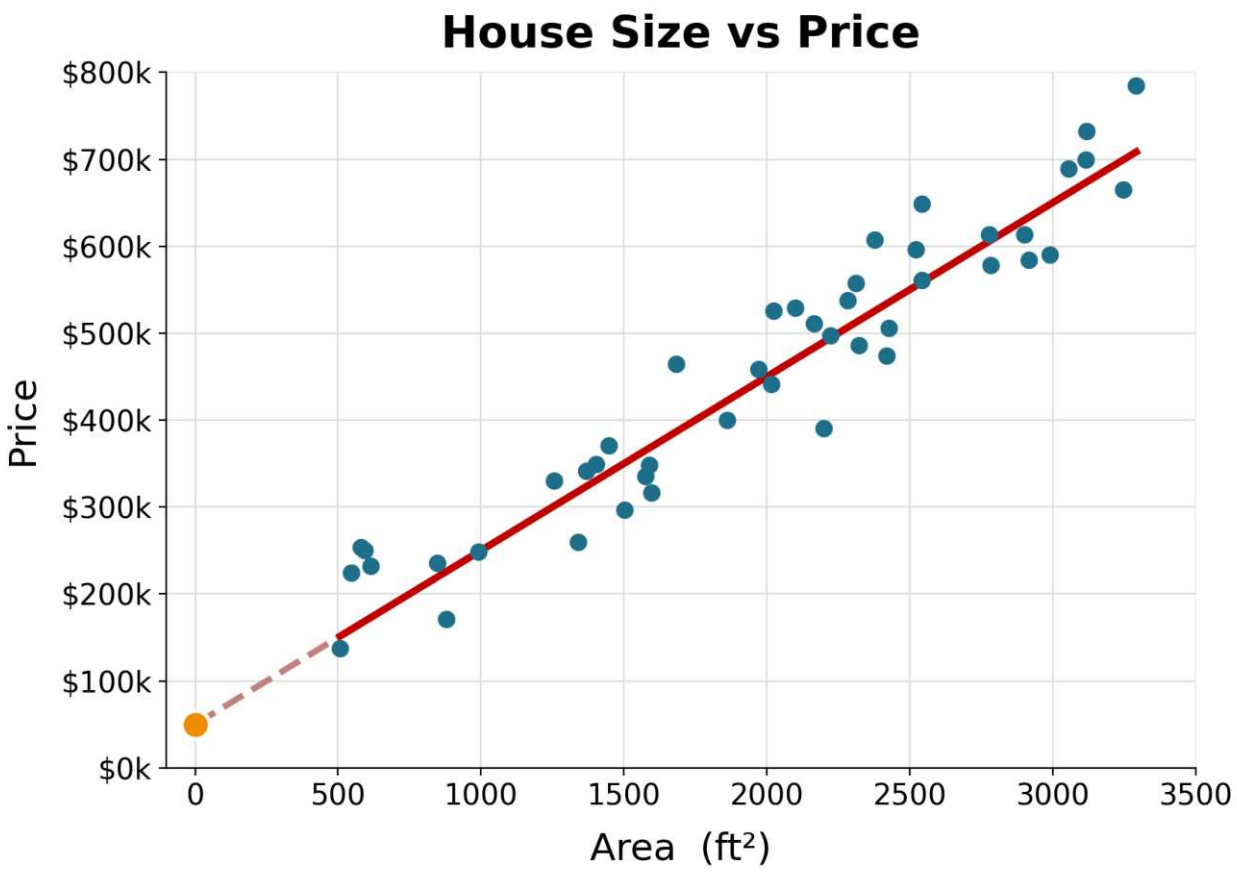
Which predictors really matter…And how sure are we?

When doing **inference**, we often care just as much about the **input features**, and whether (or by how much) they influence the output

Recap: Interpreting linear coefficients

$$\hat{y} = \beta_0 + \beta_1 x$$

Inference often starts with a question



Question: How does the size of a house (in ft) affect its sale price?

From our linear fit:

- **The intercept $(\beta_0) \approx$ \$50k**
  - In this case, the intercept is not physically meaningful
- **The slope $(\beta_1) \approx$ \$200/ft²**
  - This means we pay \$200 more for every added sq. ft.

Recap: Interpreting linear coefficients

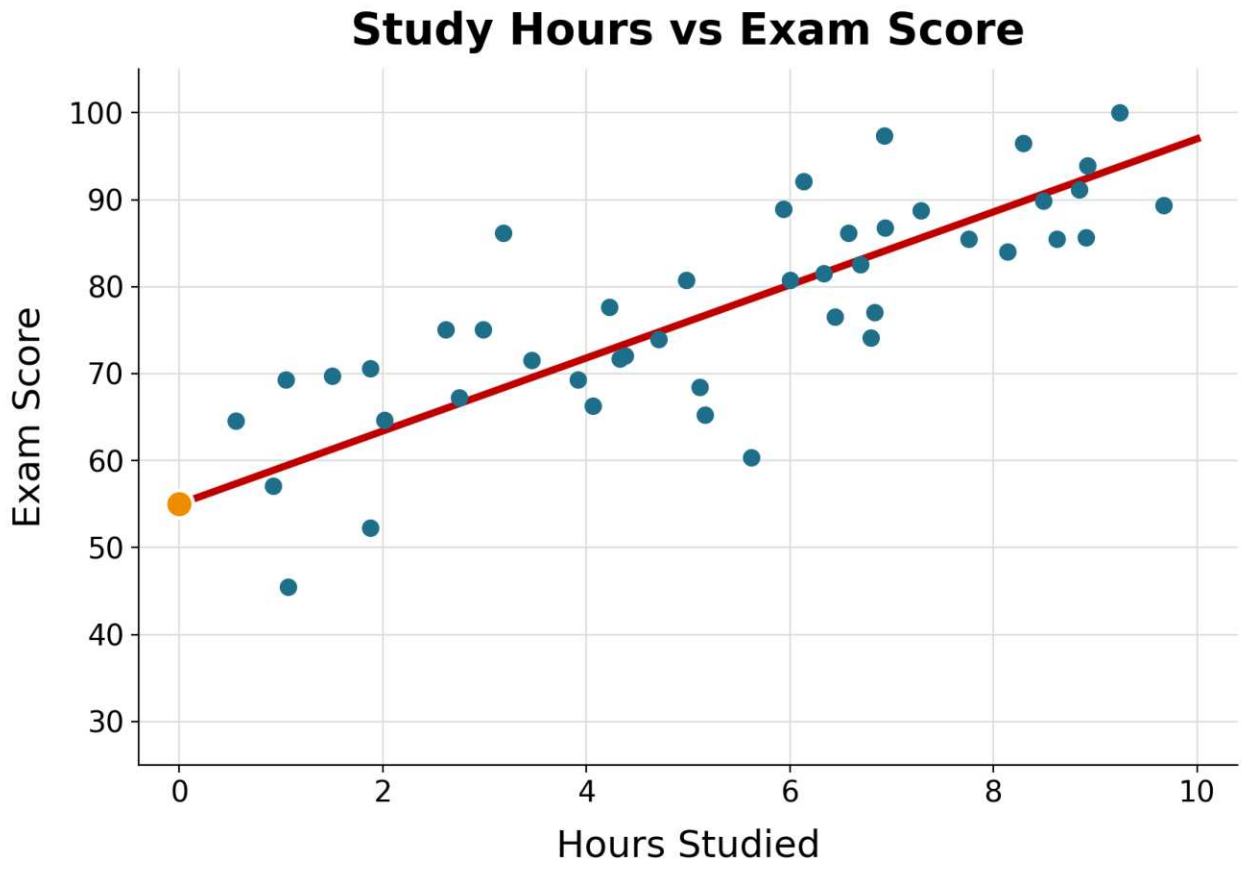
$$\hat{y} = \beta_0 + \beta_1 x$$

Inference often starts with a question

Question: Does studying more lead to better exam scores?

From our linear fit:

- **The intercept $(\beta_0) \approx 55\%$**
  - Here, the intercept *is* meaningful (score without studying)
- **The slope $(\beta_1) \approx 4\%/hr$**
  - The positive slope suggests that **yes, studying improve scores**



