NYU FRE 7773 - Week 3

Machine Learning in Financial Engineering

Ethan Rosenthal

Feature Engineering & Model Selection

Machine Learning in Financial Engineering

Ethan Rosenthal

Feature Engineering

The ML Recipe

1. Think up some model
2. Feed **data** into the model and make predictions.
3. Calculate the loss between predictions and true values.
4. Determine the model parameters that produce the minimum loss.

The ML Recipe

1. Think up some model
2. Feed **data** into the model and make predictions.
    a. We decide what data to include.
    b. We decide *how* to turn data into features.
3. Calculate the loss between predictions and true values.
4. Determine the model parameters that produce the minimum loss.

Feature Engineering:

Turning *data* into *features*

Nonlinear Features for Linear Models

- We can do whatever we want to the features **X**.

$$y_i = \sum_{j=0}^{p} \beta_j X_{ij}$$

- We can square a feature, we can multiple features by each other, we can apply a sine, etc...

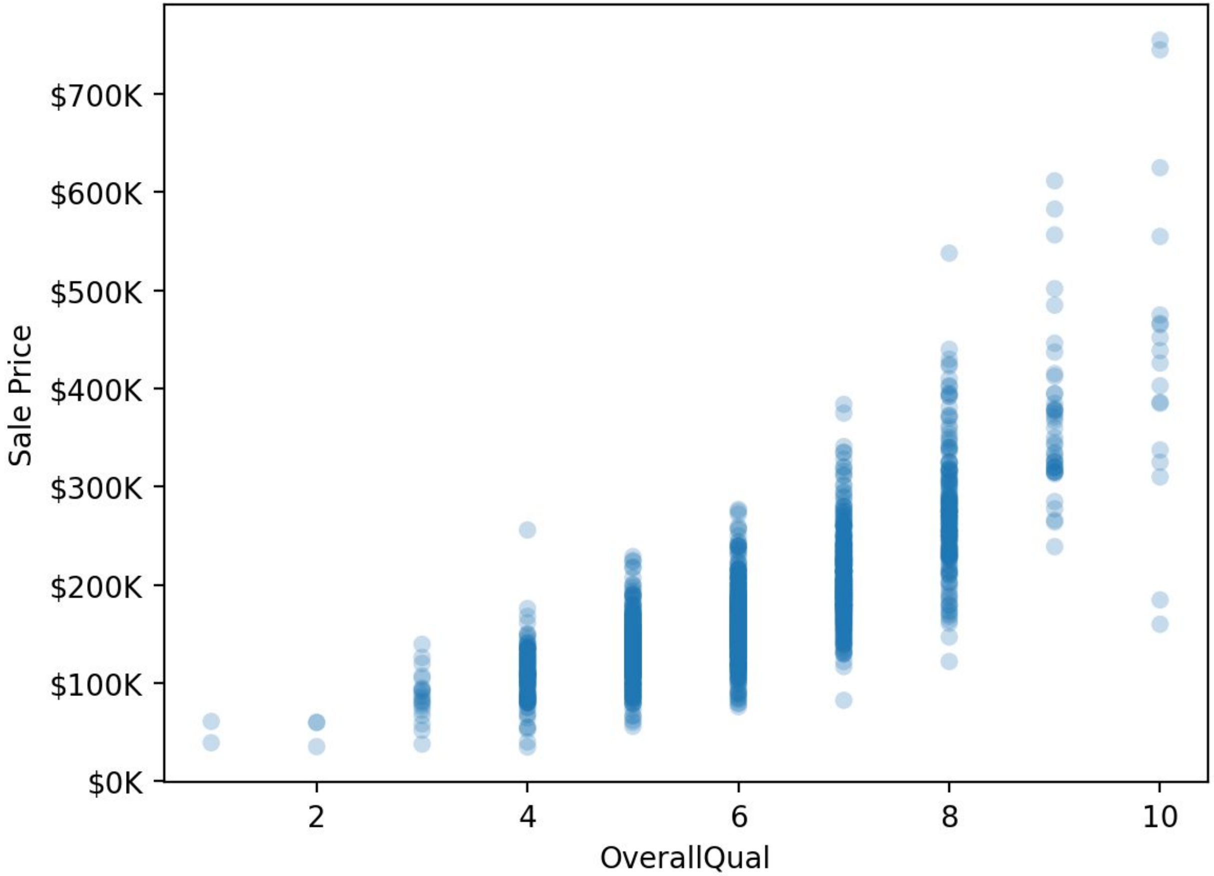
$$y_i = \beta_0 + \beta_1 X_{i1} + \beta_2 X_{i1}^2 + \beta_3 X_{i1} X_{i2} + \beta_4 sin\left(X_{i3}\right)$$

- While these features are nonlinear, the model is linear in the parameters β.

Why Engineer Features?

We know a better

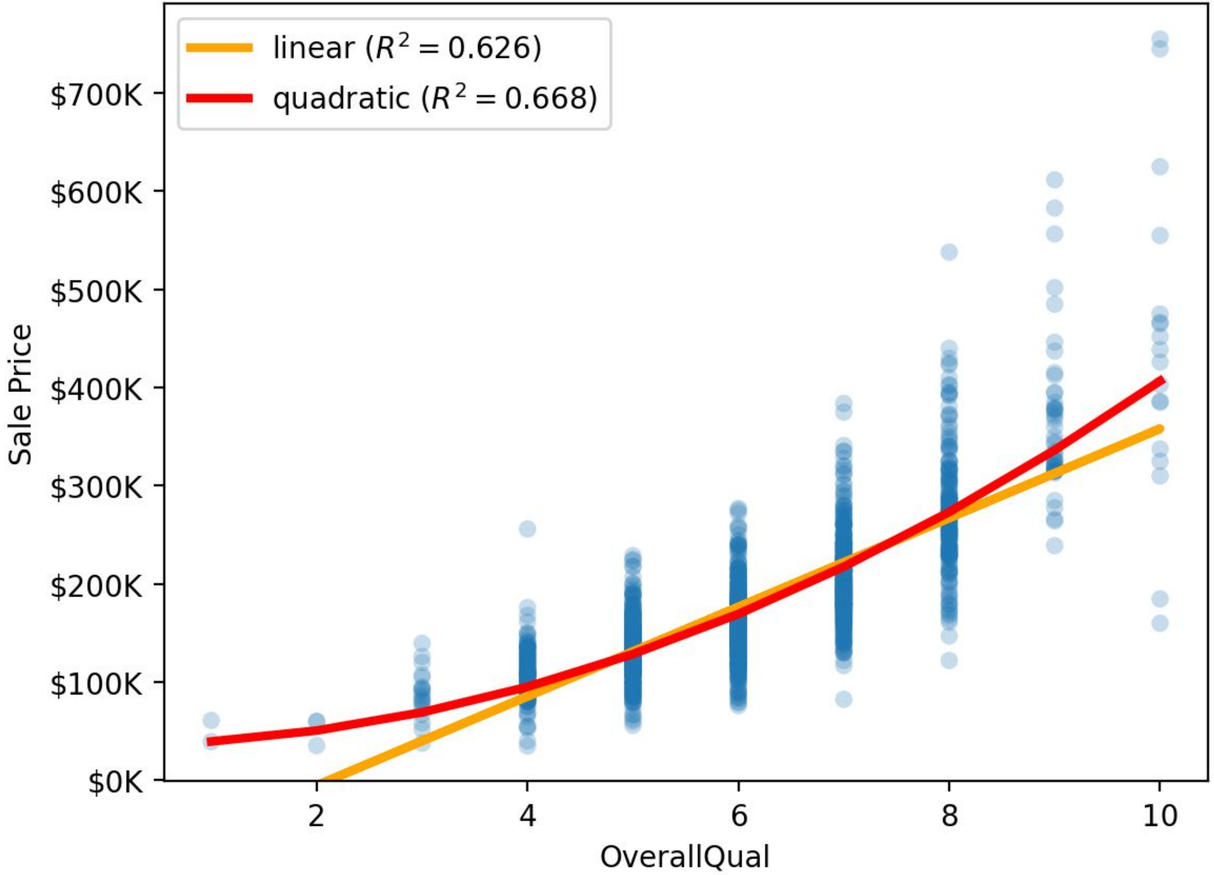
data <> target relationship



Why Engineer Features?

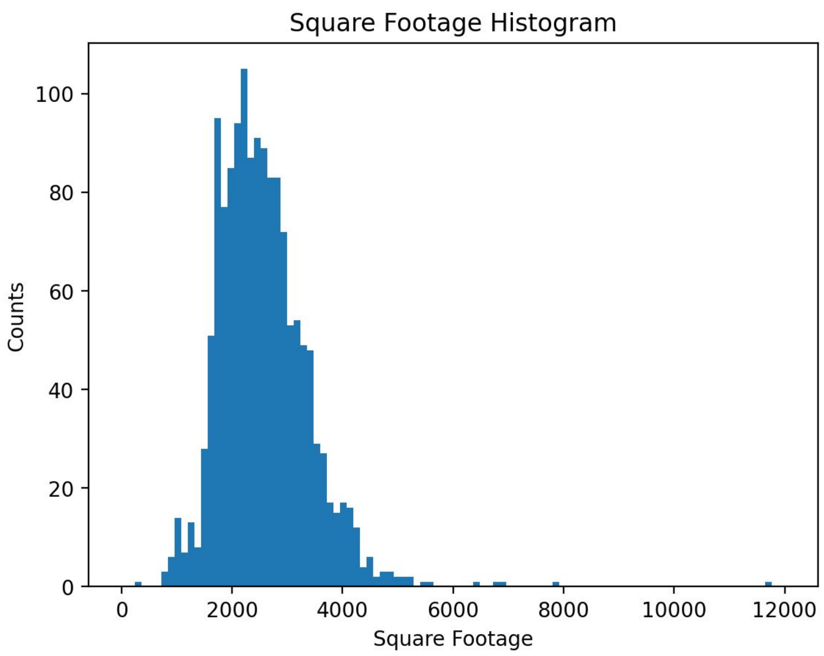
We know a better

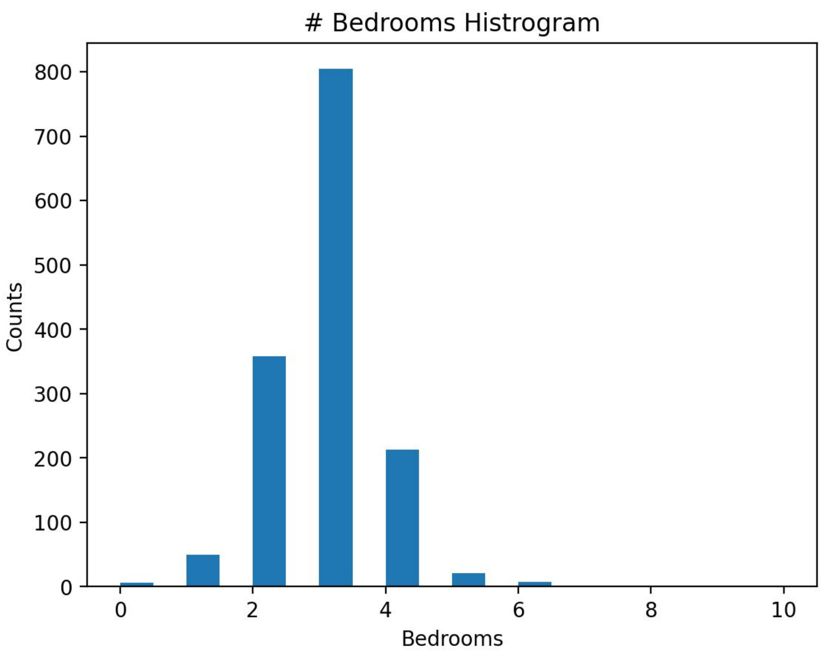
data <> target relationship



Why Engineer Features?

