Data Analysis

Chapter IV – part 2:
Regression evaluation
Data leakage

The modeling process

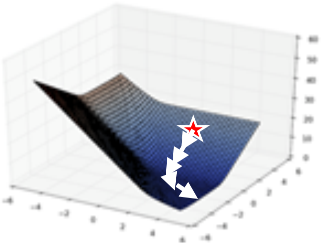
4 Stuy

1. Choose a model	How should we represent the world?
2. Choose a loss function	How do we quantify prediction error?
3. Fit the model	How do we choose the best parameters of our model given our data?
4. Evaluate model performance	How do we evaluate whether this process gave rise to a good model?

$$\hat{\mathbb{Y}} = \mathbb{X}\theta$$

MSE: $R(\theta) = \frac{1}{n}(||\mathbb{Y} - \hat{\mathbb{Y}}||_2)^2$

Minimize average loss with **calculus, geometry, algorithms!**



Evaluating regression models

we can do Plot (help in lower Dim)

1. **Visualize data, compute statistics:**

- Plot original data (if we are in lower dimensions)
- Compute column means, standard deviation.
- If we want to fit a linear model, compute correlation *r* (also might be R^2)

2. **Performance metrics:**

Root Mean Square Error (RMSE) $\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$

- It is the square root of **MSE**, which is the average loss that we've been minimizing to determine optimal model parameters.
- RMSE is in the **same units** as y.
- A lower RMSE indicates more "accurate" predictions (lower "average loss" across data)

3. **Visualization:**

Look at a residual plot of errors to visualize the difference between actual and predicted values.

Evaluating regression models - R^2

- But how low is low? RMSE can be difficult to interpret by itself.
- A more easily interpretable number is the $\boxed{R^2 \text{ value}}$.
    - An alternative way to measure how good a fit the model is to the data.
    - R^2 takes values between 0 and 1
    - We define R^2 in relation to the SSR and TSS (Total Sum of Squares).

Measure

$$SSR = \sum_{i=1}^{n}(y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i)^2 \quad \text{and} \quad TSS = \sum_{i=1}^{n}(y_i - \bar{y})^2. \qquad R^2 = 1 - \frac{SSR}{TSS}$$

Always Predict the Mean

Total Var In my Data

-
    - SSR measures the amount of variability left unexplained after the linear regression
    - TSS measures the total variance in the target data
- R^2 shows the proportion of the variance explained by the model
    - For regression with 1 predictor R^2 relates to the value of correlation
    - For multiple regression problems we usually control for the number of predictors (p) and the number of data points (n)

$$\text{adj.}R^2 = 1 - \frac{(1 - R^2)(n-1)}{n-p-1}$$

If we have more than 1 Para

Evaluating regression models – Residual plots

Residual plots (the code)