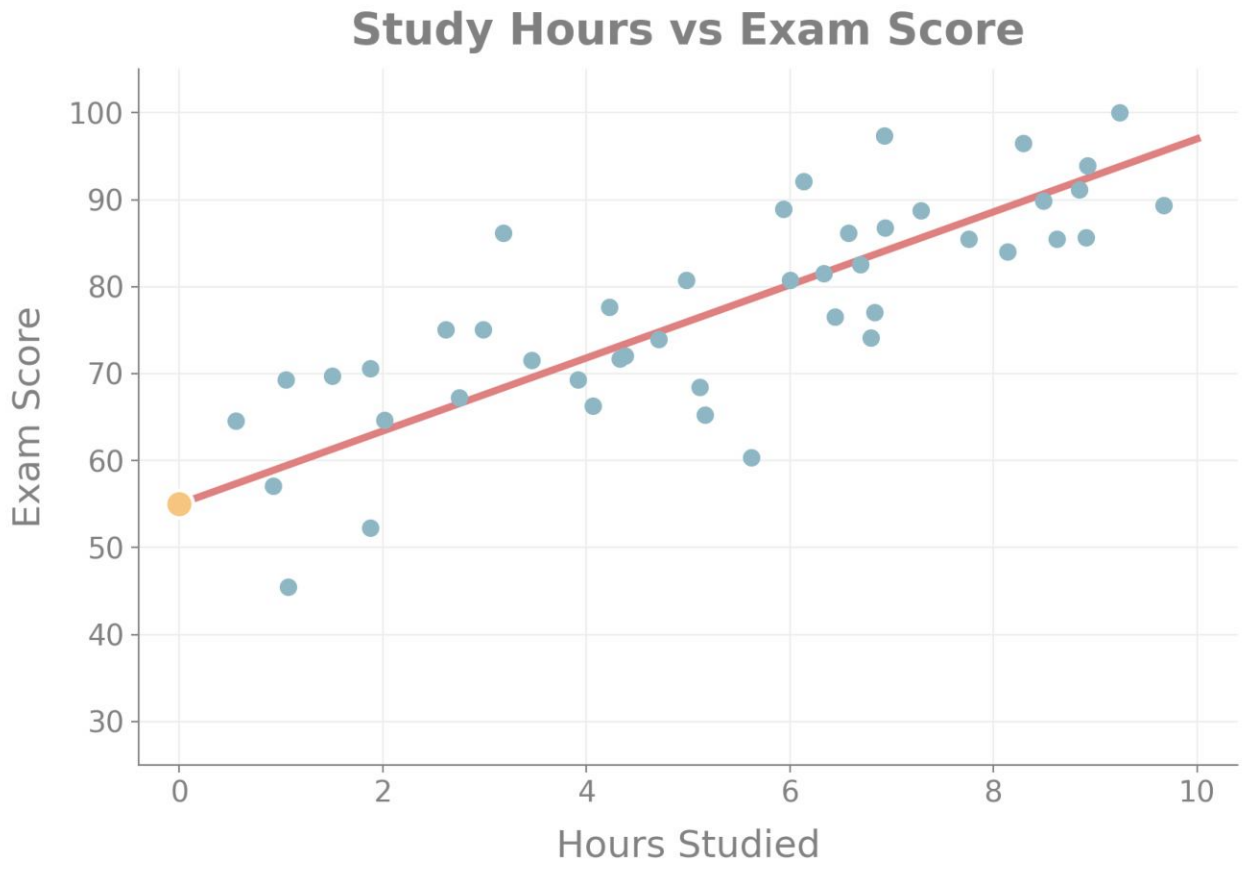
Recap: *Variability* in linear coefficients

How confident are we in our answer to the question? We've seen from cross-val that **results can vary** quite a bit **depending on which data we sample**…
So, is this positive slope "real"? Or did we get it by chance?

Question: Does studying more lead to better exam scores?

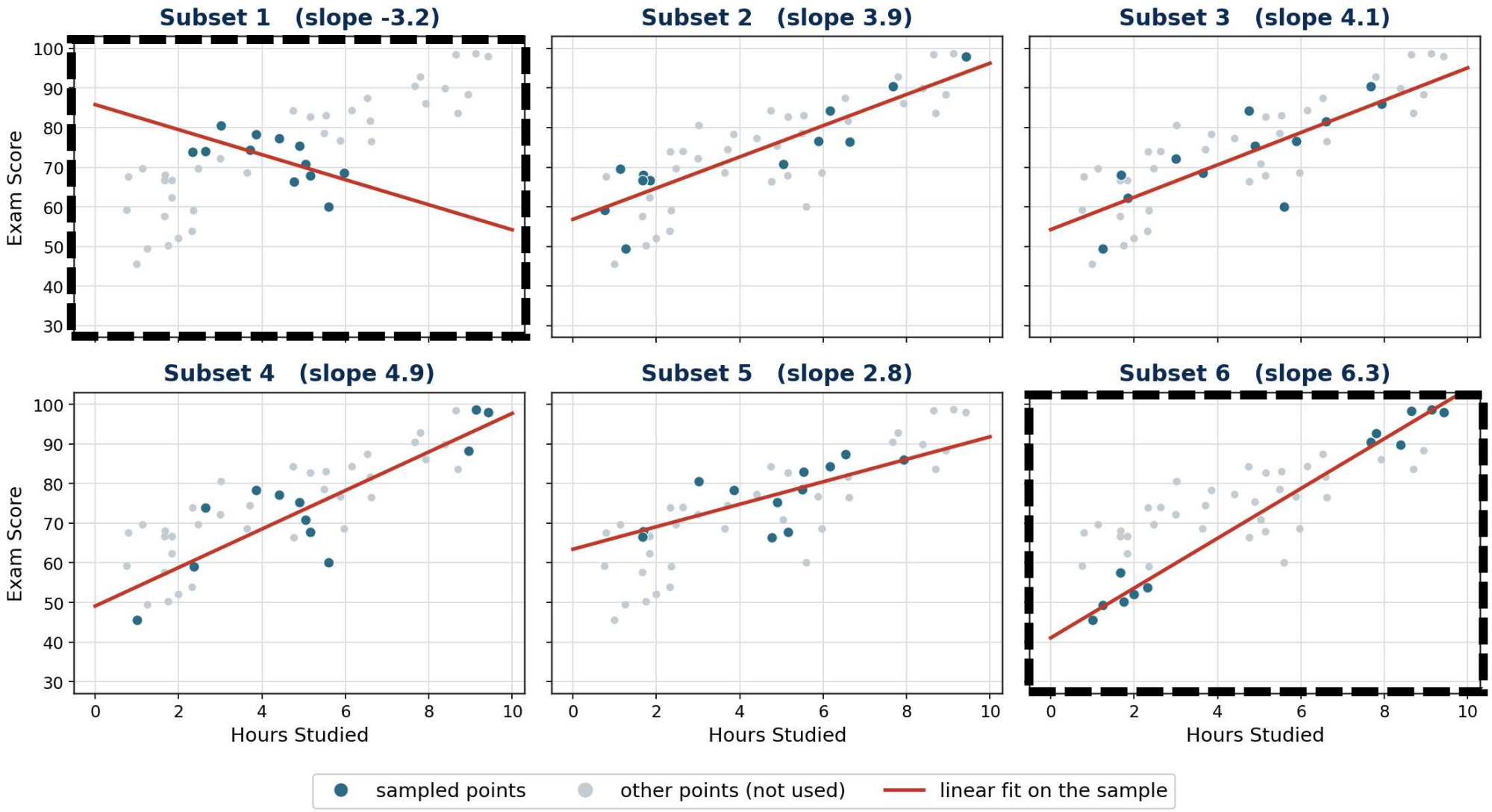
From our linear fit:

- **The intercept $(\beta_0) \approx 55\%$**
  - Here, the intercept *is* meaningful (score without studying)
- **The slope $(\beta_1) \approx 4\%/hr$**
  - The positive slope suggests that **yes, studying improve scores**



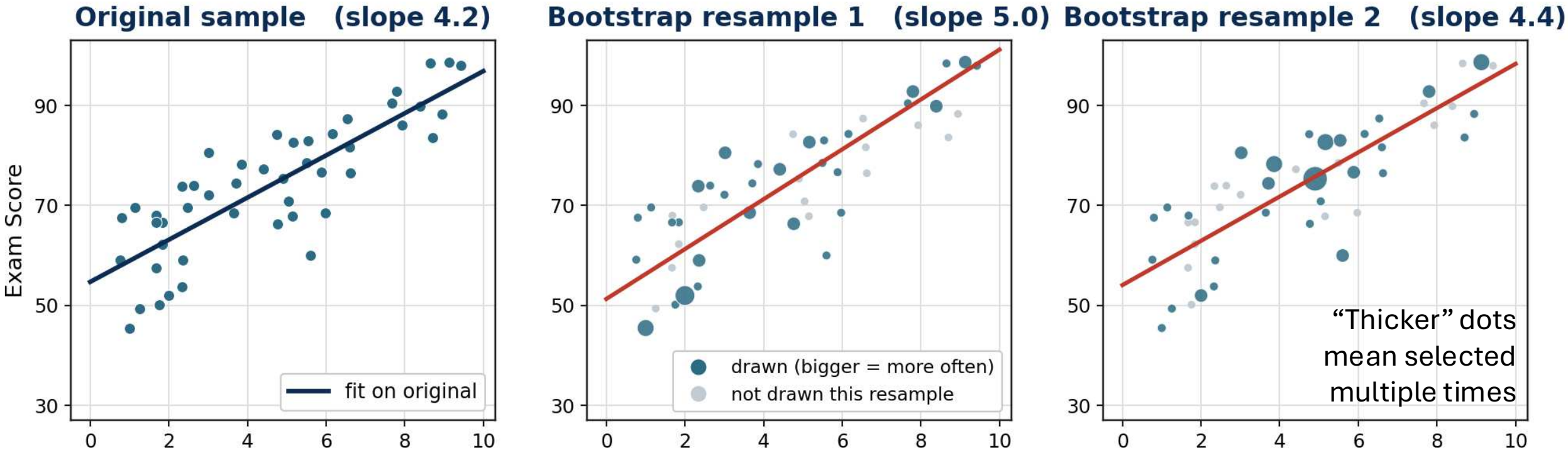
Different subsets → different conclusions!

“I guess studying hurts exam scores?



“Studying improves scores by a lot!”

We quantify variability through *bootstrapping*



Bootstrapping repeatedly creates new datasets by **randomly sampling** the original data **with replacement**, meaning some observations may be selected multiple times while others are omitted, then refits the model to estimate the variability of the fitted coefficients.