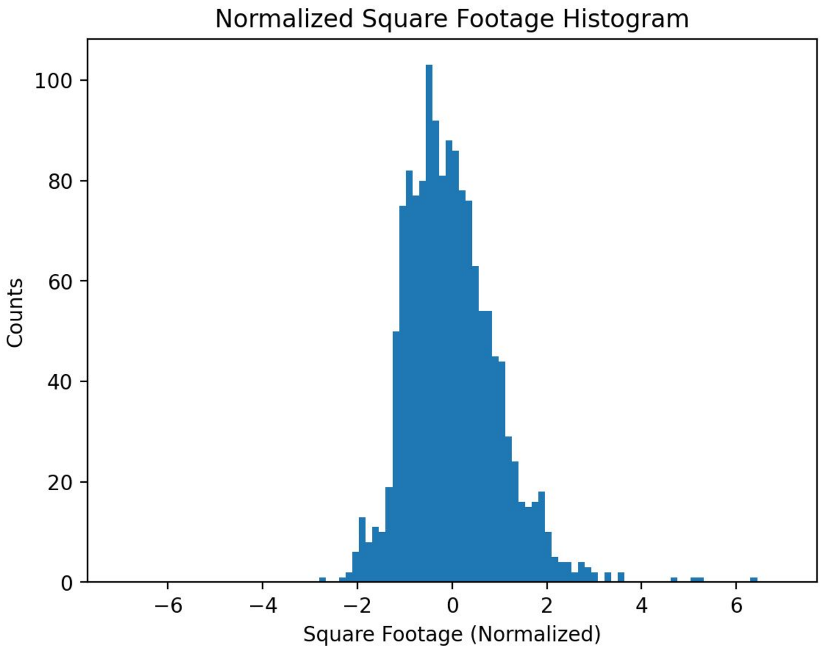
Put features on the same scale

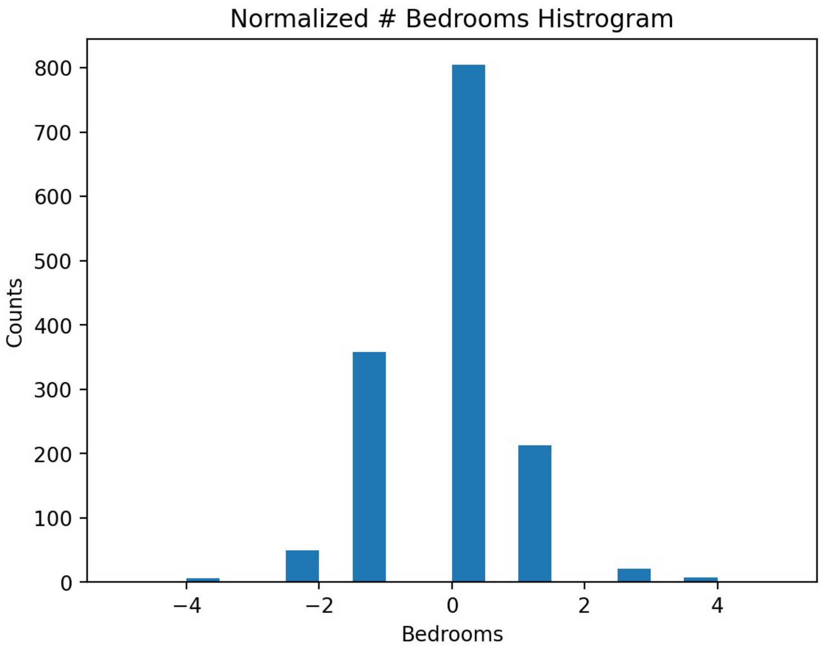




Why Engineer Features?

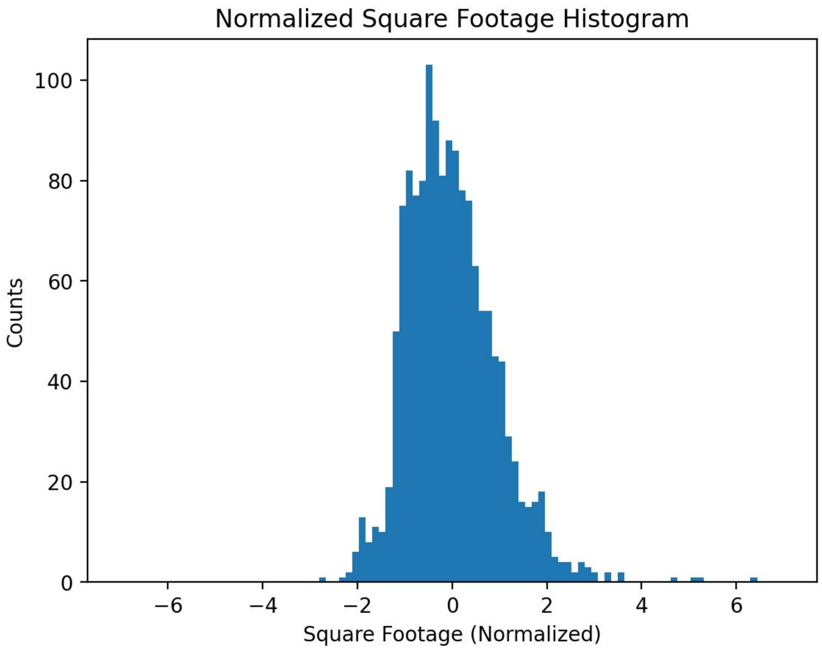
Put features on the same scale





Why Engineer Features?

Put features on the same scale



Normalization: $$\vec{\mathbf{X}}_j^* = \frac{\vec{\mathbf{X}}_j - \bar{X}_j}{VAR\left(\vec{\mathbf{X}}_j\right)}$$

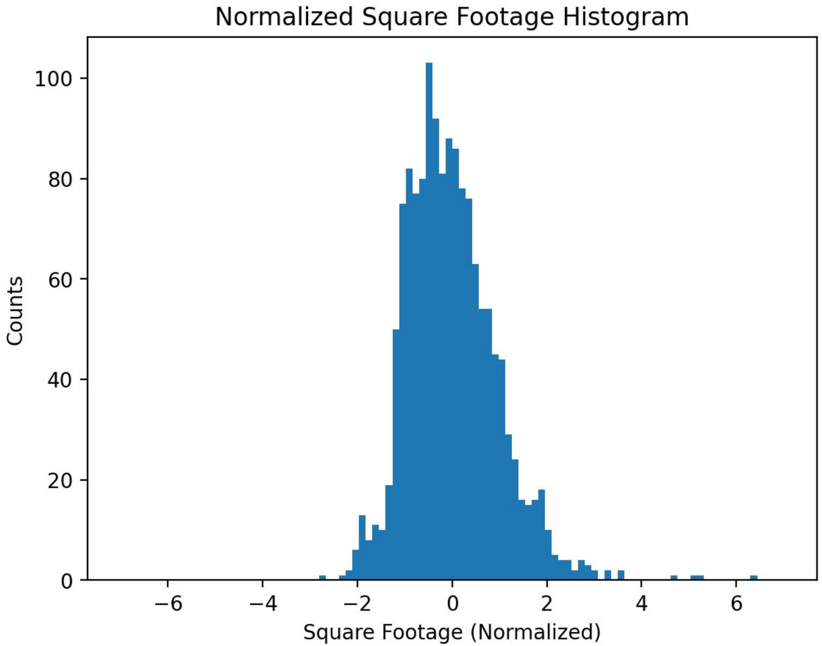
where

$$\bar{X}_j = \frac{1}{n}\sum_{i=1}^{n} X_{ij}$$

$$VAR\left(X_j\right) = \frac{1}{n-1}\sum_{i=1}^{n}\left(X_{ij} - \bar{X}_j\right)^2$$

Why Engineer Features?

Put features on the same scale

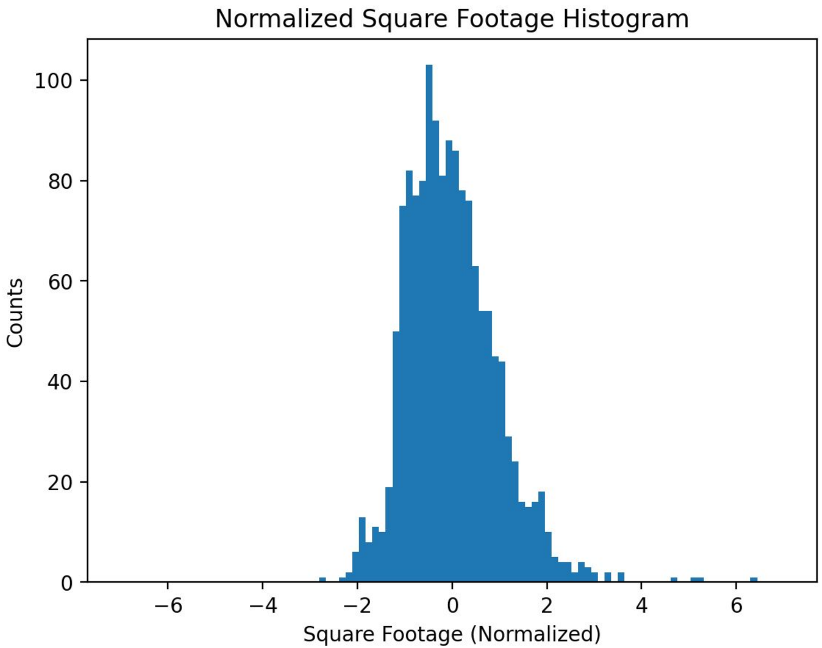


Many other feature scaling techniques:

- Log transform
- Min/Max scaler
- Max Abs Scaler
- Power transform

Why Engineer Features?

Put features on the same scale

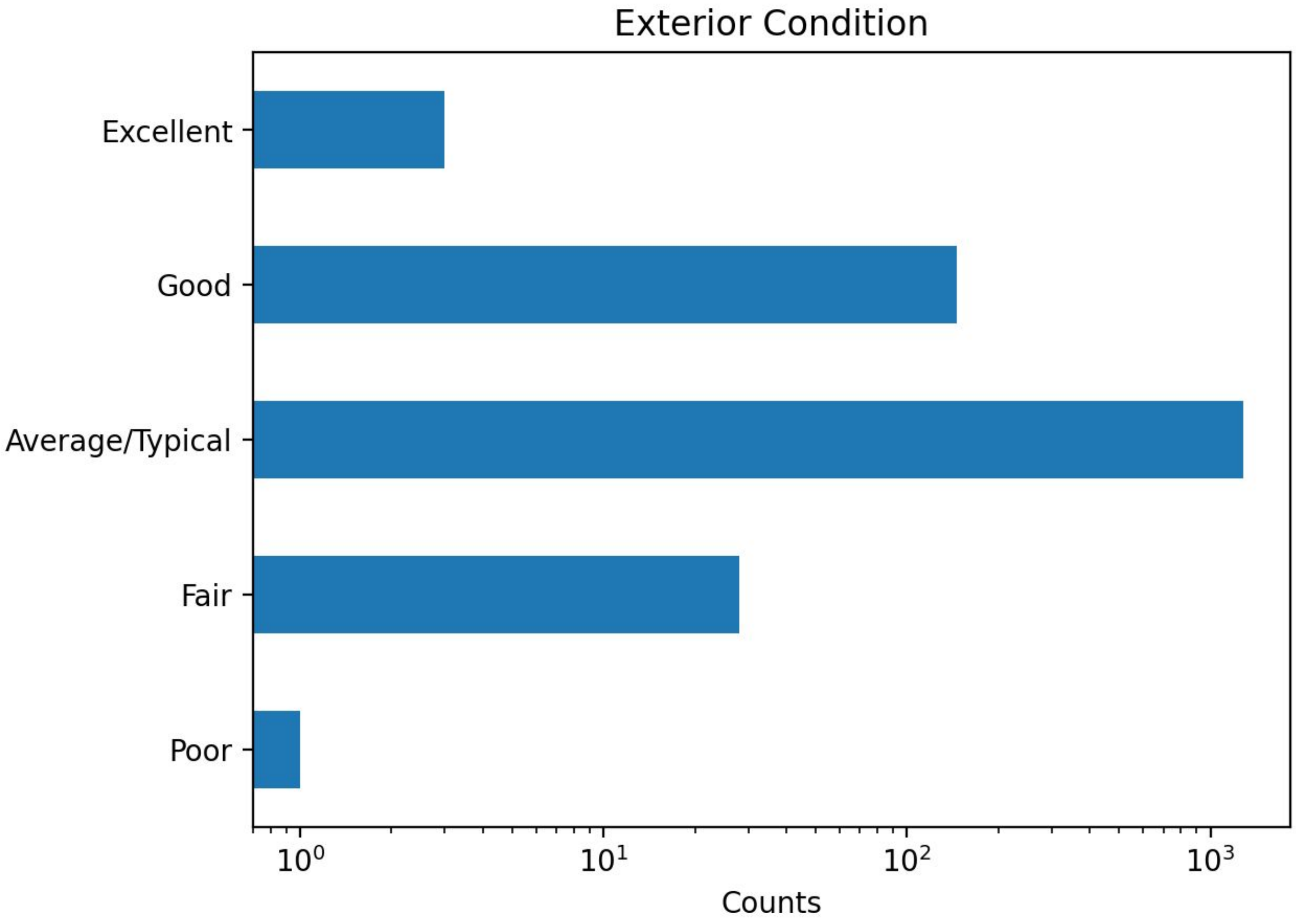


Why put features on the same scale?

- Can draw inferences from linear models.
- Some algorithms converge faster.
- Some algorithms *only* converge if features are scaled.

Why Engineer Features?