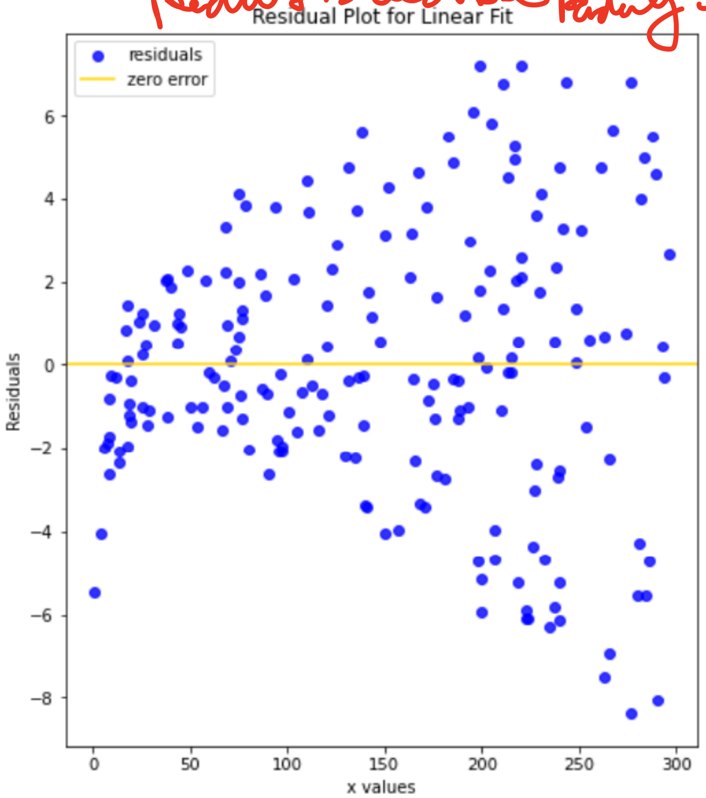
```
#we do it for the scikit learn model
lr = linear_model.LinearRegression() # create a linear regression ob
# scikit-learn doesn't work as well with pandas, so we have to extra
x = advert['TV'].values.reshape(advert['TV'].shape[0],1)
y = advert['Sales'].values.reshape(advert['Sales'].shape[0],1)
lr.fit(X=x, y=y)
preds= lr.predict(x)
resid= y - preds

fig, ax = plt.subplots(1, 2, figsize=(15, 8))
ax[0].scatter(x=x, y=resid, color='blue', alpha=0.8, label='residual
ax[0].axhline(y=0, color='gold', label='zero error')
ax[0].set_xlabel('x values')
ax[0].set_ylabel('Residuals')
ax[0].set_title('Residual Plot for Linear Fit')
ax[0].legend(loc='best')
ax[1].hist(resid, color='blue', alpha=0.8, label='residuals', bins=2
ax[1].axvline(x=0, color='gold', label='zero error')
ax[1].set_xlabel('Value bins')
ax[1].set_ylabel('Residuals')
ax[1].set_title('Residual Plot for Linear Fit')
ax[1].legend(loc='best')
```

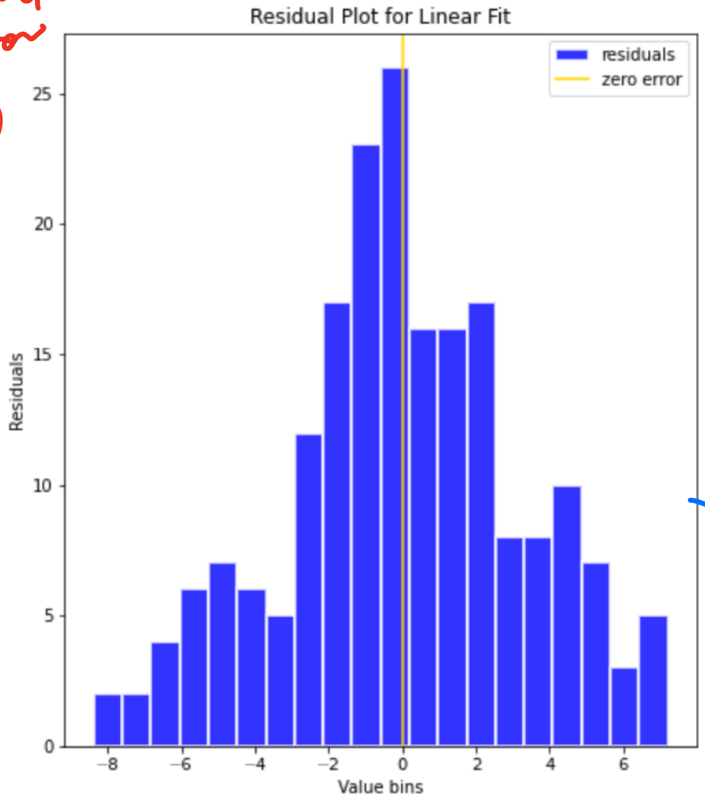
Evaluating regression models – Residual plots

Residual plots (how do they look like)

Residuals should be Randomly spread around 0



Scatter Plot



Histogram

Sometimes, instead of the histogram of residuals we might be given the normal-probability plot https://en.wikipedia.org/wiki/Normal_probability_plot