We quantify variability through *bootstrapping*

Original dataset: A B C D E (5 samples)

Bootstrapped set #1: D C B A B (5 samples)

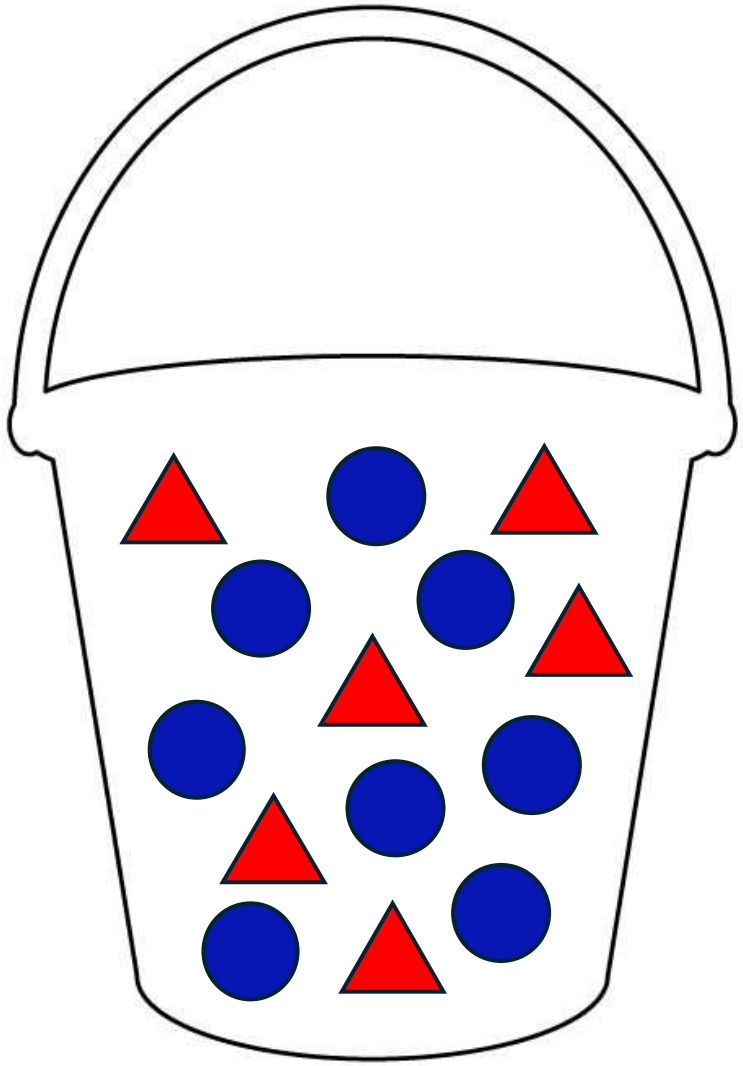
Bootstrapped set #2: A D A E E (5 samples)

...And so on...

We **randomly re-draw** from the original sample each time, **each time replacing the chosen sample**, to achieve variety while maintaining original sample size

Bootstrapping repeatedly creates new datasets by **randomly sampling** the original data **with replacement**, meaning some observations may be selected multiple times while others are omitted, then refits the model to estimate the variability of the fitted coefficients.

Another way to visualize bootstrapping

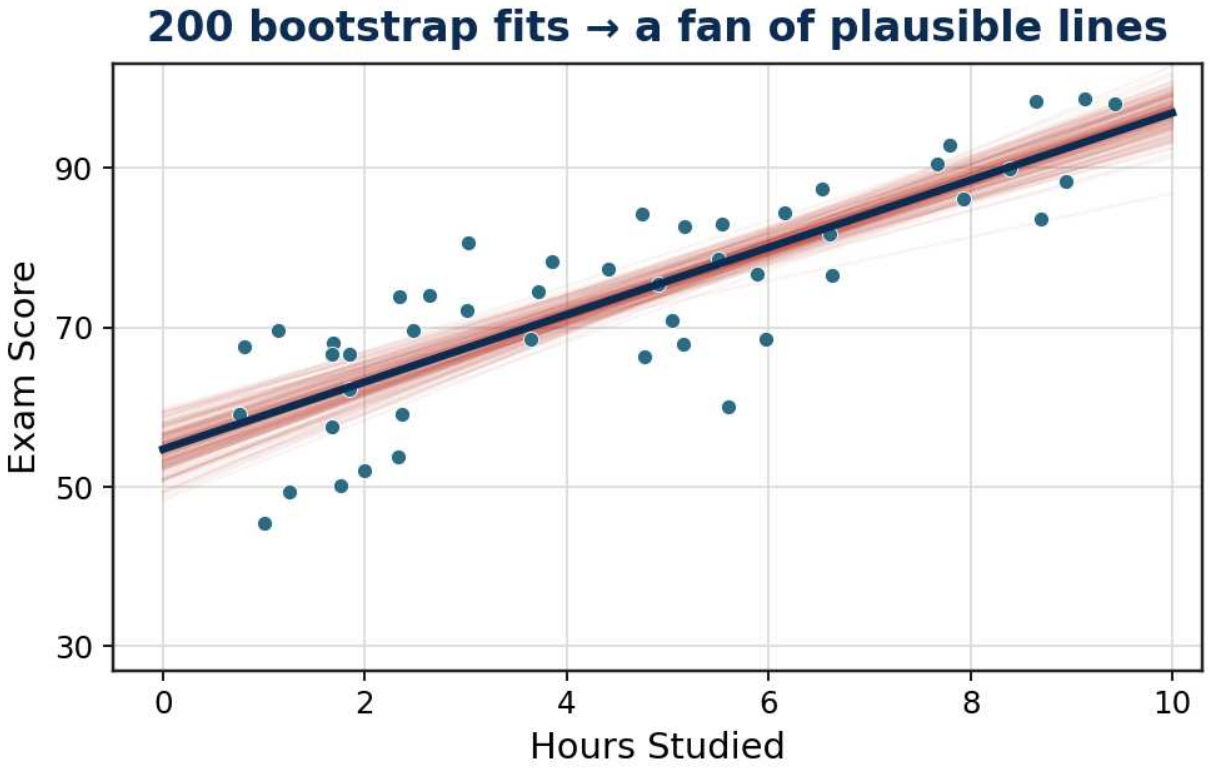


I randomly choose a sample from this bucket

Then I put it back

And randomly choose again…
Until I've selected enough shapes to reach my total dataset size

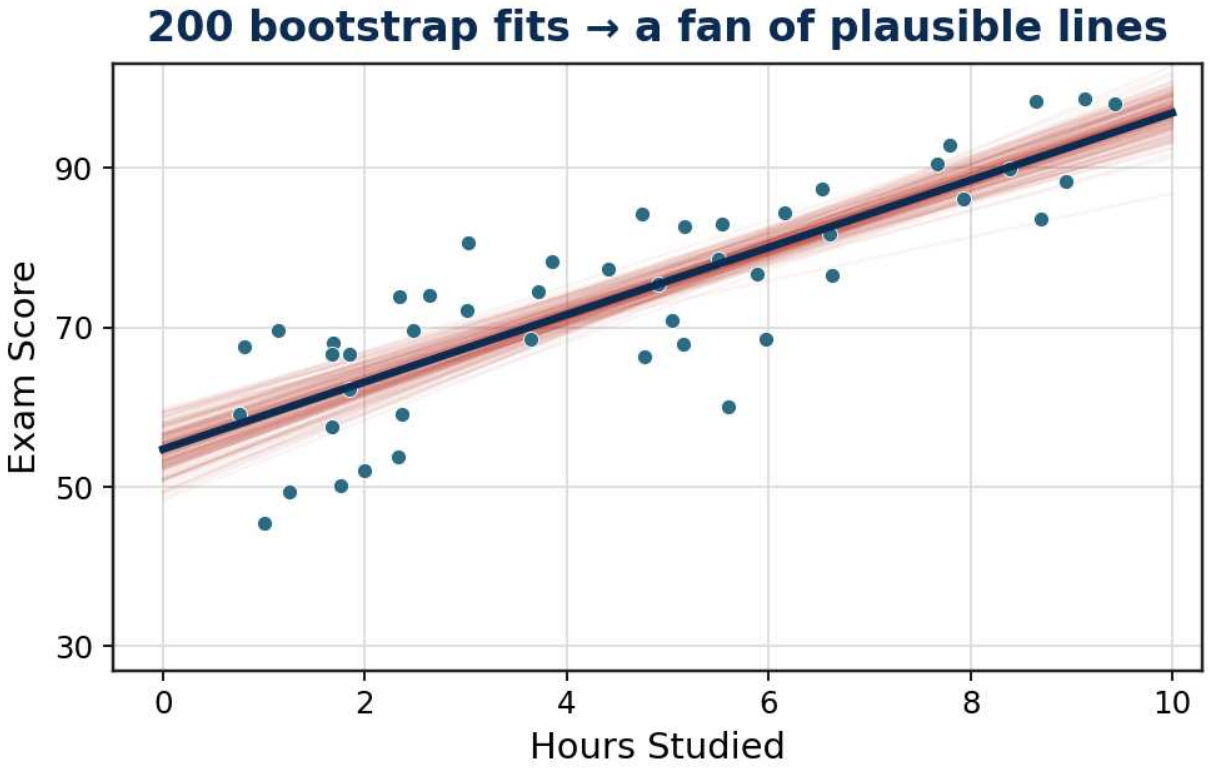
Bootstrapping is used to quantify variability