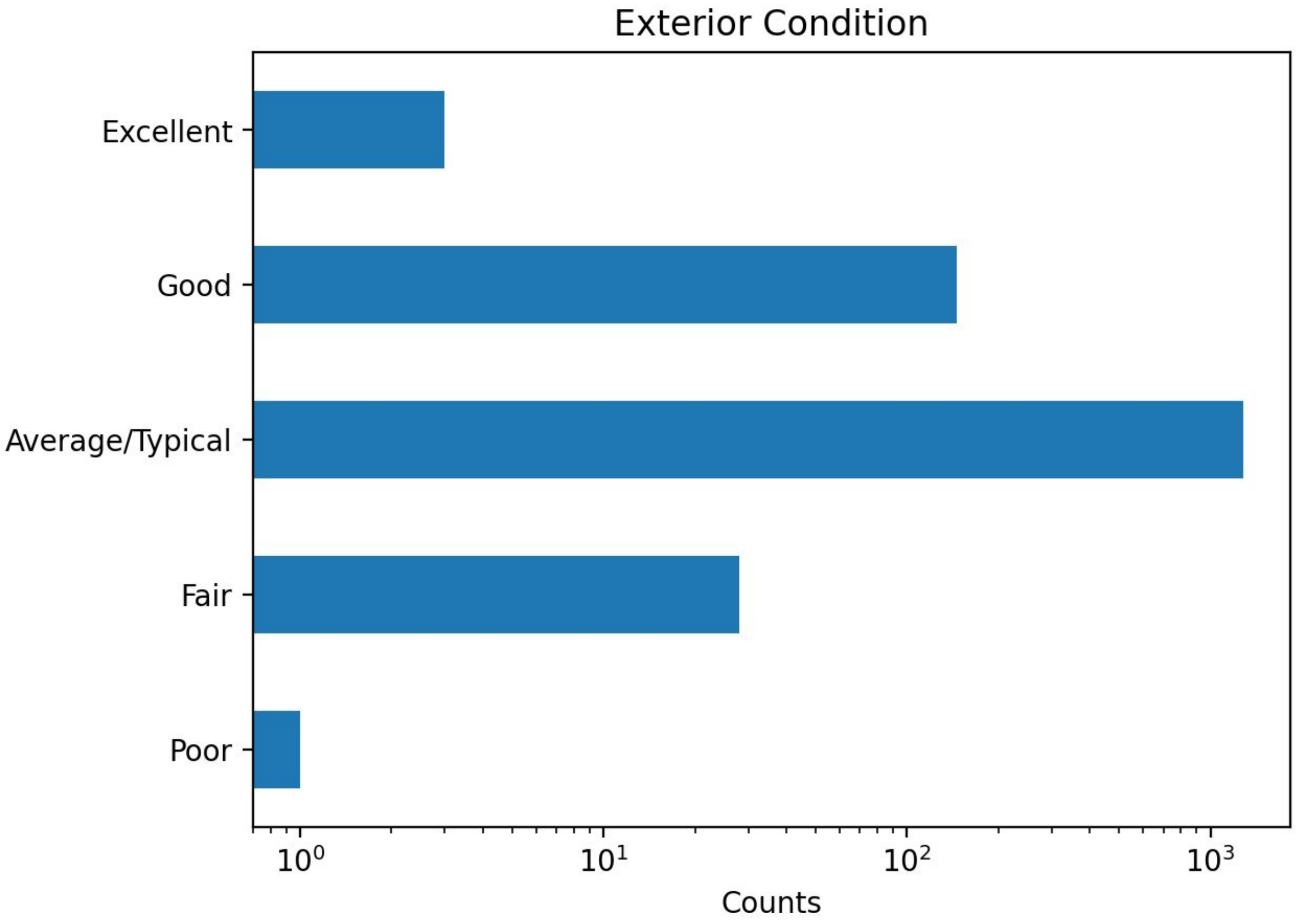
Sometimes we have to:



All Features Must Be Numbers

Why Engineer Features?

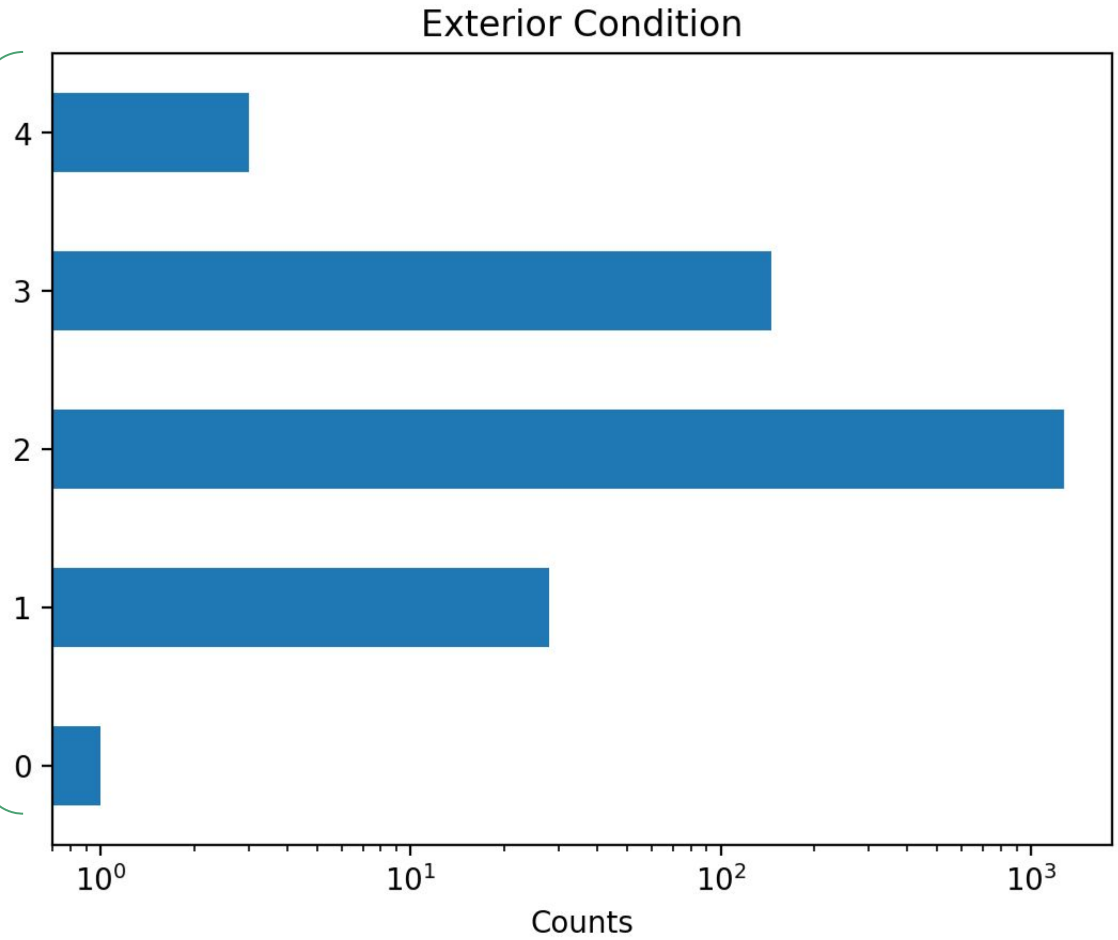
Sometimes we have to:



Why Engineer Features?

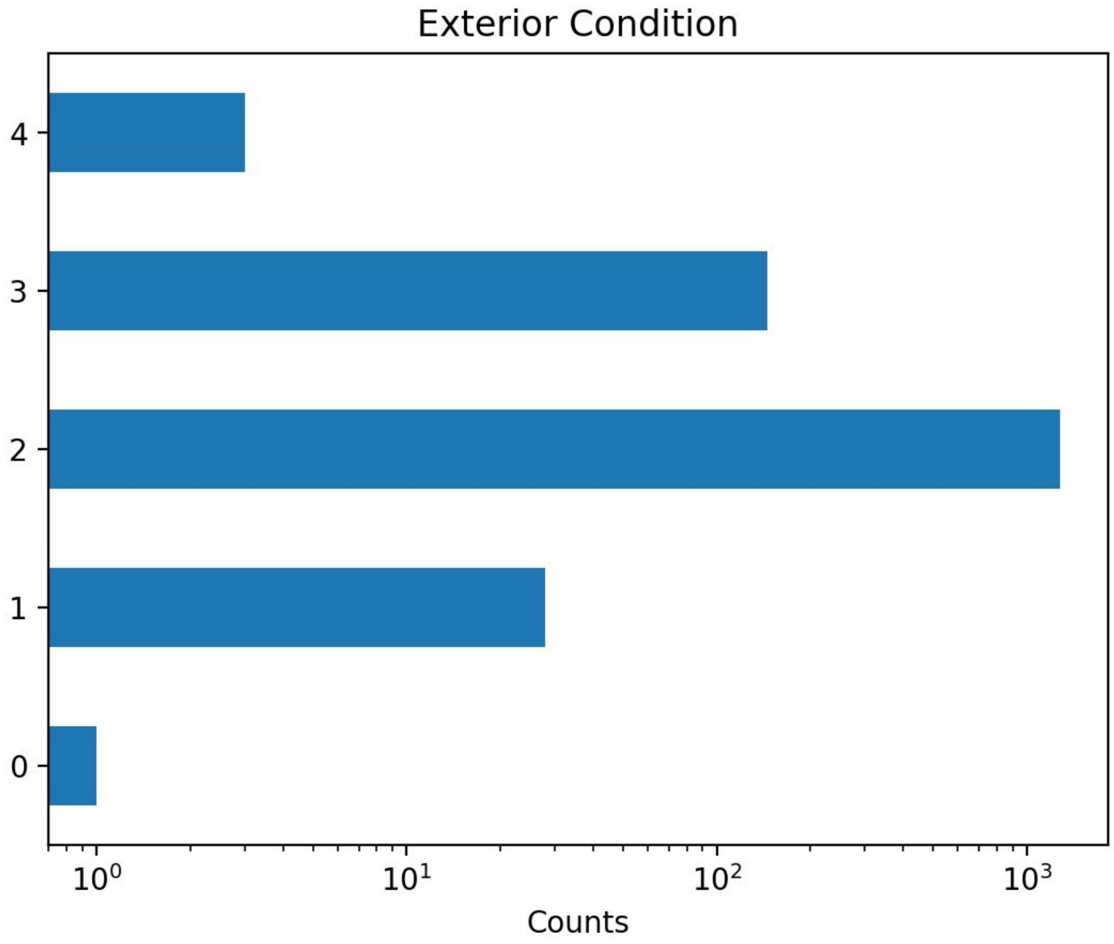
Sometimes we have to:

Numbers!



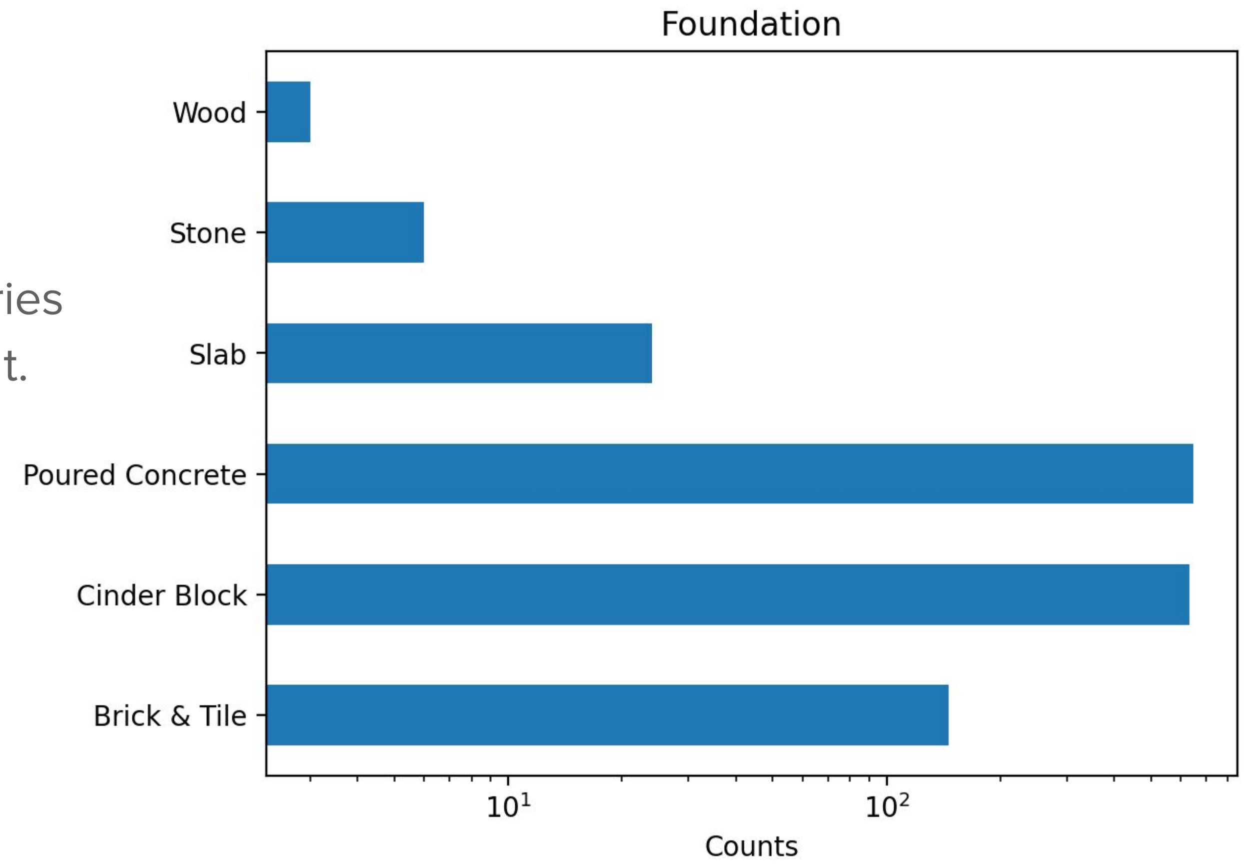
Ordinal Encoding

- If categories have different “distances” from each other, then do ordinal encoding.
- e.g.
  - too small, fits well, too big
  - Strongly disagree, disagree, neurtral, agree, strongly aggree