The Variance Seems Normally

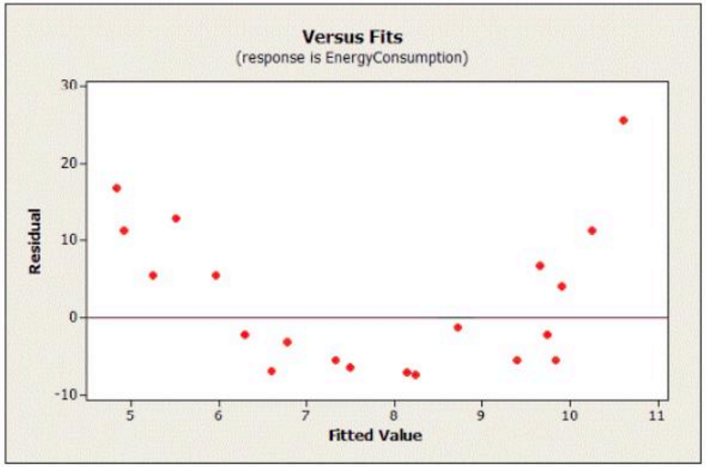
Residuals: how much off any Y will each fit

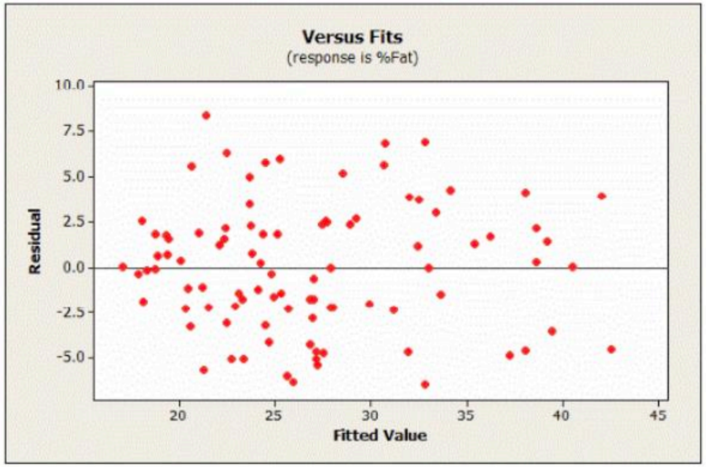
Evaluating regression models

- Which model is a better fit?

  X-axis here shows fitted values (no problem for us)