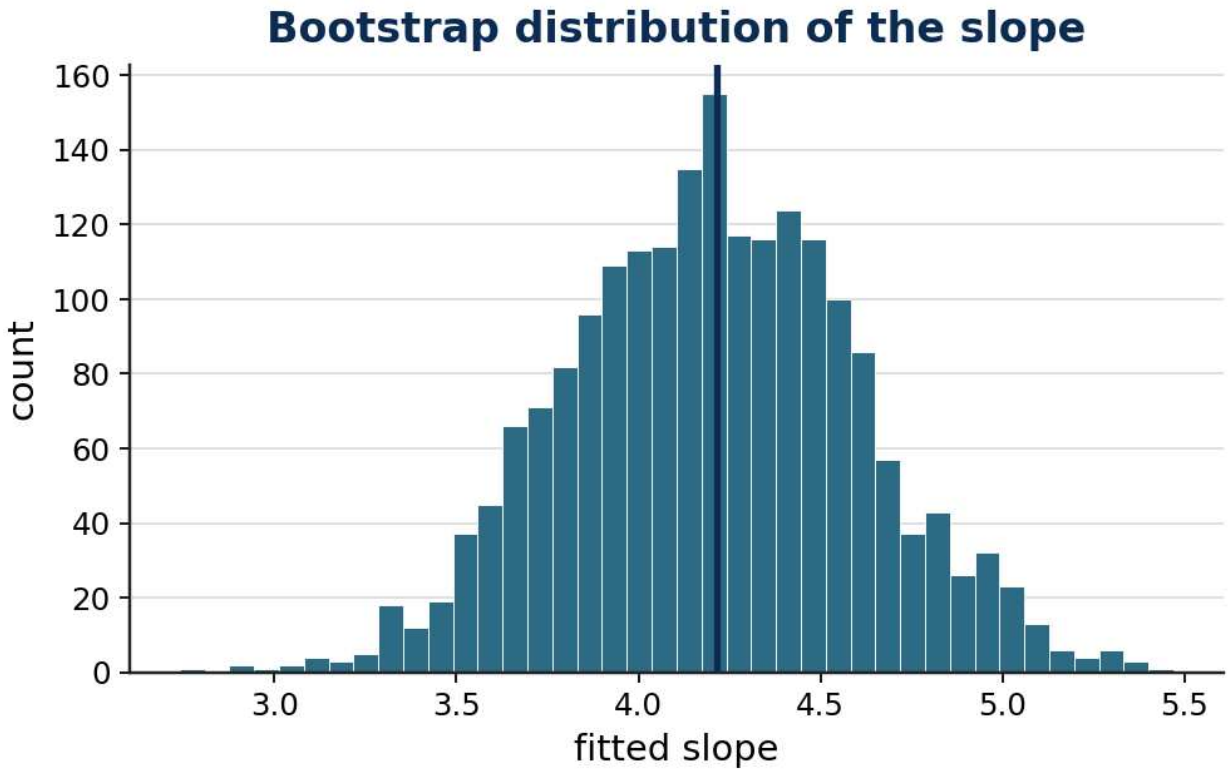


There is no strict rule on how many times you need to bootstrap, but anywhere from a few hundred to a few thousand is typical

Linear regression is cheap, so this is no big deal! It will matter more when dealing with neural networks…

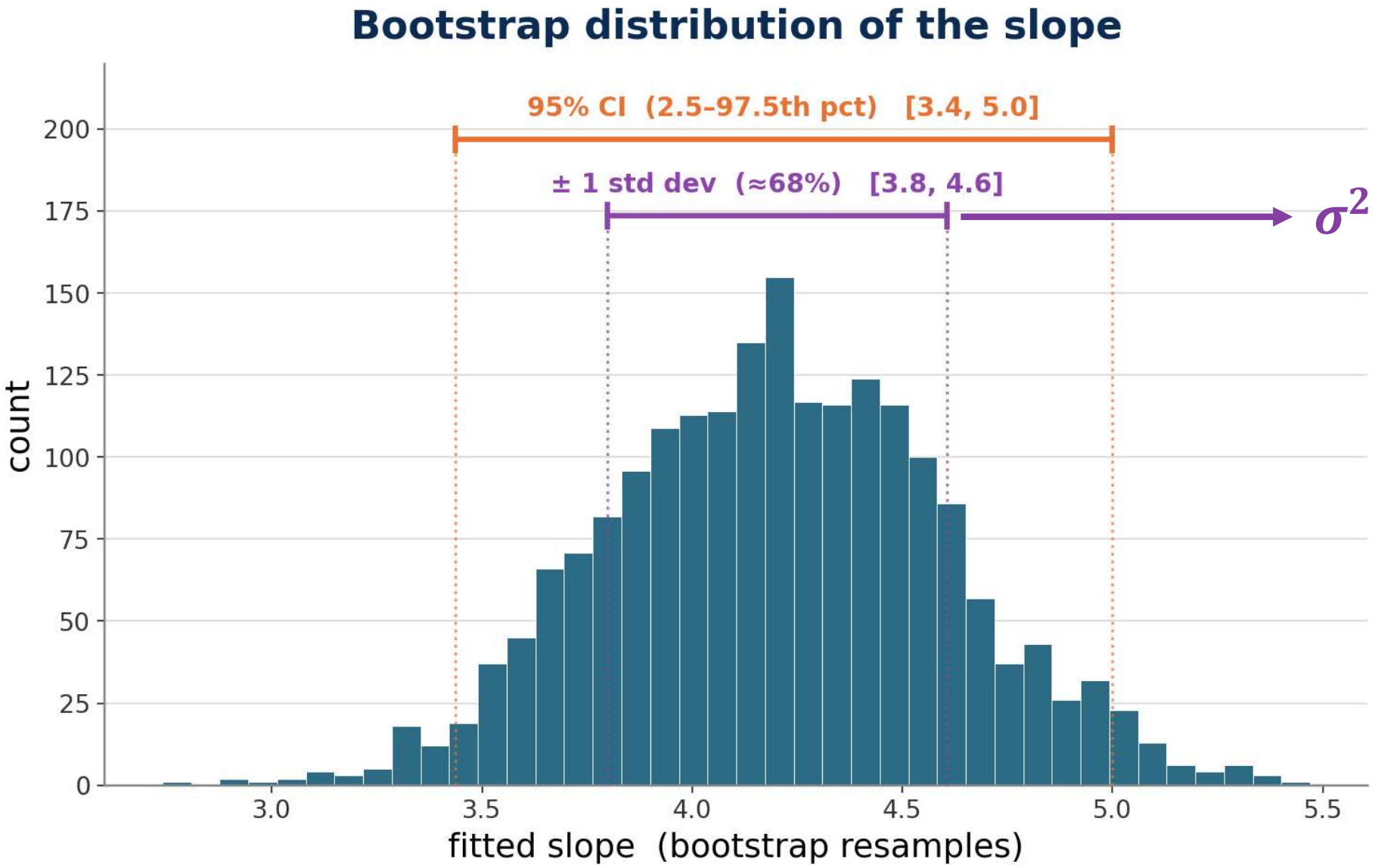
Bootstrapping is used to quantify variability





Fitting 200 models on these bootstrapped datasets leads to a **distribution of the coefficients (β_i) – not just one**