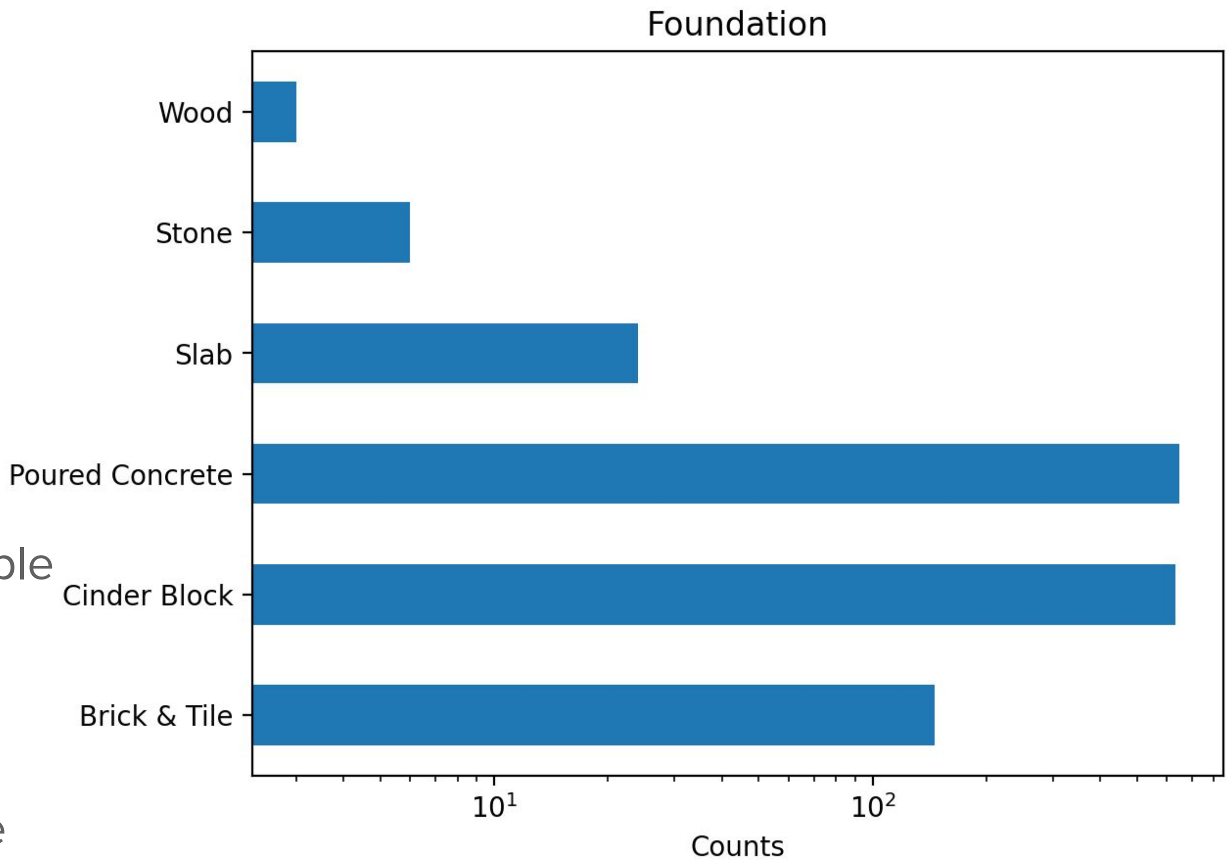
Non-ordinal Categories

- Sometimes the categories are largely independent.



One Hot Encoding

- Imagine C categories.
- Break up into C-1 binary features
- Each feature indicates whether or not the sample belongs to a specific category.
- If all features are False, then this implies sample belongs to Cth category.



One Hot Encoding

	is_wood	is_stone	is_slab	is_poured	is_cinder
“Wood”	1	0	0	0	0
“Slab”	0	0	1	0	0
“Brick & Tile”	0	0	0	0	0

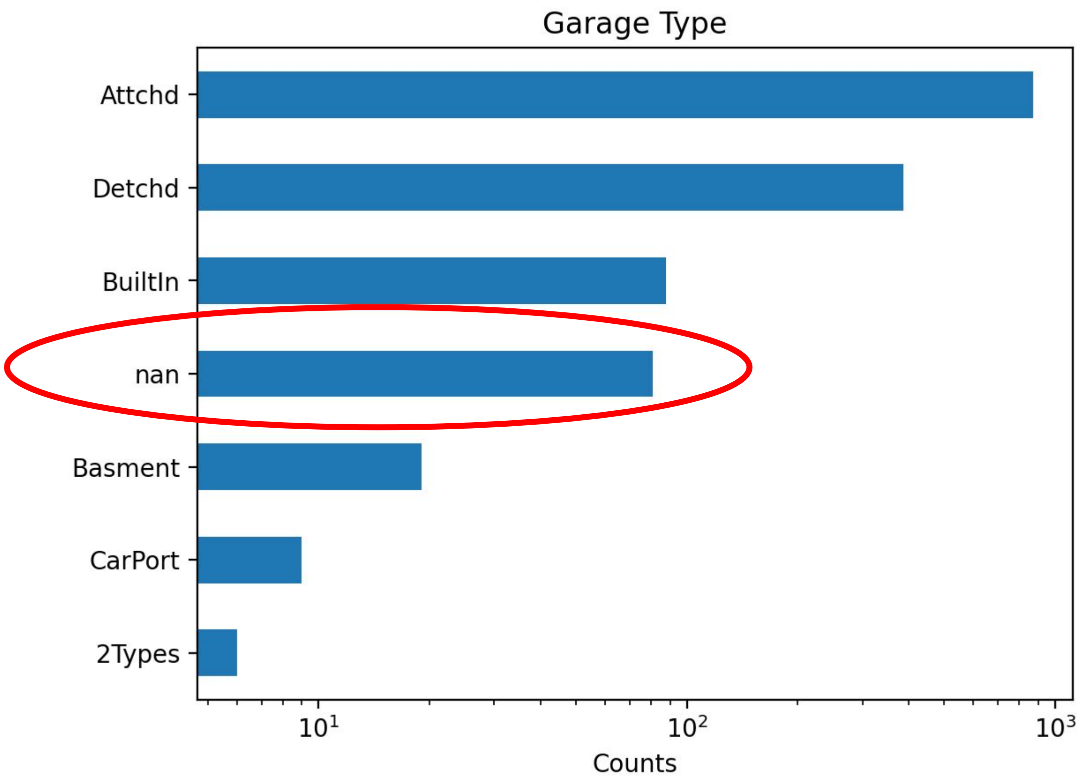
One Hot Encoding Caveats

- Blows up the size of your dataset: n X 1 -> n x C
- Need to combine with more advanced feature engineering for large C.
- Large C examples:
  - Recommender systems: one-hot-encoding every single user and item.
  - Text: one-hot-encoding every unique word or sub-word.
- Large C solutions:
  - Embeddings: embed all categories in a low-dimensional vector space.
    - Each category gets mapped to a O(100)-dimensional vector.
    - Vectors are model parameters. Learn model parameters that embed semantically similar categories near each other in this vector space.
  - Hashing: categories get randomly hashed to binary features.
    - Hash to fewer features than the number of categories.
    - Trade off reduced accuracy (due to hash collisions) for smaller feature space.

Feature Engineering - beyond scaling and encoding

- Streams -> Features
  - “Average order value in the last month”
  - “Standard deviation of the time between keystrokes”
  - “Total minutes watched by this customer for this video category in the last week”
- Images
  - Models -> models
    - Image -> “Pedestrian in front of car” -> “should brake”
  - Deep learning
    - Feed in raw pixels and learn features
- Text
  - Annotation
    - Part of speech tagging
    - Sentiment analysis
    - Named Entity Recognition

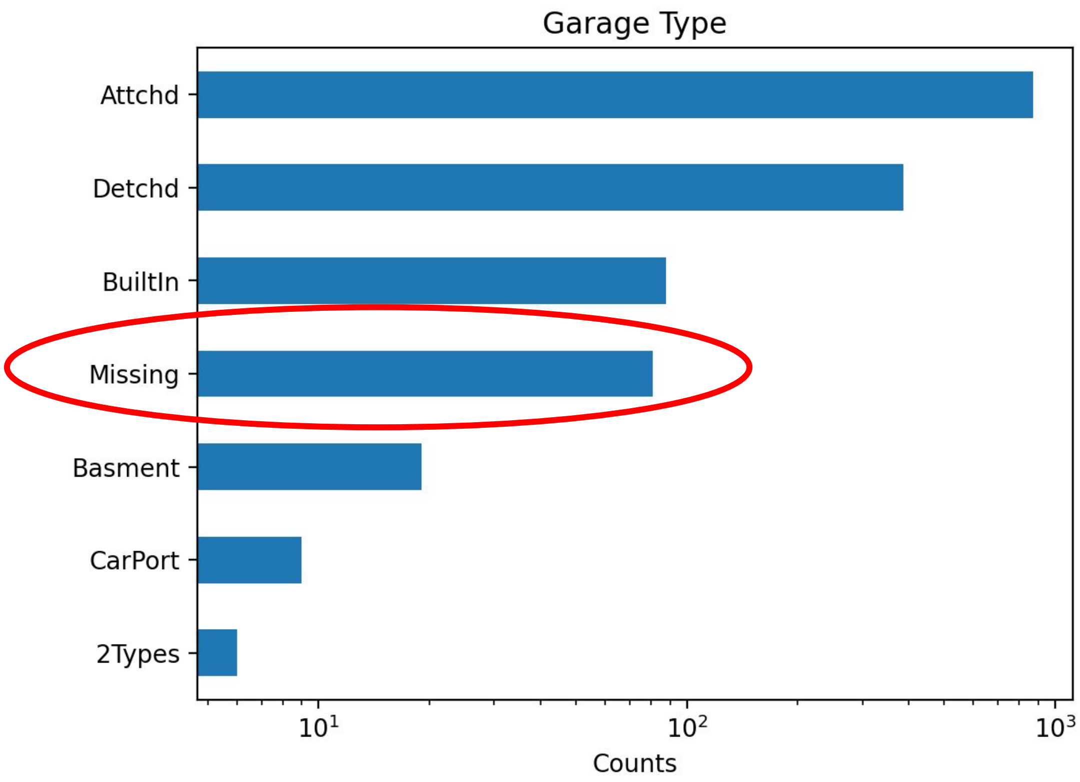
Feature Engineering - Missing Category



Feature Engineering - Missing Category

Simplest solution:

- Create a new "missing" category and then encode.



Feature Engineering - Missing Numerical Data

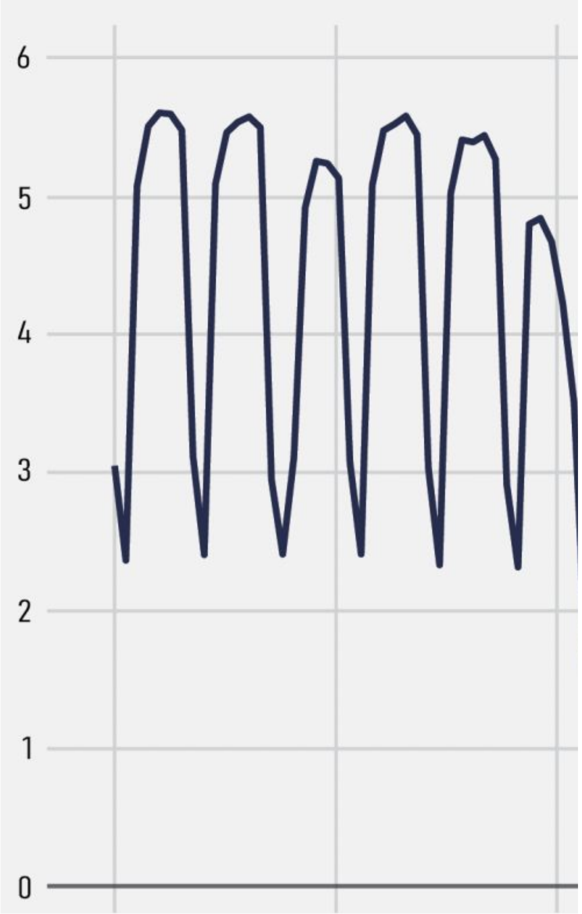
- Handling missing data is generally called “imputation”.
- Simple solutions:
  - Fill in missing values with the average value
  - Fill in missing values with 0 and create a separate “indicator” binary feature that’s 1/0 when the data is missing/not missing.
- Beware!
  - Data may not be randomly missing.
  - Missing data may be a valuable signal in itself.
  - Do not simply remove missing data from your dataset.