A. Left
B. Right
C. Equally good fits
D. Equally bad fits
E. Unicorns are pink

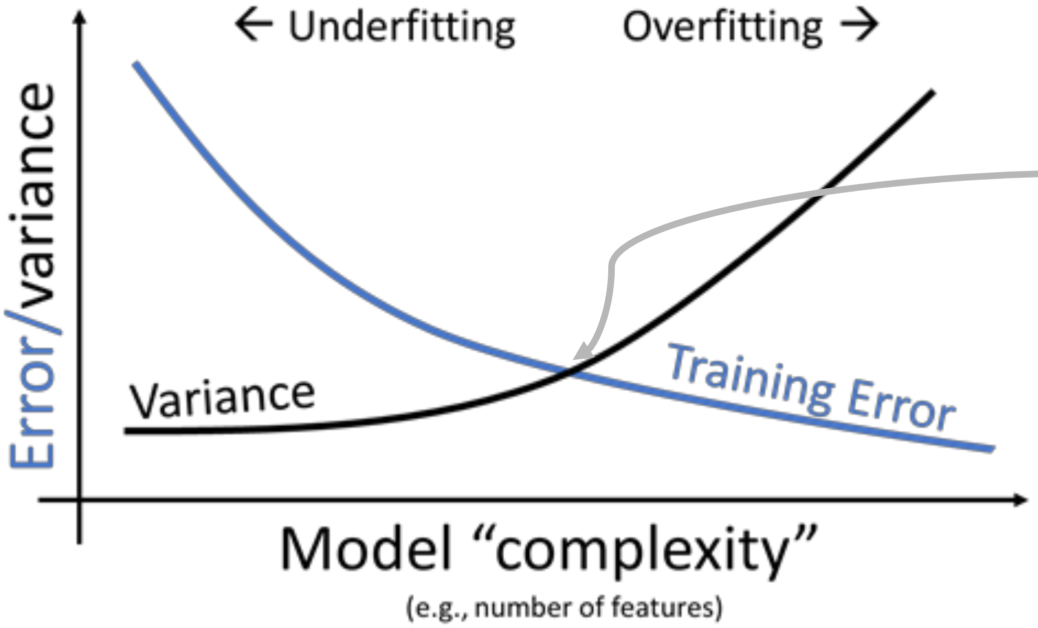




Let's pick up the variance thingie...

At the end of the notebook, we had two lingering thoughts:

- The idea of "unseen" data – data that the model did not encounter during training
- The idea of model complexity – a model's complexity influences if it under- or overfits



Our goal: find this "sweet spot"

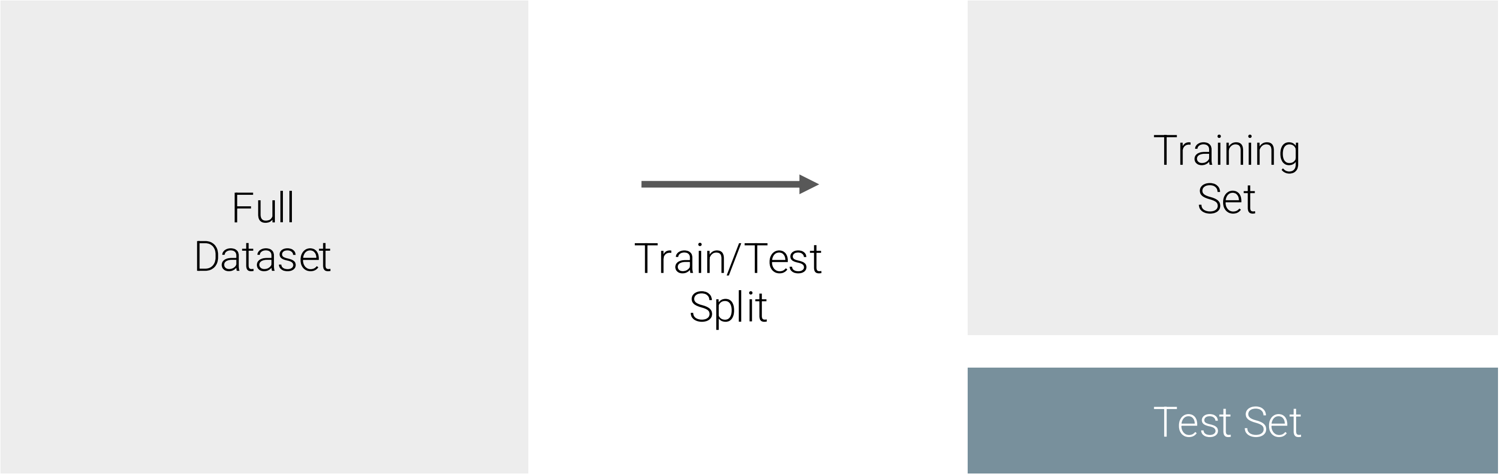
Test Sets

1st Separation We Make

A **test set** is a portion of our dataset that we set aside for testing purposes.

- We do *not* consider the test set when fitting/training the model.
- The test set is only ever touched once: to compute the performance (MSE, RMSE, etc) of the model *after* all fine-tuning has been completed.