Standard error and confidence intervals



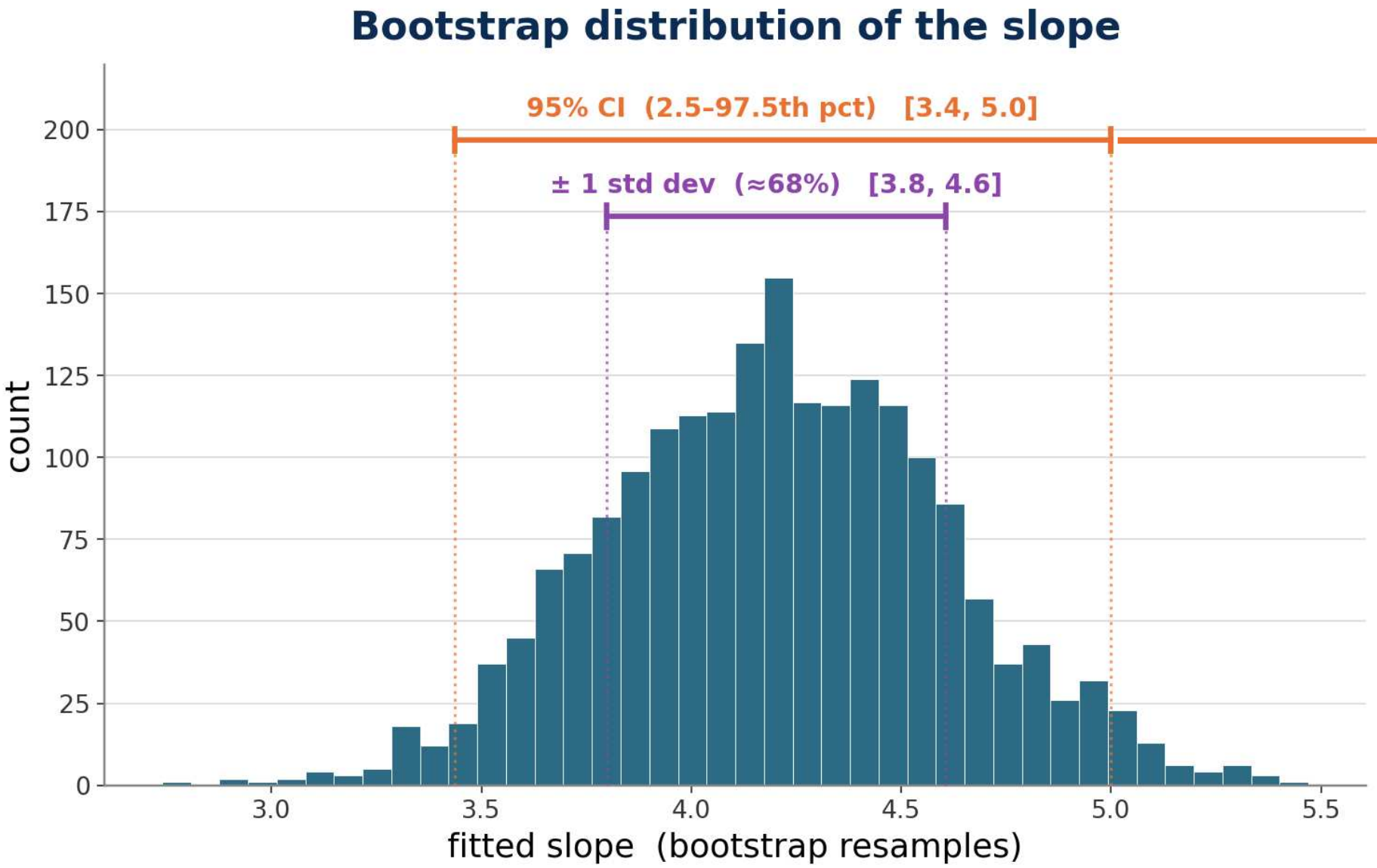
Each bootstrapped coefficient

$$\sigma^2 = \frac{1}{n-1}\sum_{i=1}^{n}(\beta_i - \mu)^2$$

Mean

This estimates the **standard error (SE)** which we discussed previously – but this expression is more general than just linear regression

Standard error and confidence intervals

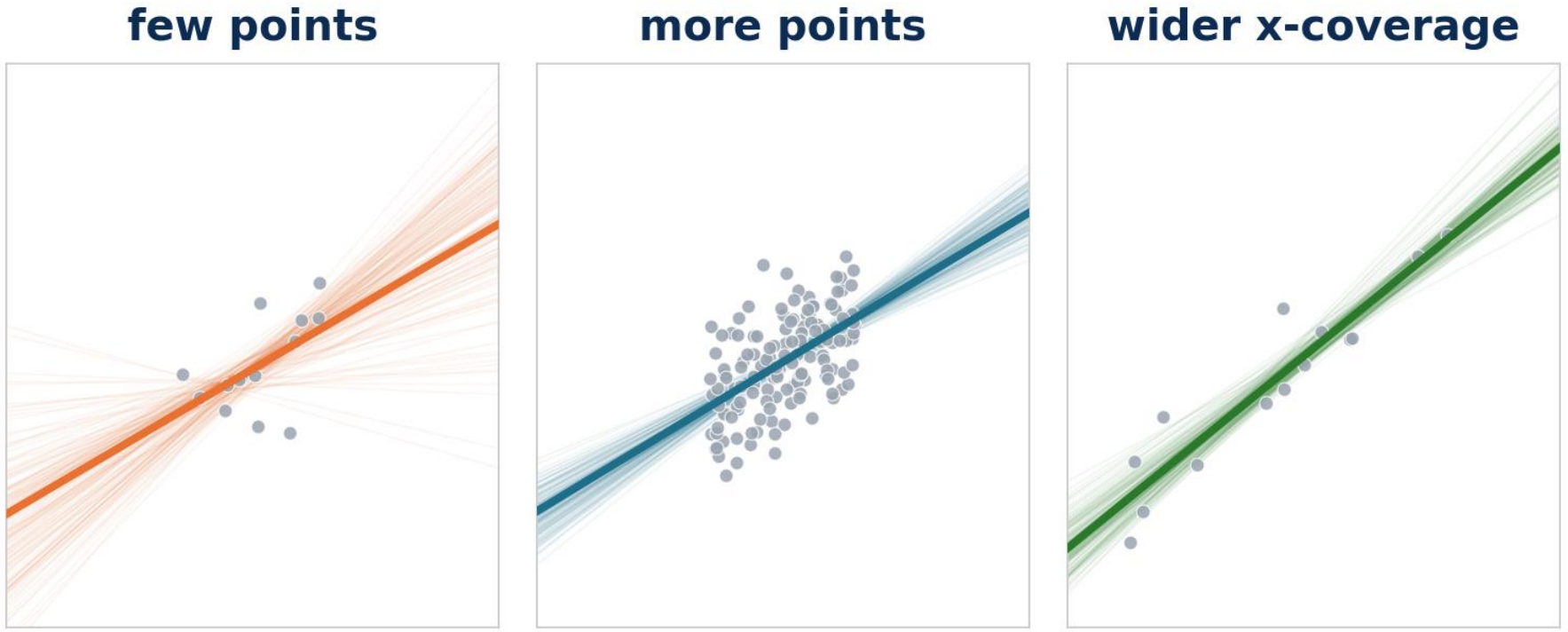


The 95% confidence interval (CI) is the range that cuts off the lowest and highest 2.5%

Approximately:

$\beta_i \pm 2 \cdot \mathbf{SE}$

How to minimize the standard error



$$\sigma^2 = \frac{1}{n-1}\sum_{i=1}^{n}(\beta_i - \mu)^2$$

Approximately:

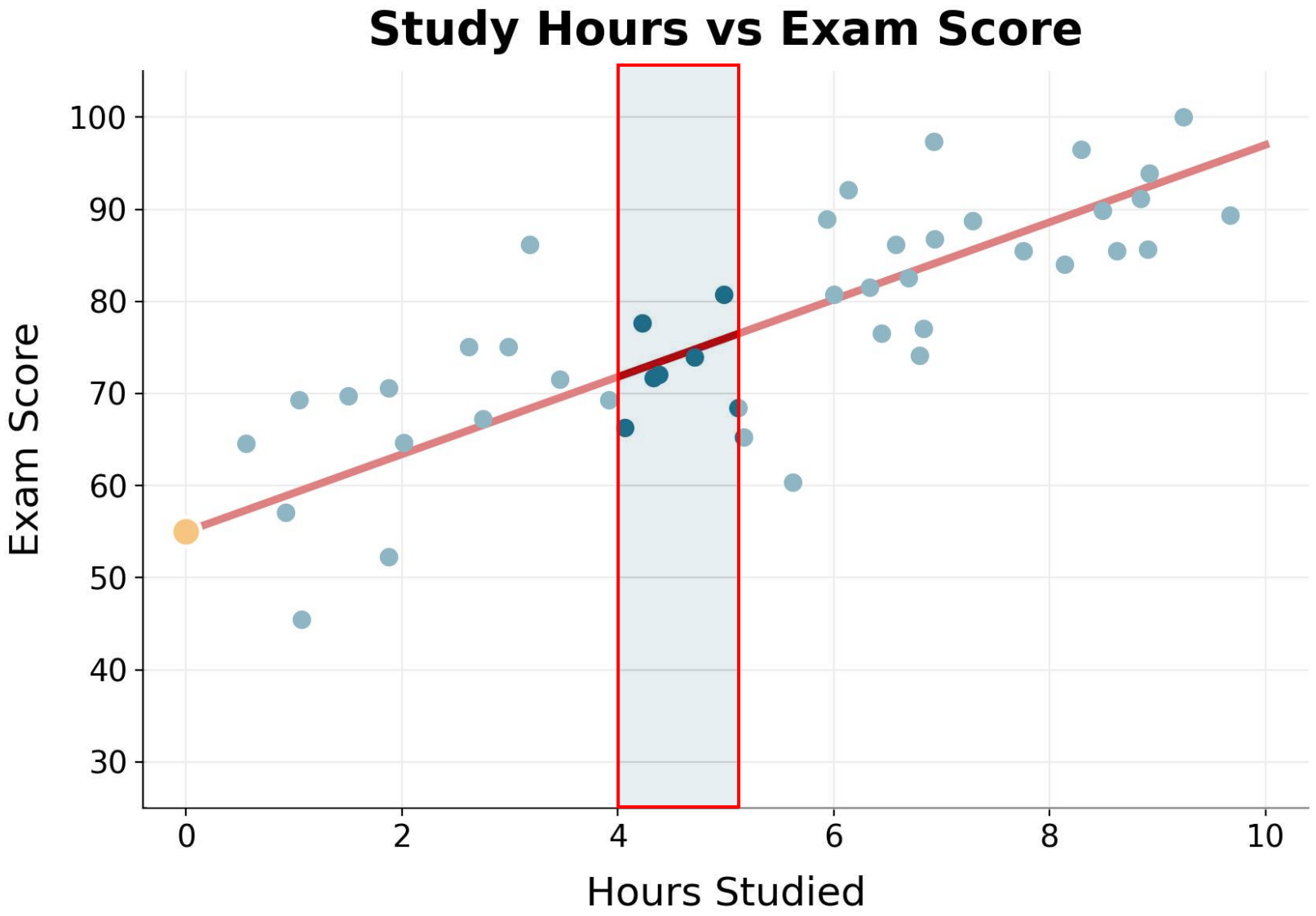
$$\beta_i \pm 2 \cdot \mathrm{SE}$$

If we want to **minimize standard error**, we can either:

1) Collect *more* data
2) Collect data over a *wider range* of predictors

Smaller SE and narrower CI → more reliable estimates!