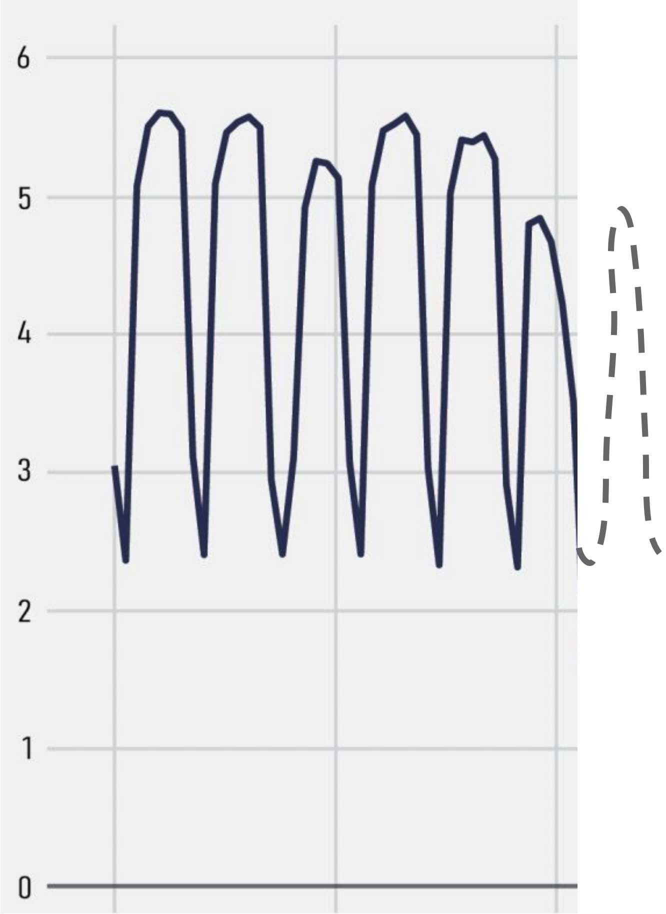
The ML Recipe

1. Think up some model
2. Feed **data** into the model and make predictions.
    a. We decide what data to include.
    b. We decide *how* to turn data into features.
3. Calculate the loss between predictions and true values.
4. Determine the model parameters that produce the minimum loss.

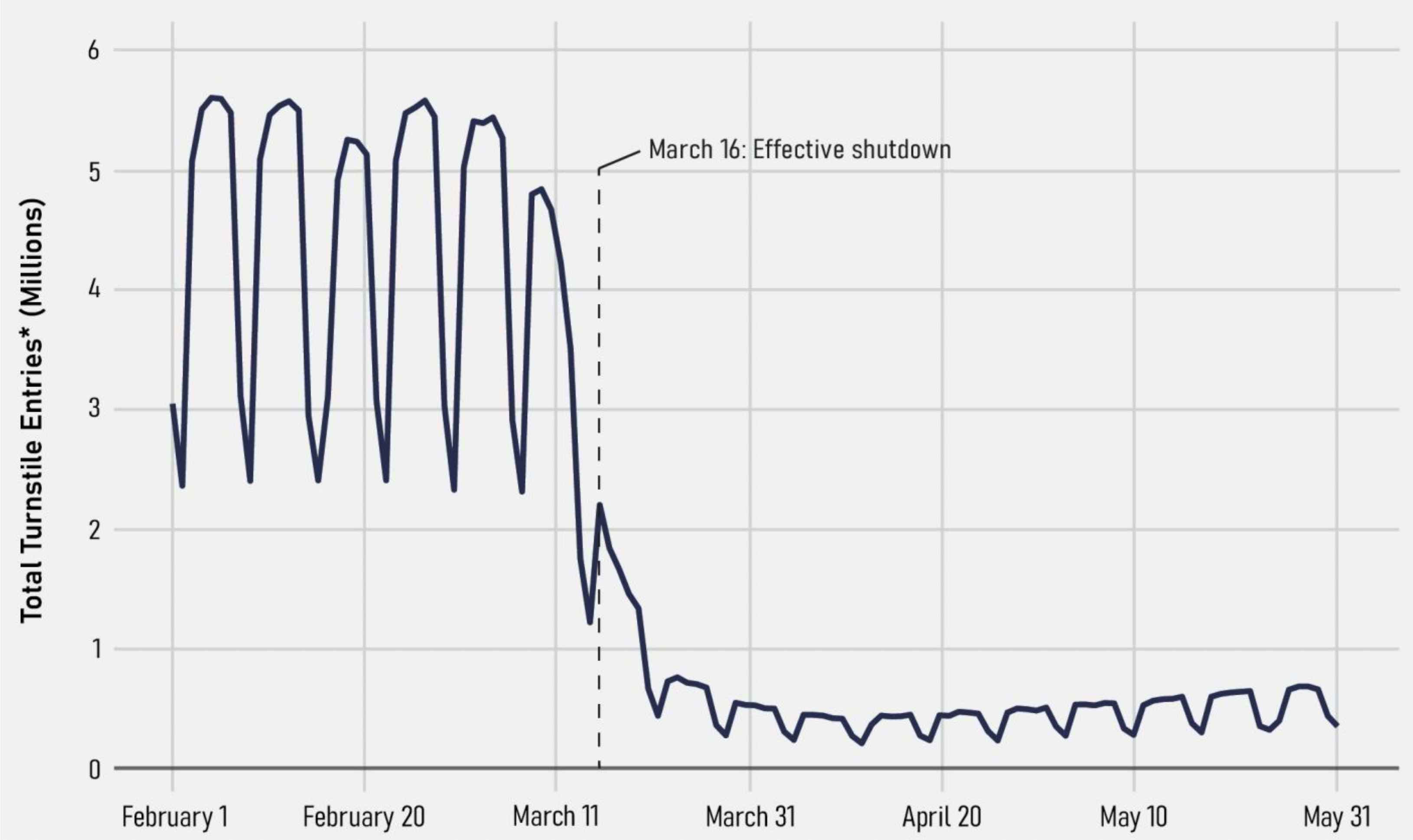
For many use cases, feature engineering is much more important than anything else in this recipe.

Model Selection

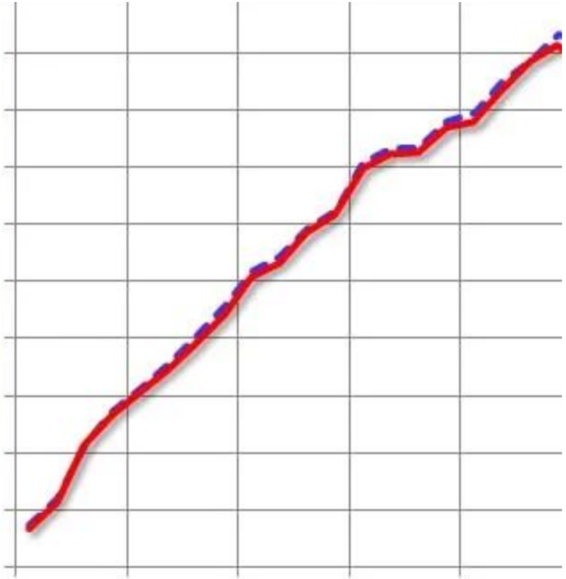

6
5
4
3
2
1
0

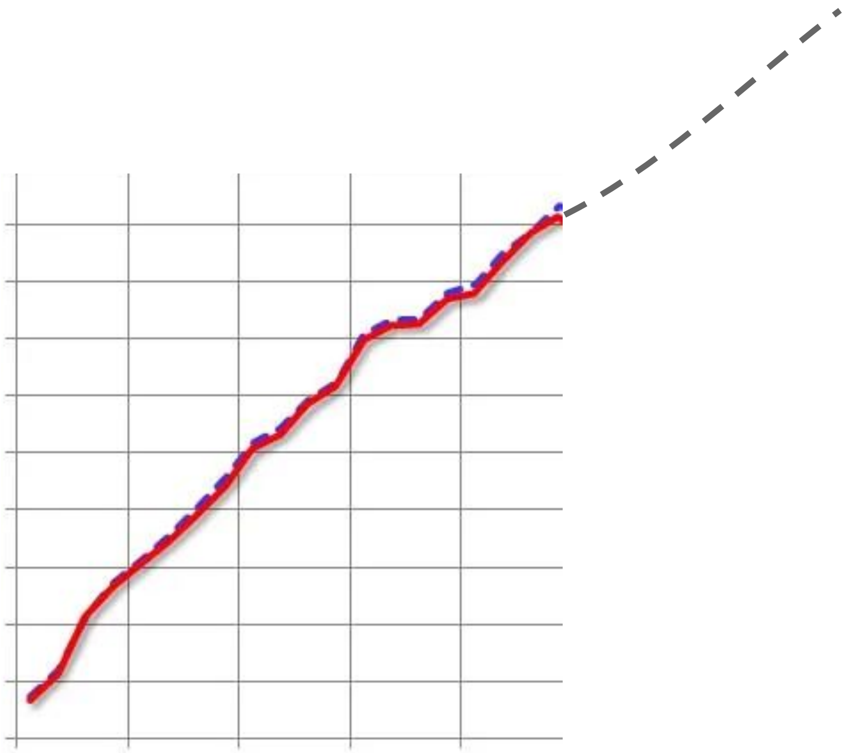

6
5
4
3
2
1
0

MTA Subway Ridership

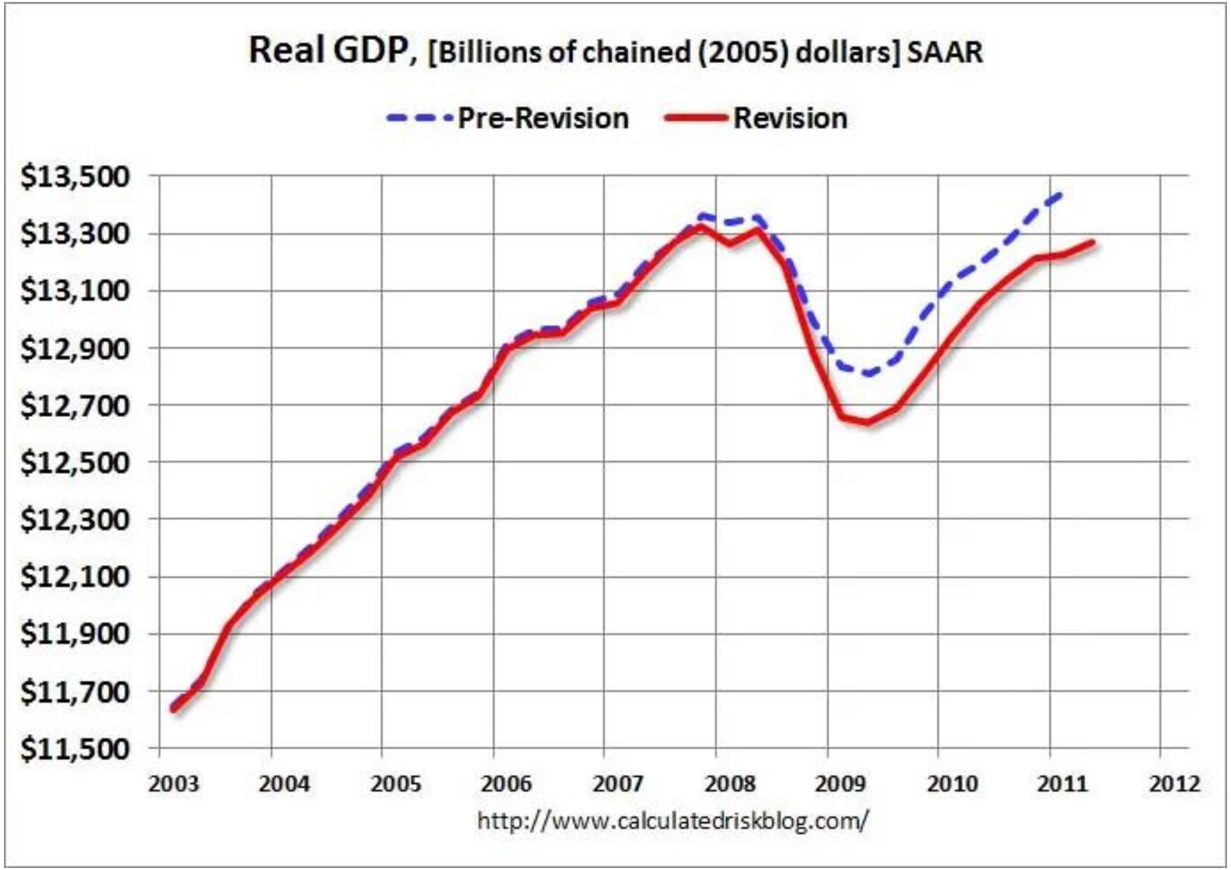


https://wagner.nyu.edu/files/faculty/publications/Full%20Report.pdf





US GDP and the Great Recession



Twitter taught Microsoft's AI chatbot to be a racist asshole in less than a day

68

By James Vincent | Mar 24, 2016, 6:43am EDT

Via The Guardian | *Source TayandYou (Twitter)*
| 68 comments

SHARE

What’s the Theme?