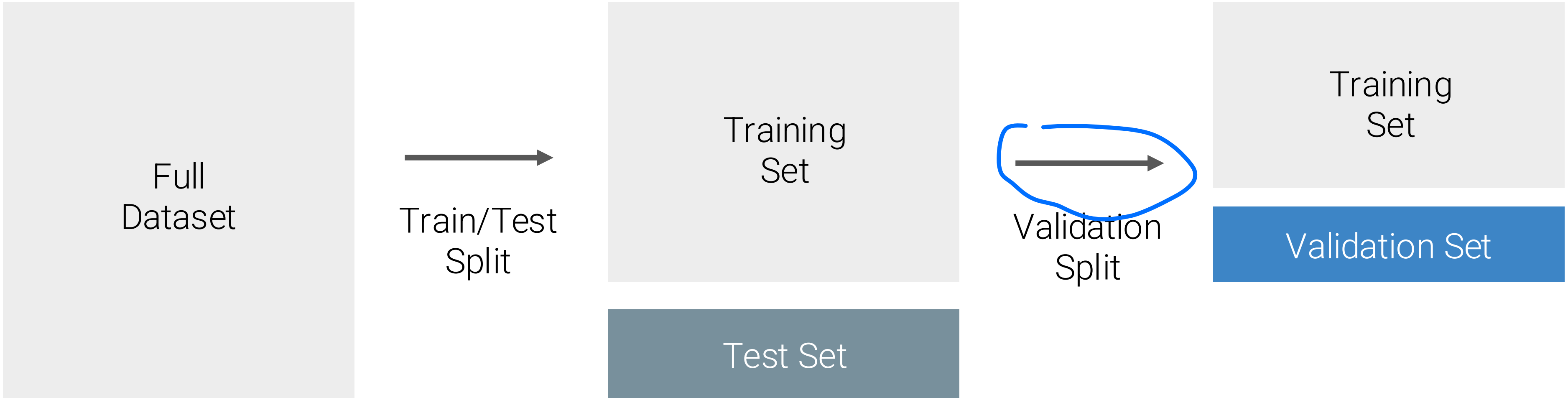
Our workflow for modeling: First, perform a **train-test split** (see documentation). Consider only the training set when designing the model. Then, evaluate on the test set.



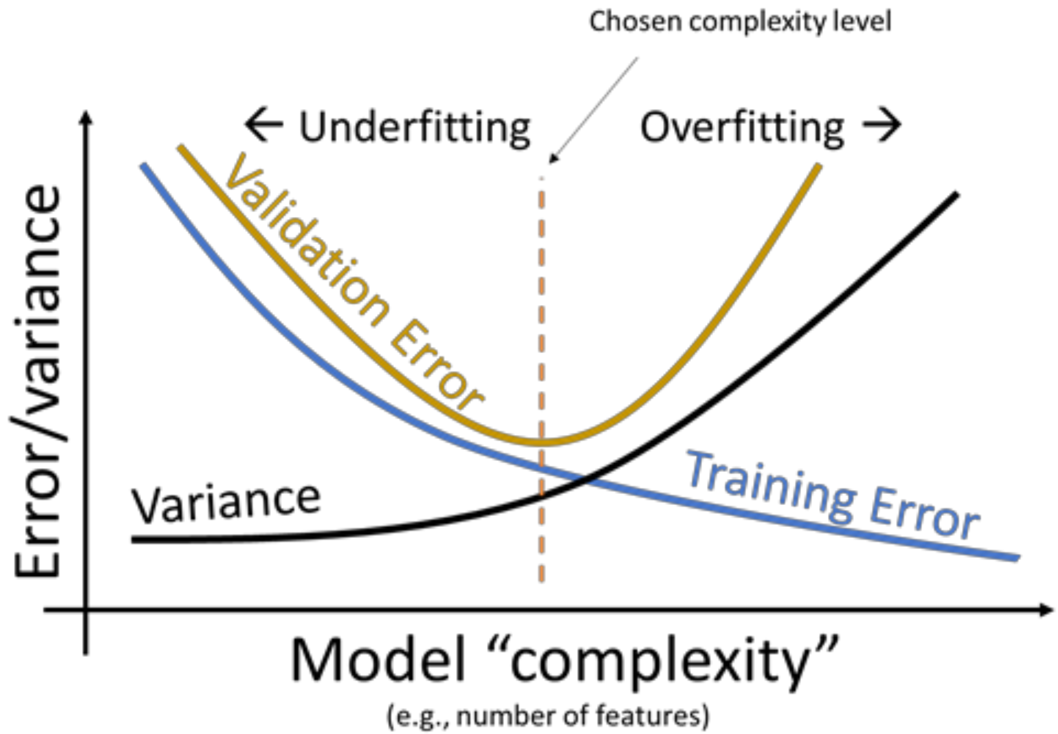
Validation Sets

A **validation set** is a portion of our *training set* that we set aside for assessing model performance while it is *still being developed*.