Why more *diverse* data is better



Question: Does studying more lead to better exam scores?

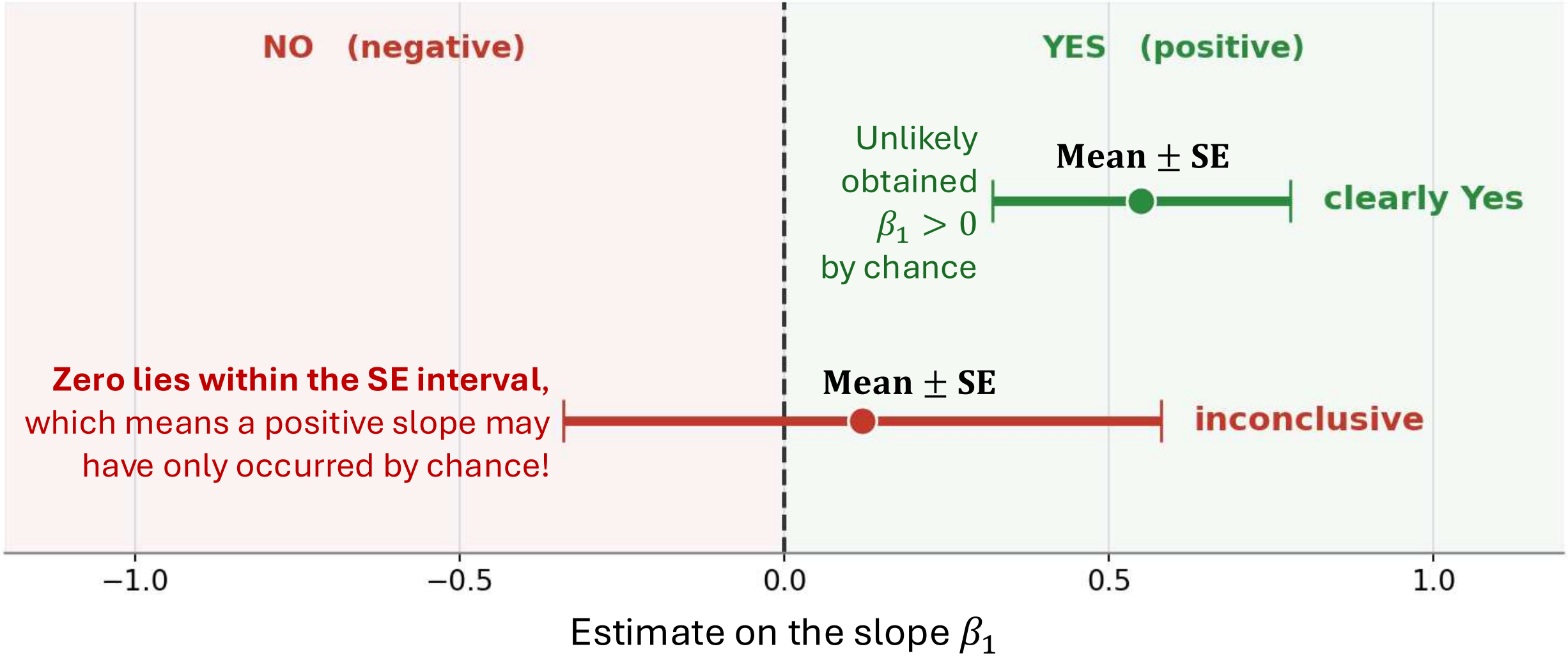
If we only sampled people who studied 4-5 hours, it doesn't matter how many people we ask...
We will not be able to answer this question reliably!

Questions?

Hypothesis Testing

Basic idea of hypothesis testing

Question: Does studying more lead to better exam scores?



Now a bit more formally

Does the predictor i affect the response?

Equivalent

Is the coefficient β_i sufficiently non-zero?

Example

Does studying more lead to better exam scores?

Example

Is the slope of $\beta_{\text{hrs studied}}$ clearly positive?

Let's start from the skeptical assumption that it *does not*, and then **assess whether β_i is too big to be a fluke**

t-statistic: Distance from null hypothesis in SE units

Null hypothesis:
$\beta_i = 0$

$$t = \hat{\beta}_i / \mathrm{SE}(\hat{\beta}_i)$$ *

If our estimate is many $\mathrm{SEs} > 0$, we can be confident we did not get it by chance



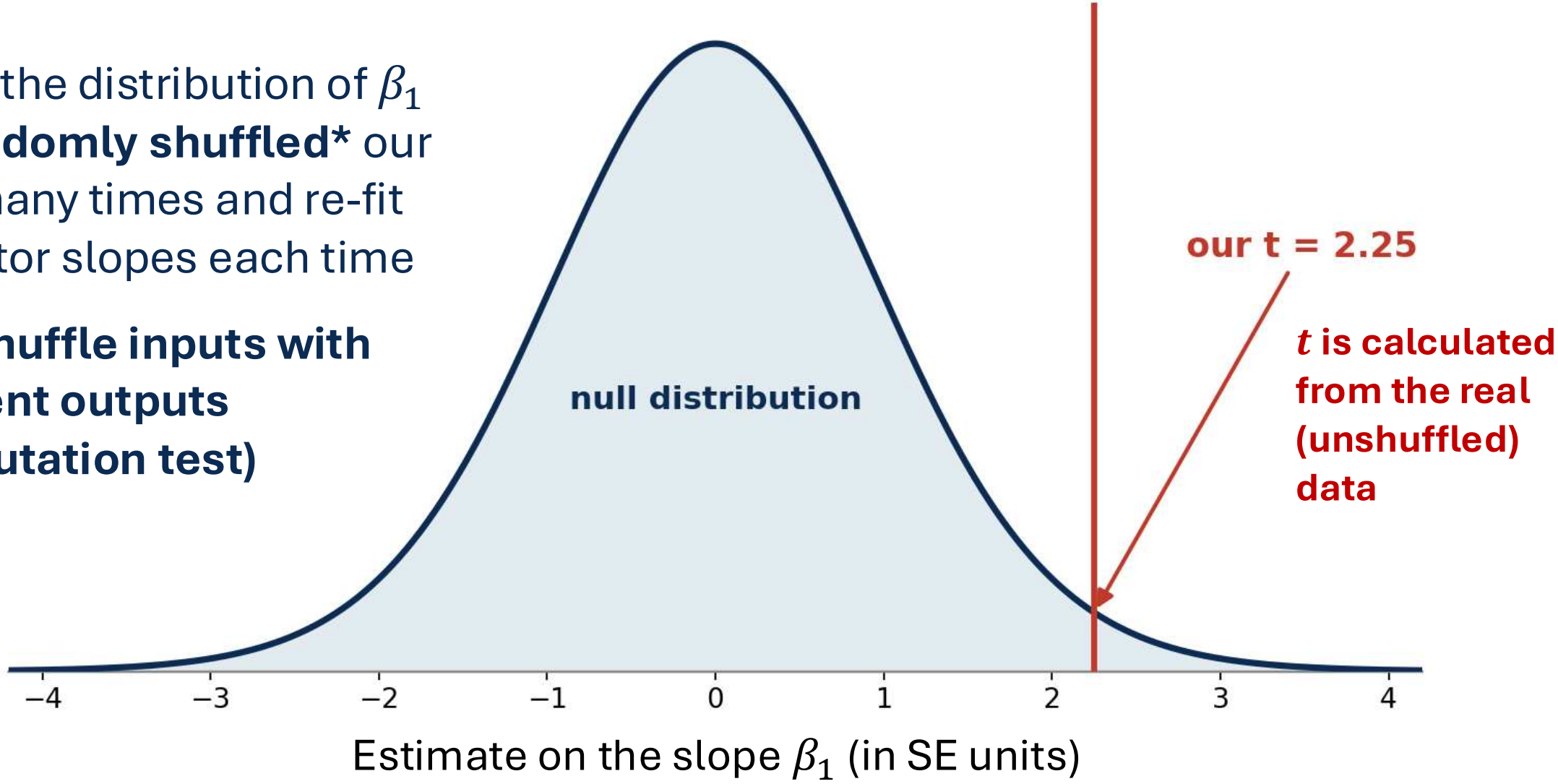
t-statistic quantifies this.
Larger $t \rightarrow$ more confident our predictor has a *real* effect

*assumes $\beta_i = 0$ is null hyp.