New data is different than the data the model was trained on

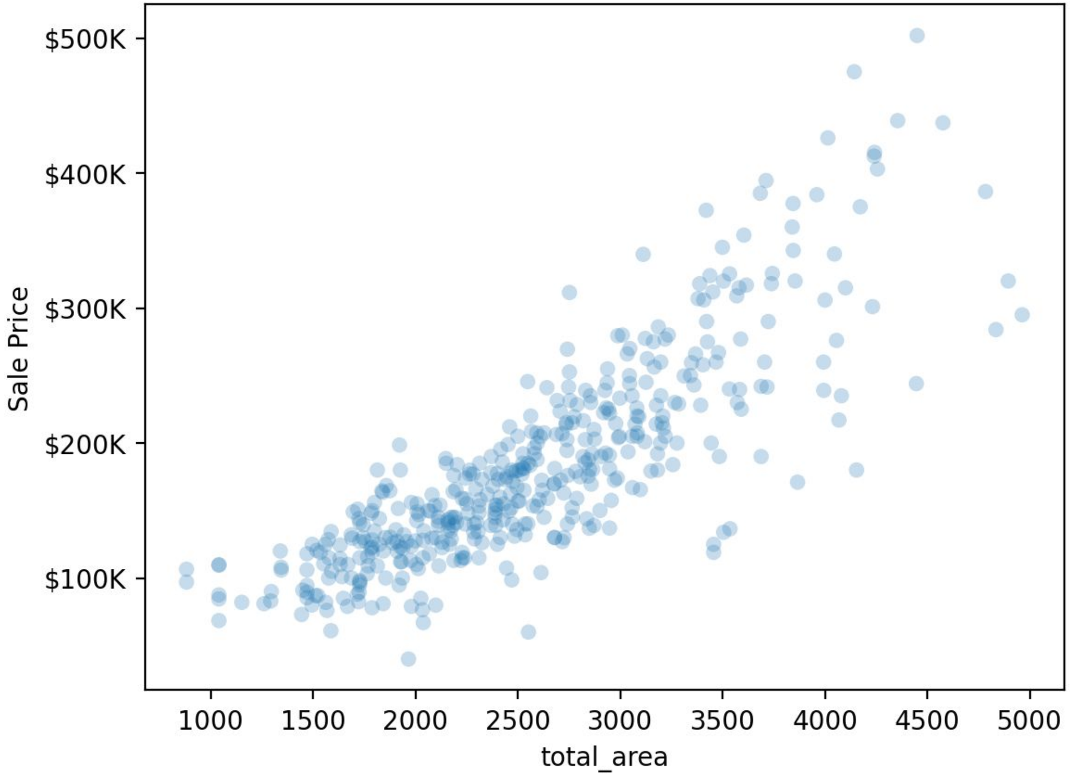
Goal of model selection

- Create a “model” that meets our performance requirements
- Build confidence that our measurement of performance will hold “in the real world”
- We want to approximate how our model will be used in production as best as possible.

Model Selection

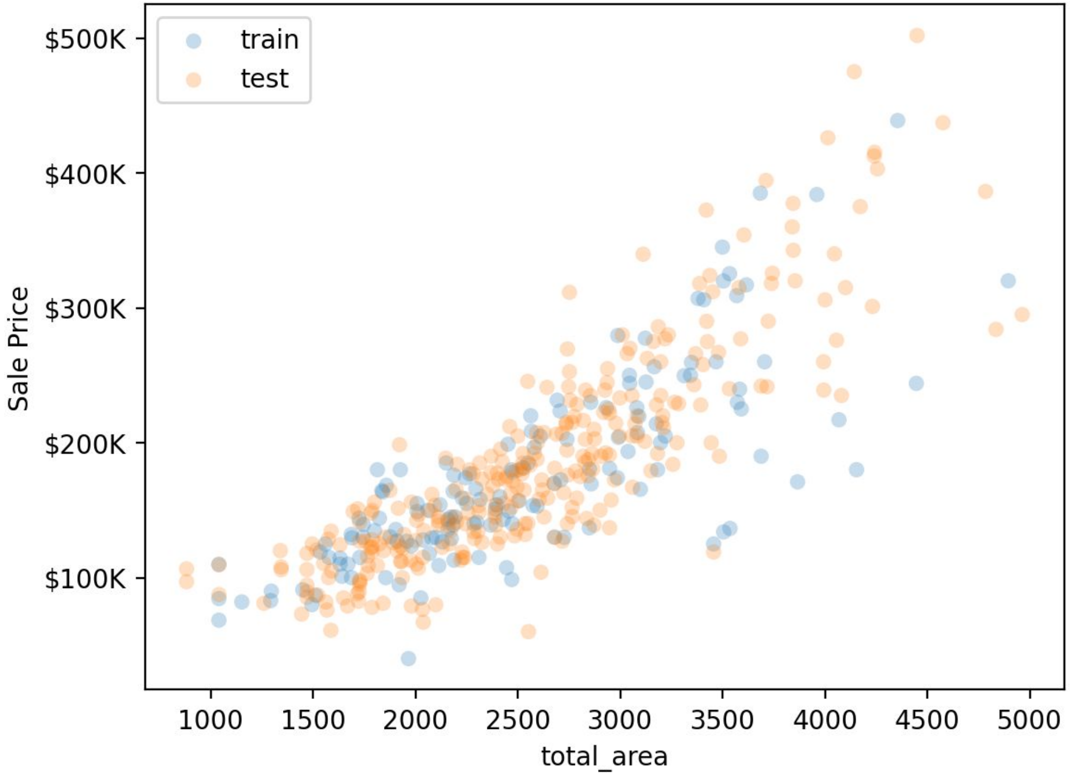
- Create a “model” that meets our performance requirements
- Build confidence that our measurement of performance will hold “in the real world”
- We want to approximate how our model will be used in production as best as possible.

Holdout set aka Train Test Split



Holdout set aka Train Test Split

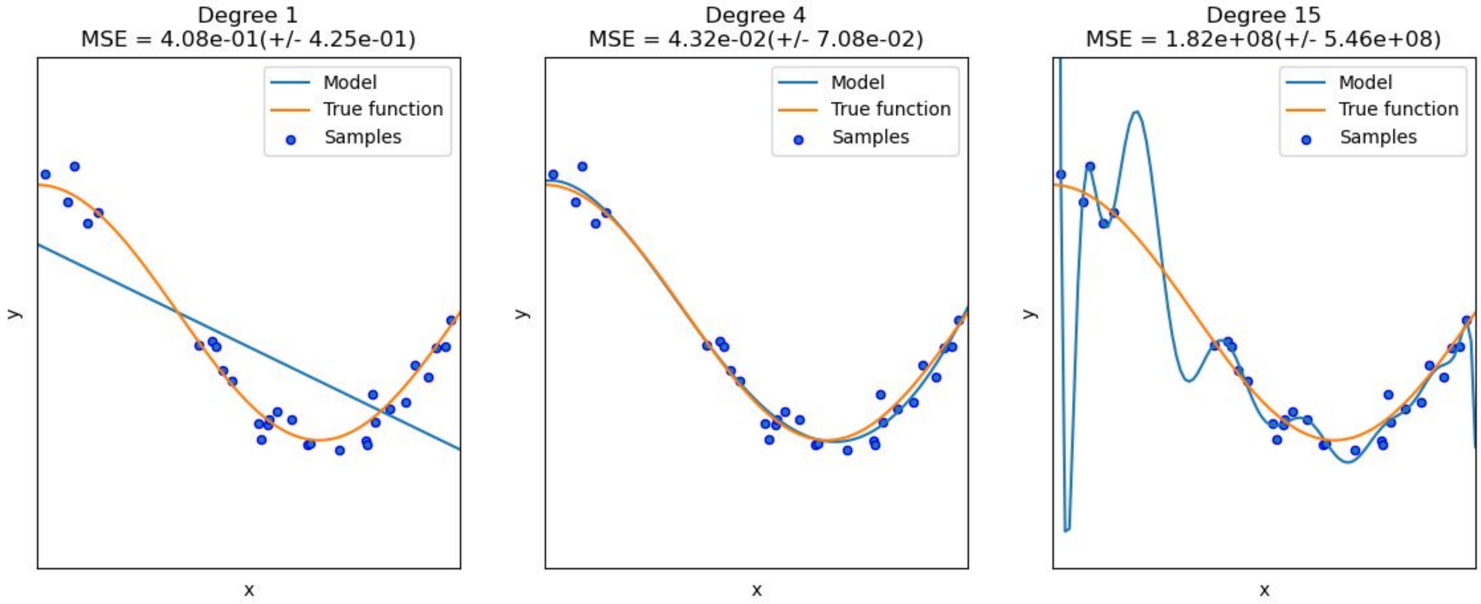
- Randomly separate data into training and test datasets.
- Train model on only the training set.
- Evaluate model using the test set.
- If data is IID, then randomly sampling can give us an unbiased estimate of the model’s “true” performance.



“With four parameters I can fit an elephant, and with five I can make him wiggle his trunk.”

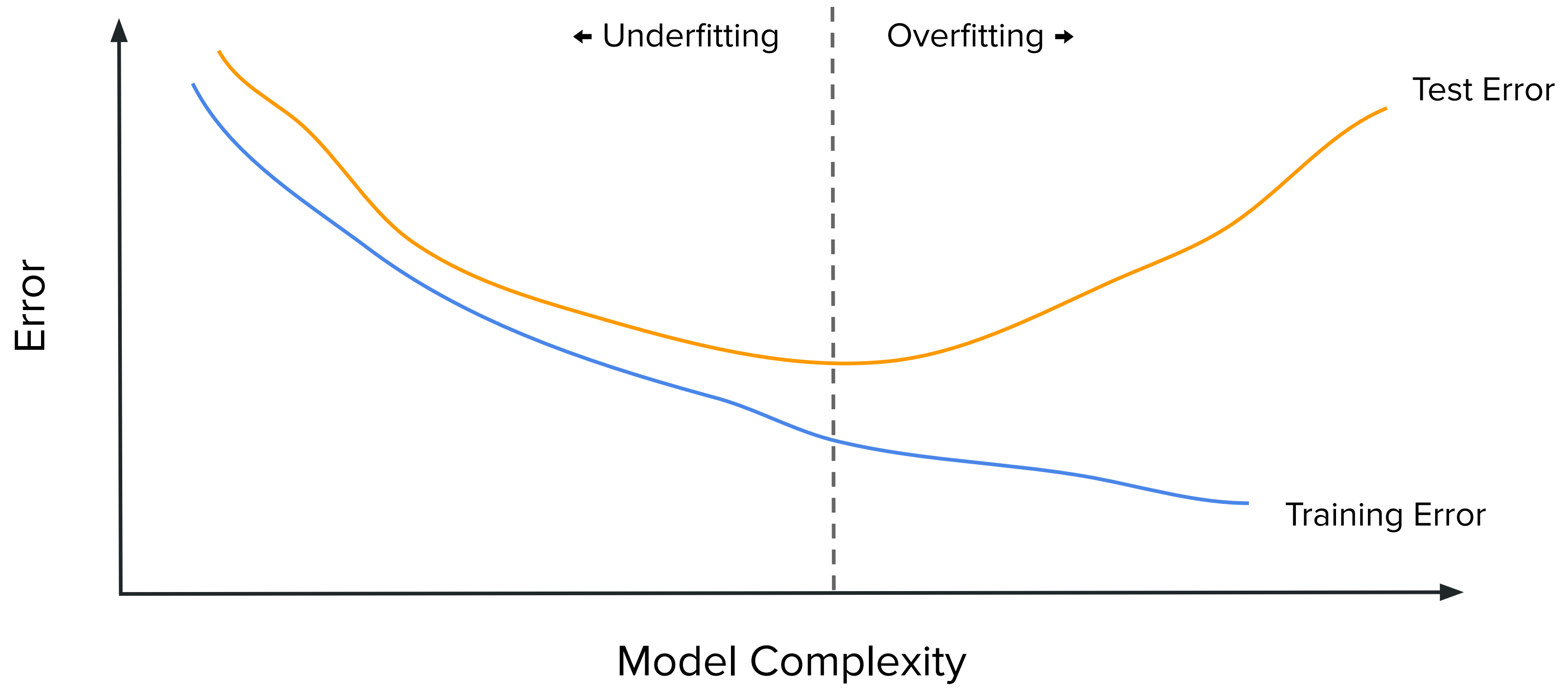
– John Von Neumann

Overfitting and Underfitting



https://scikit-learn.org/stable/auto_examples/model_selection/plot_underfitting_overfitting.html

Overfitting and Underfitting: Change Model Complexity



Overfitting and Underfitting: Regularization

$$\mathcal{L} = \sum_{i=1}^{n} \left(y_i - \hat{y}_i\right)^2$$

$$\mathcal{L} = \sum_{i=1}^{n} \left(y_i - \vec{\mathbf{X}}_i \cdot \vec{\beta}\right)^2$$

Overfitting and Underfitting: Regularization

$$\mathcal{L} = \sum_{i=1}^{n} \left(y_i - \hat{y}_i\right)^2$$

$$\mathcal{L} = \sum_{i=1}^{n} \left(y_i - \vec{\mathbf{X}}_i \cdot \vec{\beta}\right)^2$$

L2 Regularization / Ridge Regression

$$\mathcal{L} = \sum_{i=1}^{n} \left(y_i - \vec{\mathbf{X}}_i \cdot \vec{\beta}\right)^2 + \lambda_2 \sum_{j=1}^{p} \beta_j^2$$

Overfitting and Underfitting: Regularization

$$\mathcal{L} = \sum_{i=1}^{n} \left(y_i - \hat{y}_i\right)^2$$

$$\mathcal{L} = \sum_{i=1}^{n} \left(y_i - \vec{\mathbf{X}}_i \cdot \vec{\beta}\right)^2$$

L1 Regularization / Lasso

$$\mathcal{L} = \sum_{i=1}^{n} \left(y_i - \vec{\mathbf{X}}_i \cdot \vec{\beta}\right)^2 + \lambda_1 \sum_{j=1}^{p} |\beta_j|$$

Overfitting and Underfitting: Regularization

$$\mathcal{L} = \sum_{i=1}^{n} \left(y_i - \hat{y}_i\right)^2$$

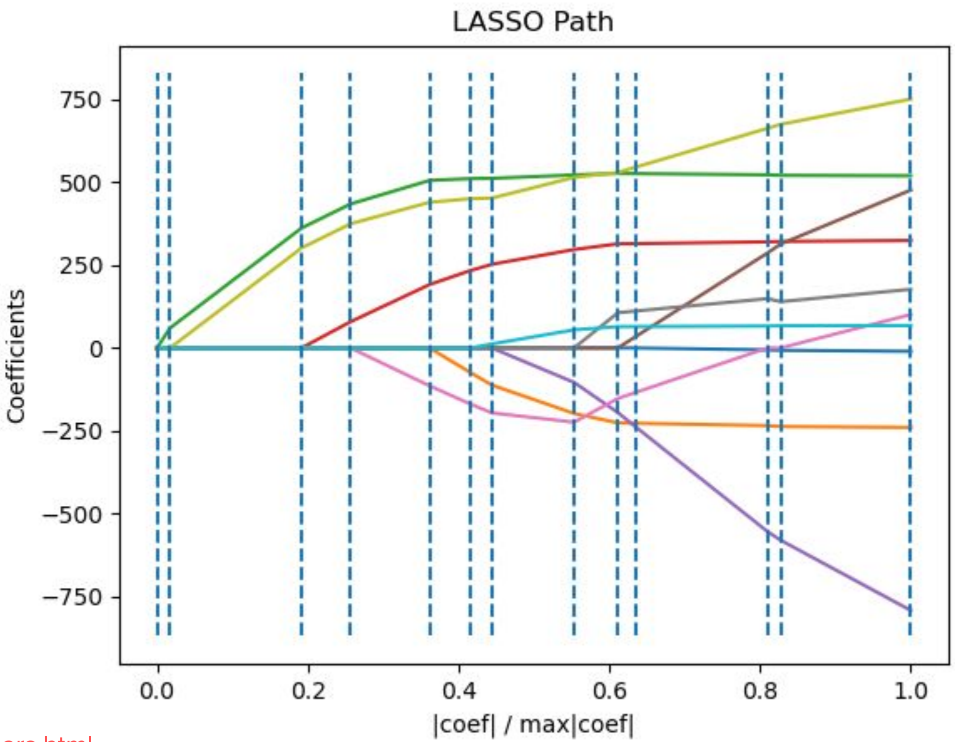
$$\mathcal{L} = \sum_{i=1}^{n} \left(y_i - \vec{\mathbf{X}}_i \cdot \vec{\beta}\right)^2$$

Elastic Net

$$\mathcal{L} = \sum_{i=1}^{n} \left(y_i - \vec{\mathbf{X}}_i \cdot \vec{\beta}\right)^2 + \lambda_1 \sum_{j=1}^{p} |\beta_j| + \lambda_2 \sum_{j=1}^{p} \beta_j^2$$

Overfitting and Underfitting: Regularization

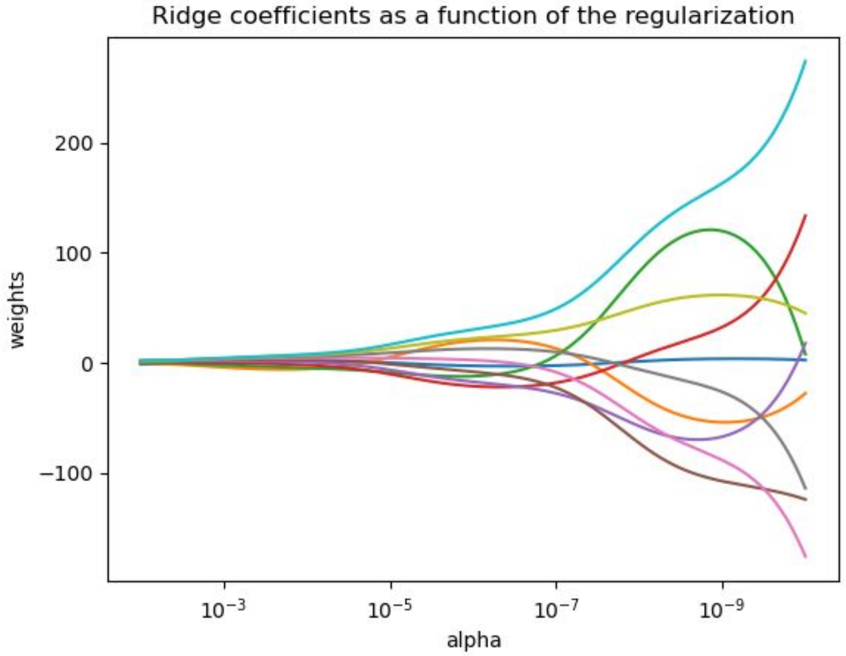
- L1 Regularization can induce "sparsity".
- As you increase the regularization strength, feature coefficients drop to zero.
- Can use this for feature selection.
- If features are correlated, then one may drop to 0 and the other one stays big.



https://scikit-learn.org/stable/auto_examples/linear_model/plot_lasso_lars.html

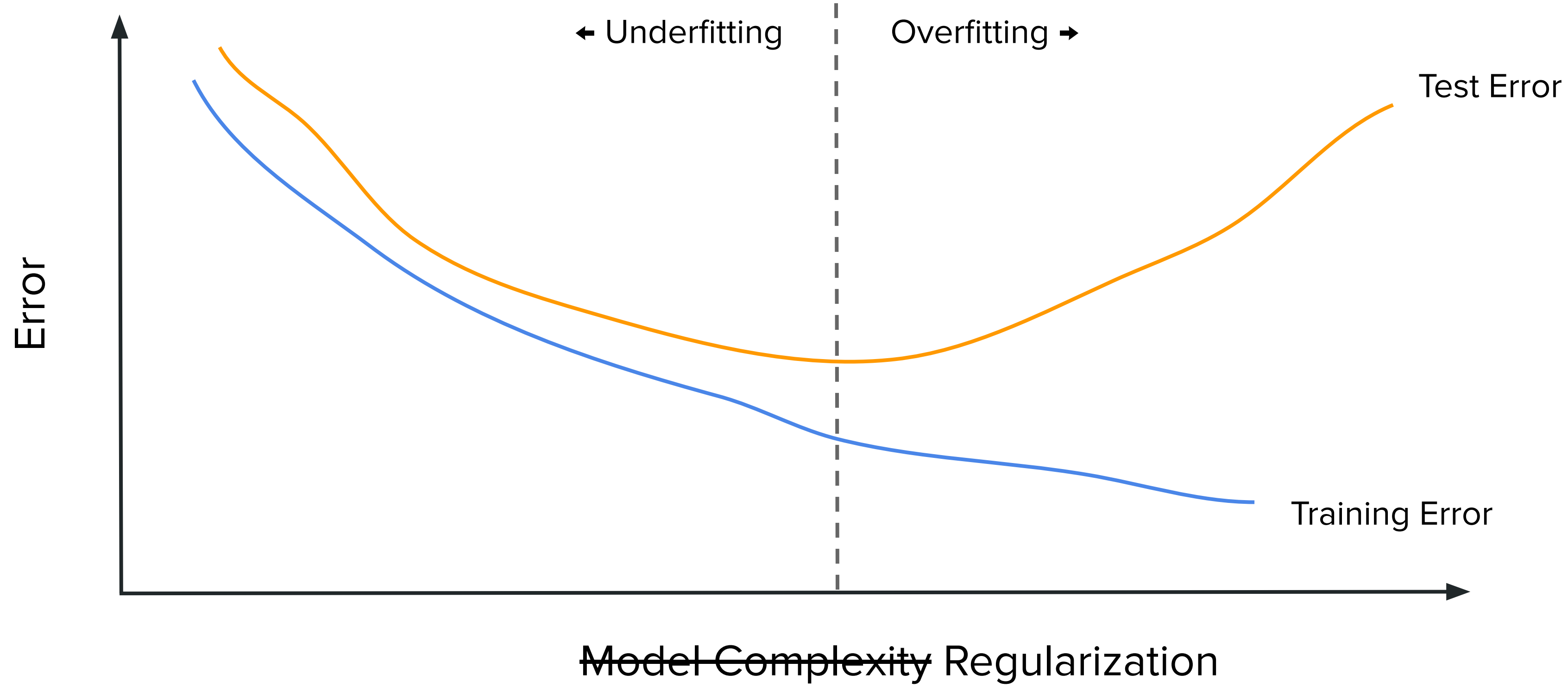
Overfitting and Underfitting: Regularization

- L2 Regularization decreases all coefficients together.
- Correlated features decrease together.
- Does not induce sparsity.
- Regularization is a type of "hyperparameter".
- A hyperparameter is a model parameter that you do not learn as part of the optimization process.



https://scikit-learn.org/stable/auto_examples/linear_model/plot_ridge_path.html

Overfitting and Underfitting: Regularization



Overfitting and Underfitting: Use More Data

Raschka (2018)

Figure 4: Learning curves of softmax classifiers fit to MNIST. https://arxiv.org/abs/1811.12808

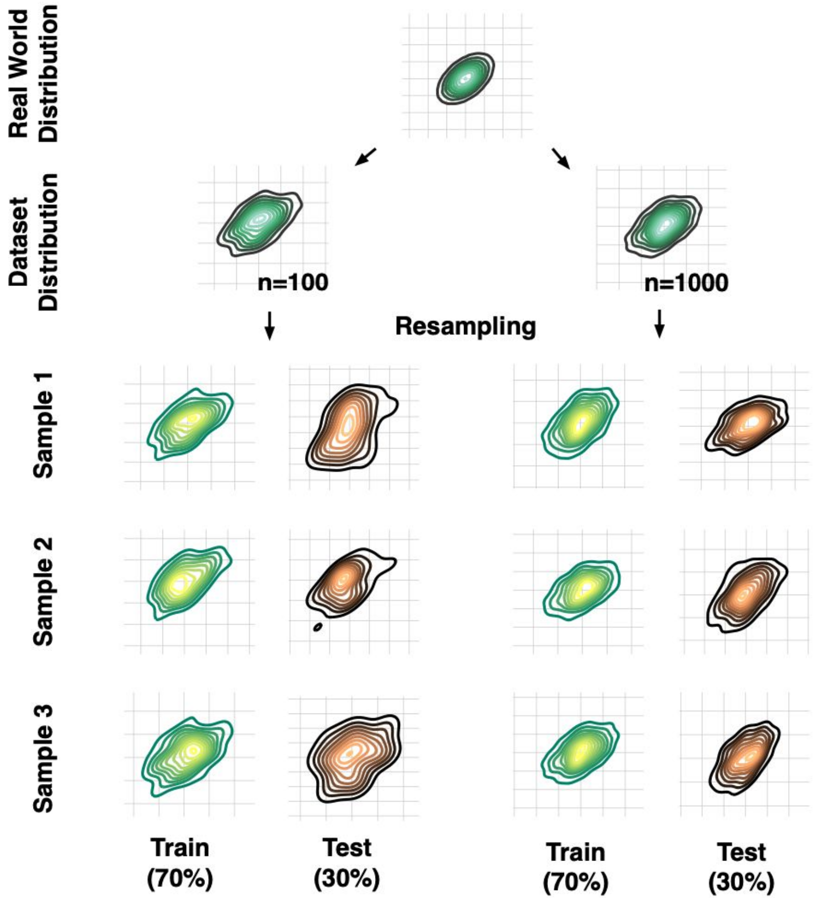


Figure 5: Repeated subsampling from a two-dimensional Gaussian distribution.

Raschka (2018)

https://arxiv.org/abs/1811.12808

- We want a robust estimate of how the model will work "in the wild".
- Small data introduces sampling bias.
- Many other biases can exist in your labelled data.
- Also, time!