In terms of the null distribution

This is the distribution of β_1 we **randomly shuffled*** our data many times and re-fit predictor slopes each time

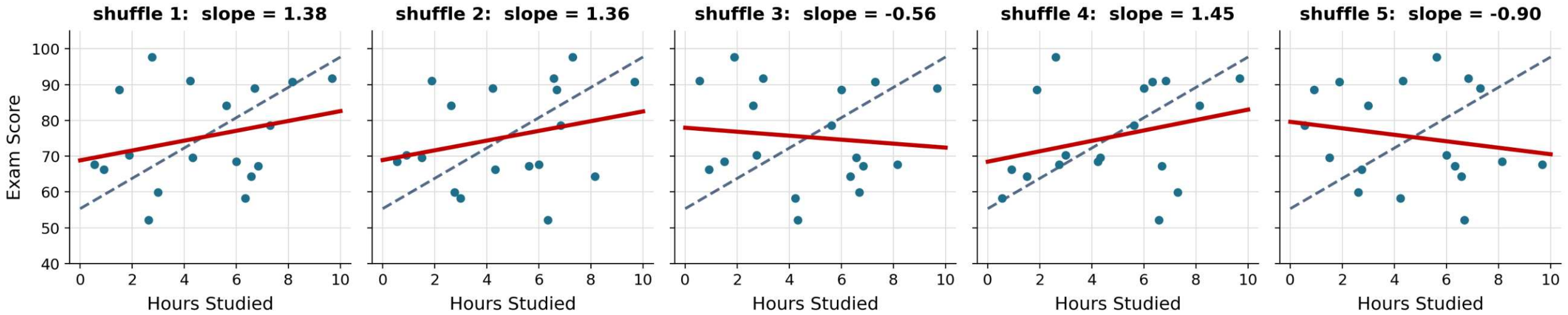
***We shuffle inputs with different outputs (permutation test)**



Permutation tests generate the null distribution

This is the distribution of β_1 we **randomly shuffled*** our data many times and re-fit predictor slopes each time

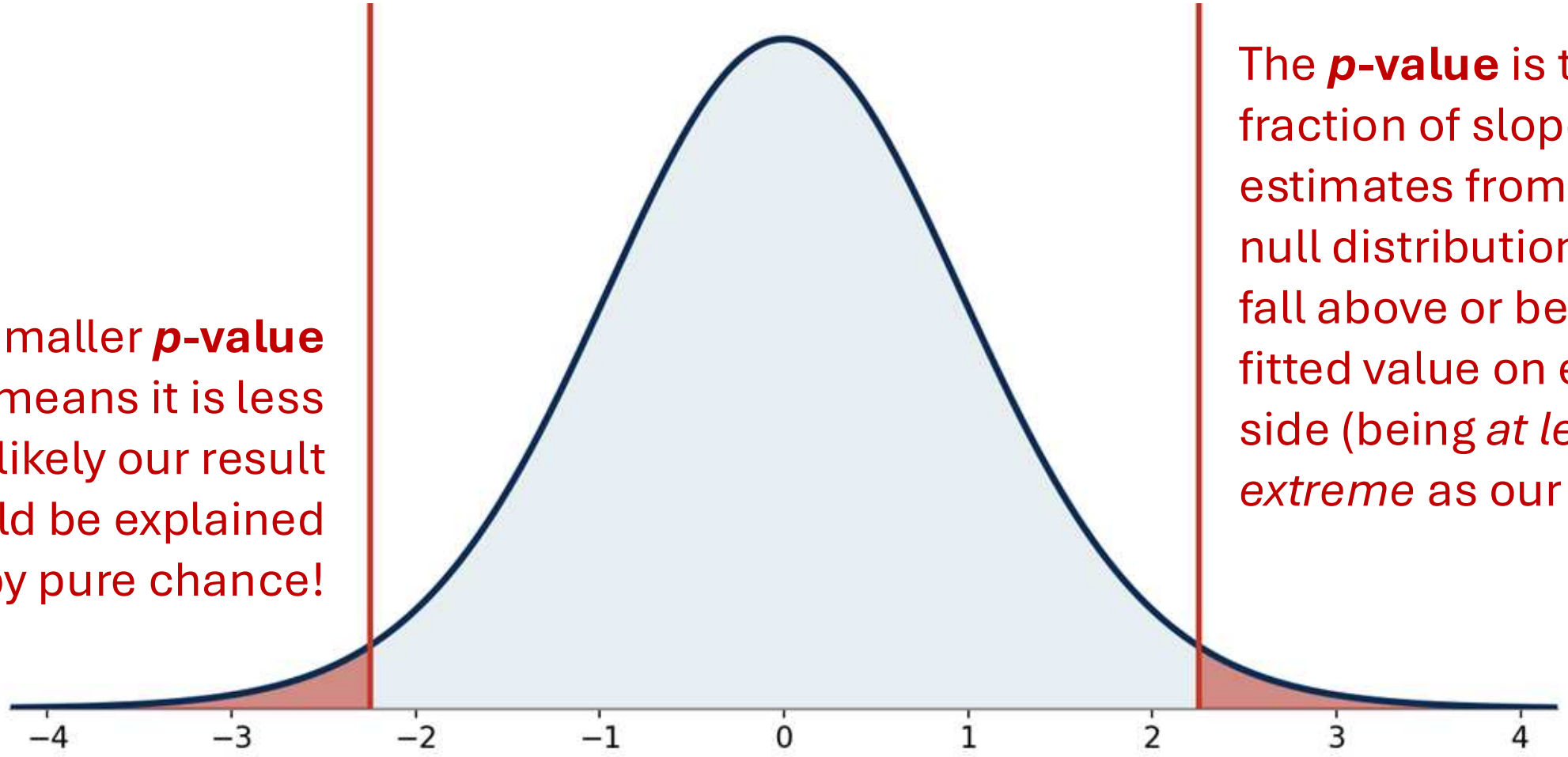
Just by chance, our slope will often deviate quite a bit from zero in the permutations – this will happen more often when our dataset is small – but it does *not* mean the slope is significant



How to get the *p*-value

***p*-value = area in the tails beyond our fitted *p*-value**

A smaller ***p*-value** means it is less likely our result could be explained by pure chance!



The ***p*-value** is the fraction of slope estimates from the null distribution that fall above or below our fitted value on either side (being *at least as extreme* as our result)

Estimate on the slope β_1 (in SE units)

How to interpret the *p*-value

A smaller ***p*-value** means it is less likely our result could be explained by pure chance!

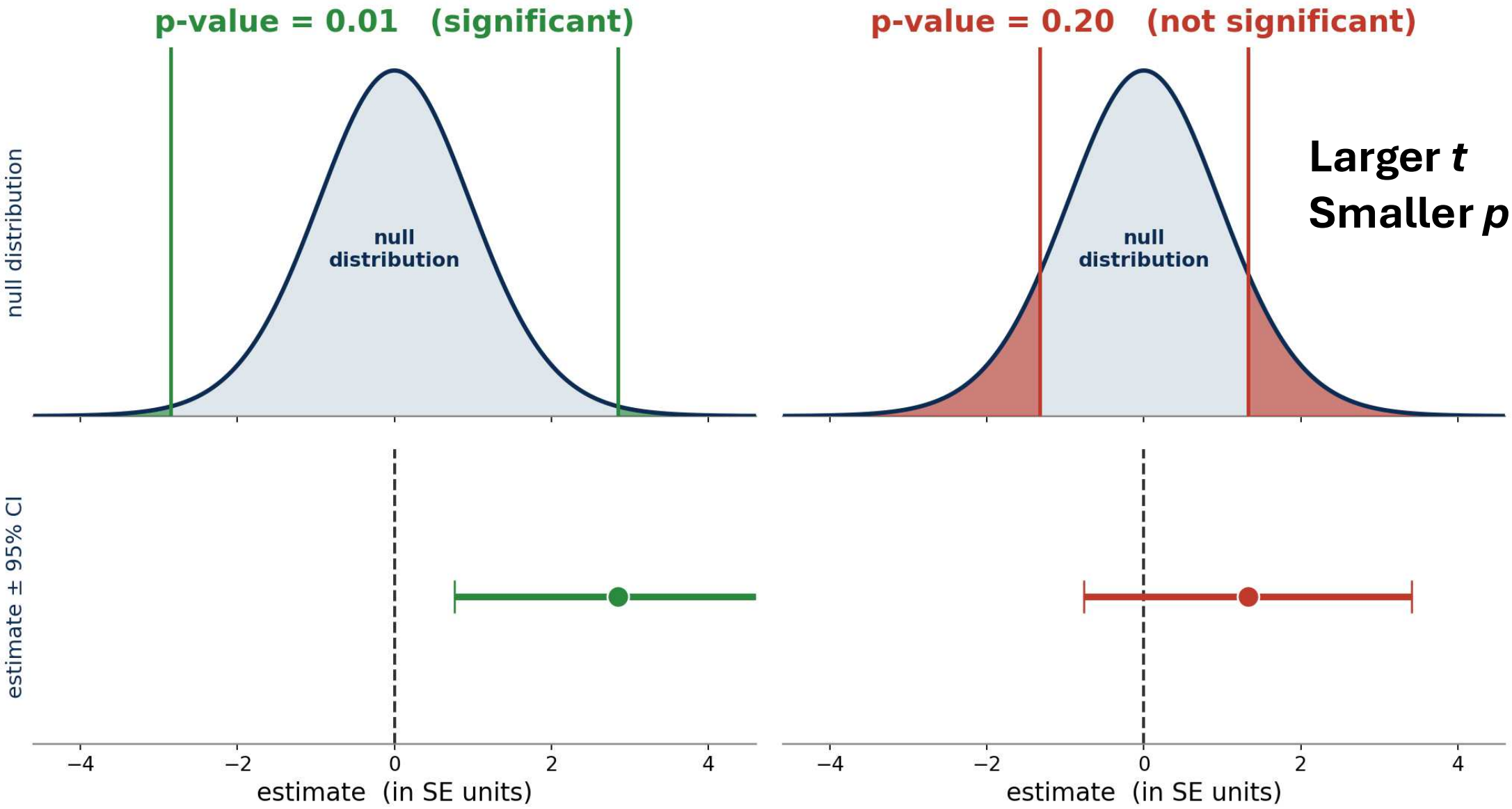
Is my result significant?
In other words, can I reliably confirm that my hypothesis (e.g., studying improves exam scores) is true?