Raschka (2018)

https://arxiv.org/abs/1811.12808

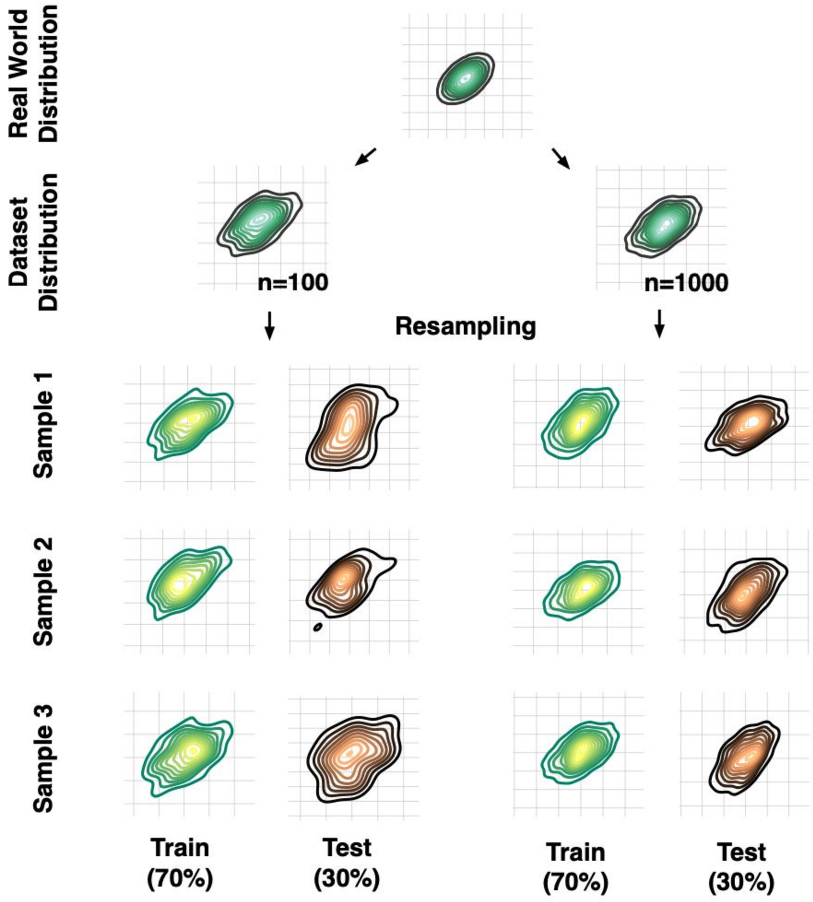
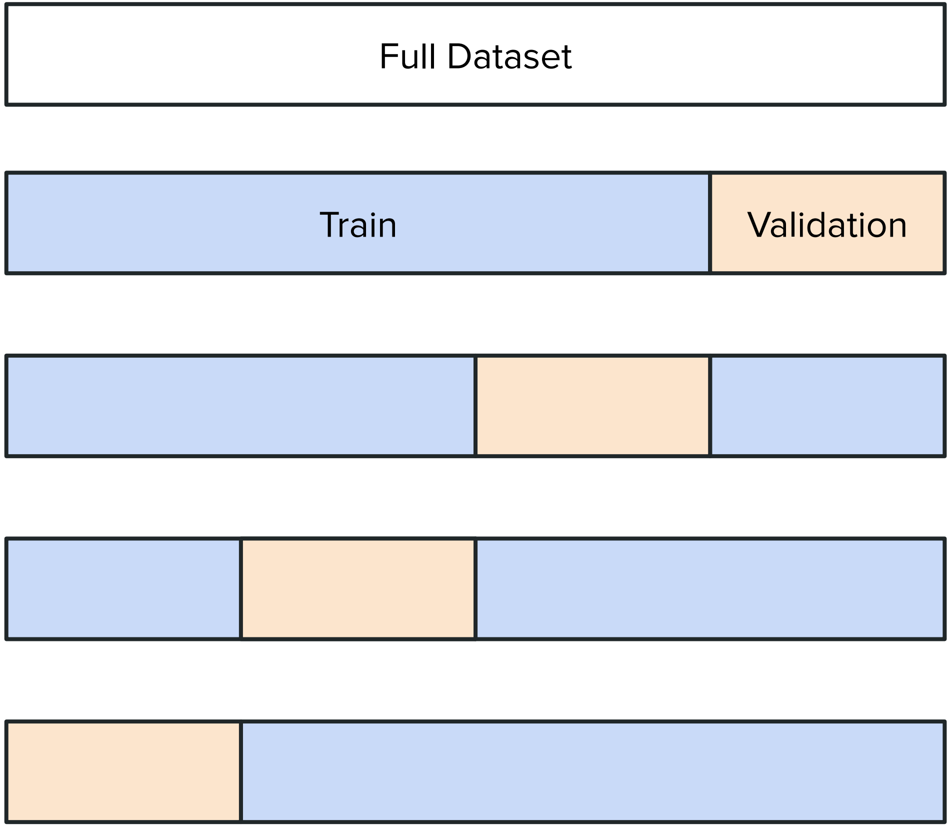


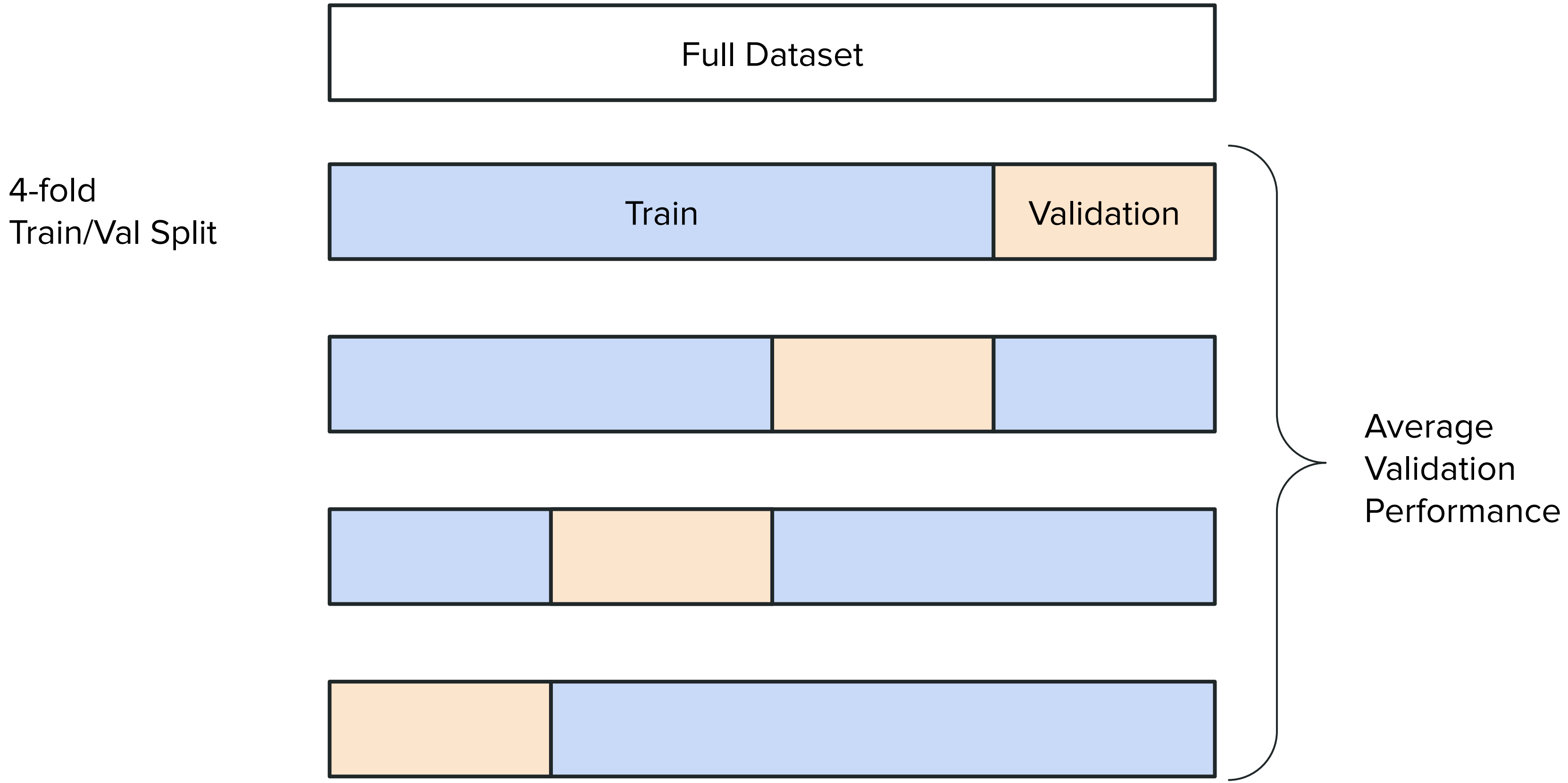
Figure 5: Repeated subsampling from a two-dimensional Gaussian distribution.

K-Fold Cross Validation

4-fold
Train/Val Split



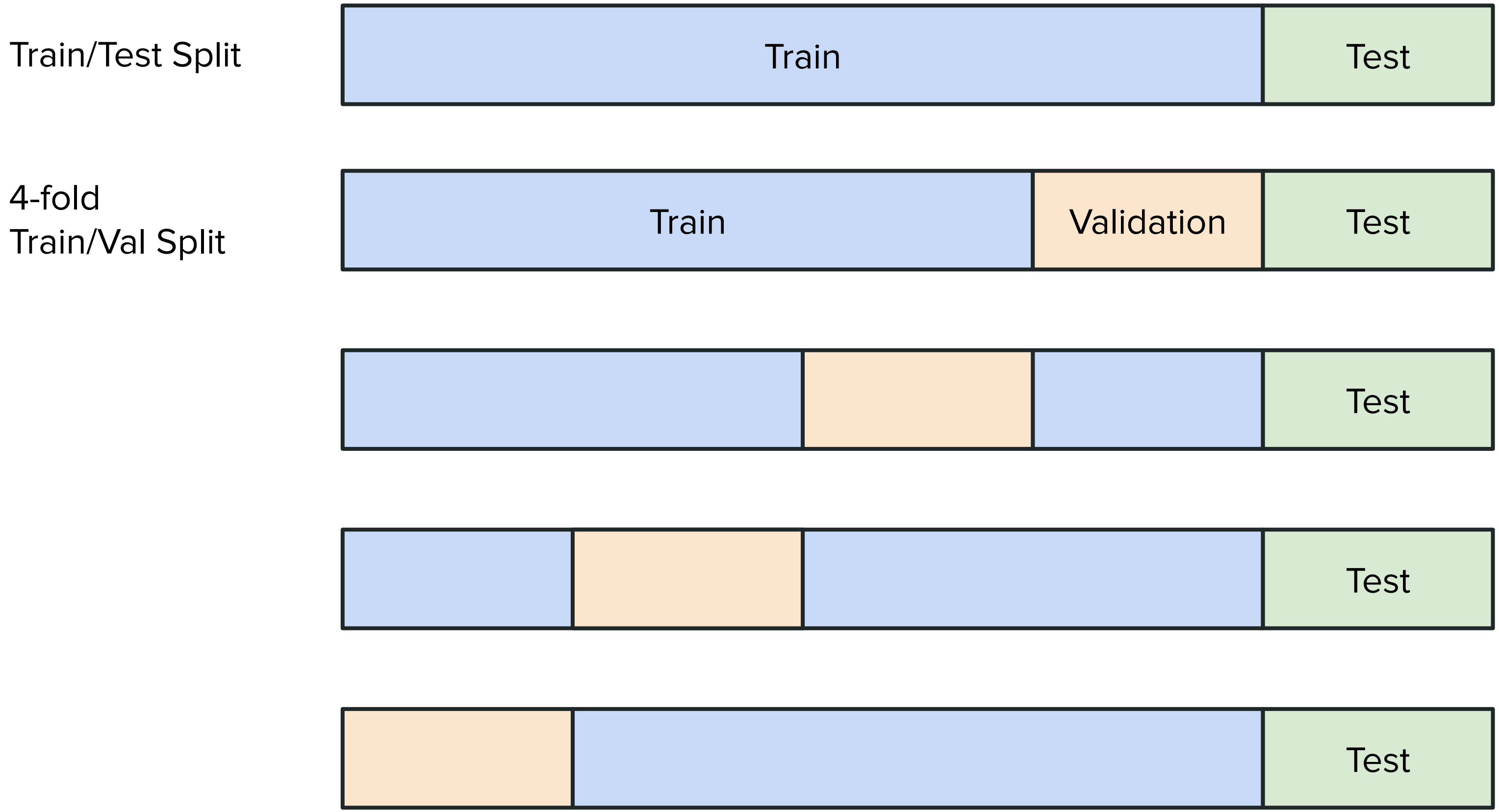
K-Fold Cross Validation



What’s wrong with this?

1. Split my data into training and test sets
2. Do some feature engineering
3. Fit a model on the training set
4. Evaluate model on the test set.
5. Do Steps 2-4 a couple more times, trying out different features.

Model Selection via K-Fold Cross Validation



Model Selection via K-Fold Cross Validation aka Hyperparameter Optimization

1. Separate test set at the beginning.
2. Split train into K disjoint train/validation partitions.
3. For each model configuration (e.g. hyperparameter combo)
   - For each K fold:
     - Fit model on training data.
     - Evaluate model on validation data.
   - Average performance across K folds
4. Determine model configuration with optimal avg performance.
5. Train optimal model on *full* training set.
6. Evaluate model on test set.

Feature Engineering
& Model Selection with scikit-learn