1) **Choose a significance level**
   - $\alpha = 0.05$ is a common choice
   - This corresponds to about $|t| > 2$

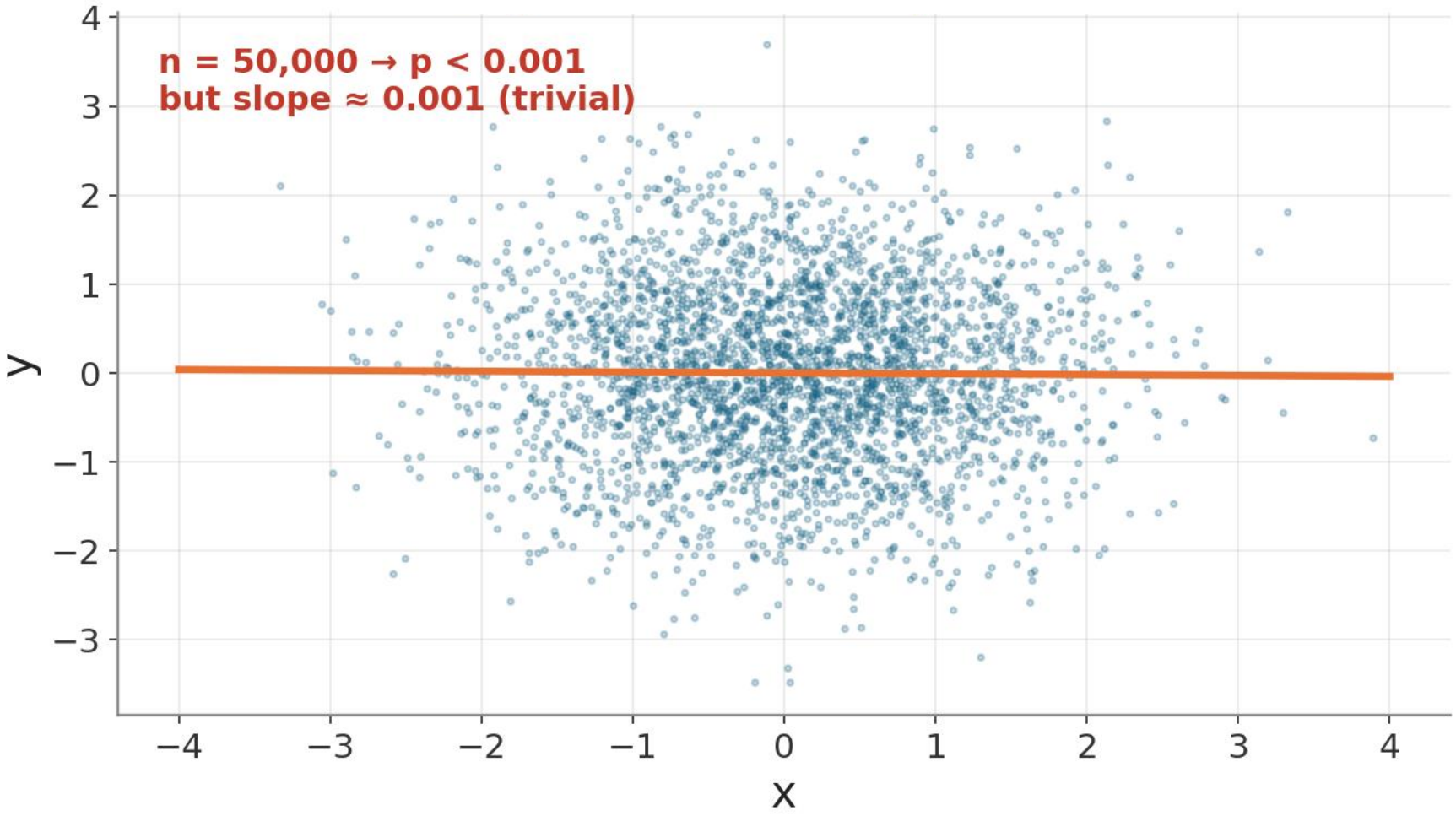
2) **If $p > \alpha \rightarrow$ No, insufficient evidence**
   - This does not mean we can for sure say there is no correlation, just that there is not enough evidence

3) **If $p \leq \alpha \rightarrow$ Yes, sufficient evidence**
   - Not 100% sure to be true…Just likely

The *p*-value and *t*-statistic give similar info



Some caveats: Significant ≠ Important

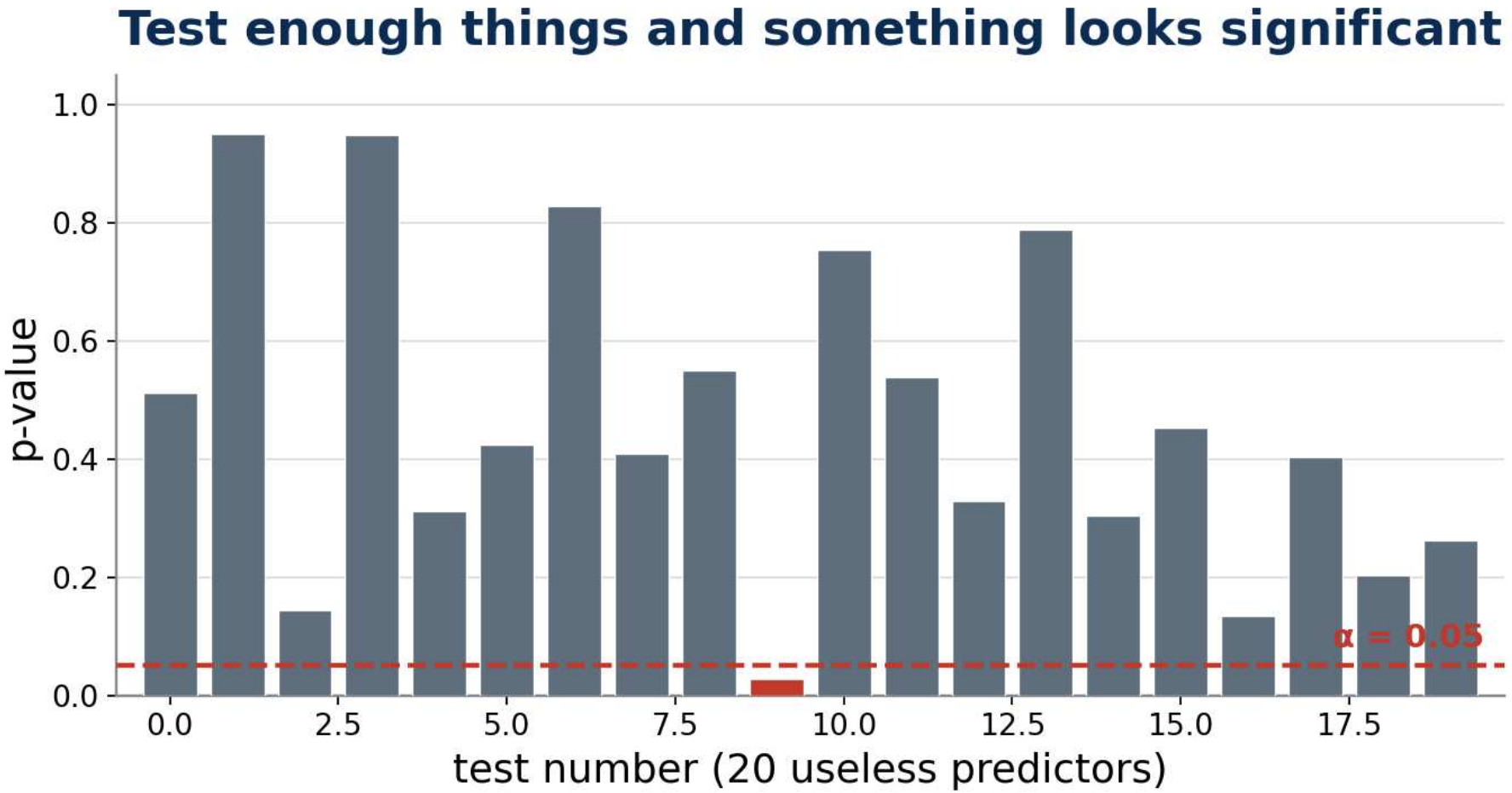


Notice the *p*-value in this case is small, well below our 0.05 threshold, but the slope is near zero…

So, the associated predictor like doesn't have much real effect

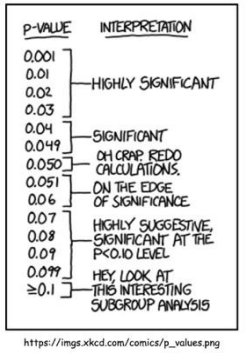
With huge n, a trivially small effect can still be "significant"

Some caveats: Testing “too many” predictors



Avoid “p-hacking”

Testing lots of predictors and just reporting the ones that look significant





Run 20 tests at α = 0.05 and ~1 looks “significant” by pure chance

Questions?

What to know for HW + Exams

- How to use **cross-validation** for hyperparameter selection (don't worry about the variants like nested CV)
- The effects of increasing the **number of folds (k)**
- The one-standard-error rule (used to keep models simple)
- The basics of bootstrapping to estimate variance
- How to **test hypotheses** using the *t*-statistic and *p*-values