- Train model on the training set. Assess performance on the validation set. Adjust the model, then repeat.
- After *all* model development is complete, assess final performance on the test set.



Model Complexity



Typically, as model complexity increases:

- Training error decreases
- Variance increases
- Error on validation set decreases, then increases

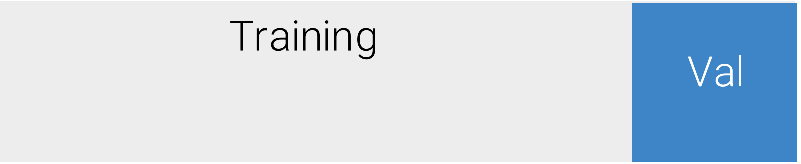
Our goal: Choose the model complexity that minimizes validation error.

That will come handy when we discuss regularization

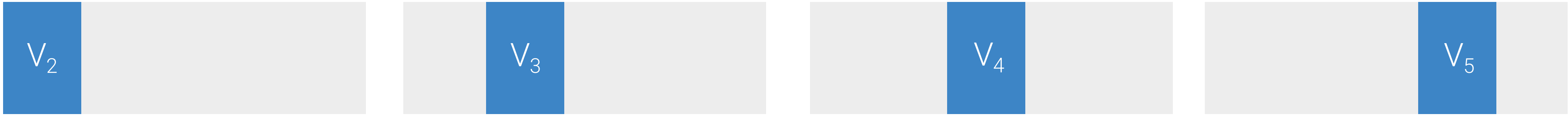
Validation Folds

In our original validation split, we set aside x% of the training data to use for validation.

- For example, 20% of the training data is used for validation



We could have selected *any* 20% portion of the training data for validation.



In total, there are 5 non-overlapping “chunks” of datapoints we could set aside for validation.

Validation Folds

The common term for one of these chunks is a "fold".

- Our training data has 5 folds, each containing 20% of the datapoints.

Another perspective: we actually have 5 validation sets "hidden" in our training set.

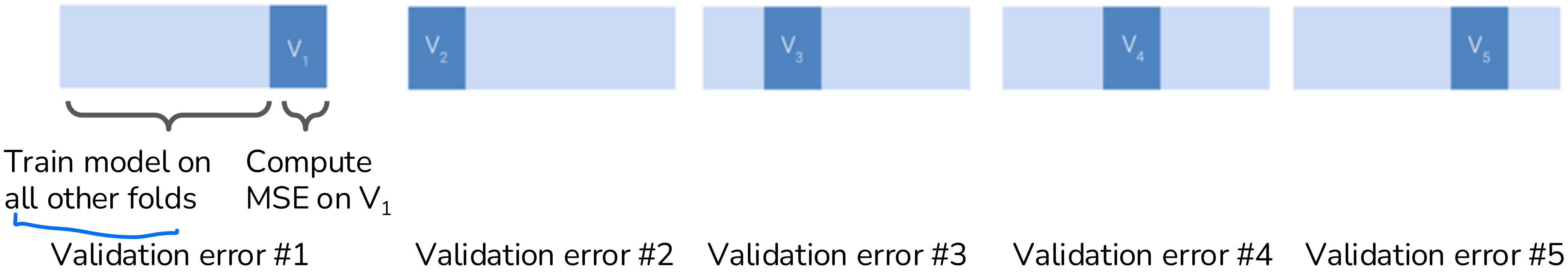
In **cross-validation**, we perform validation splits for *each* of these folds.

K-Fold Cross-Validation

For a dataset with K folds:

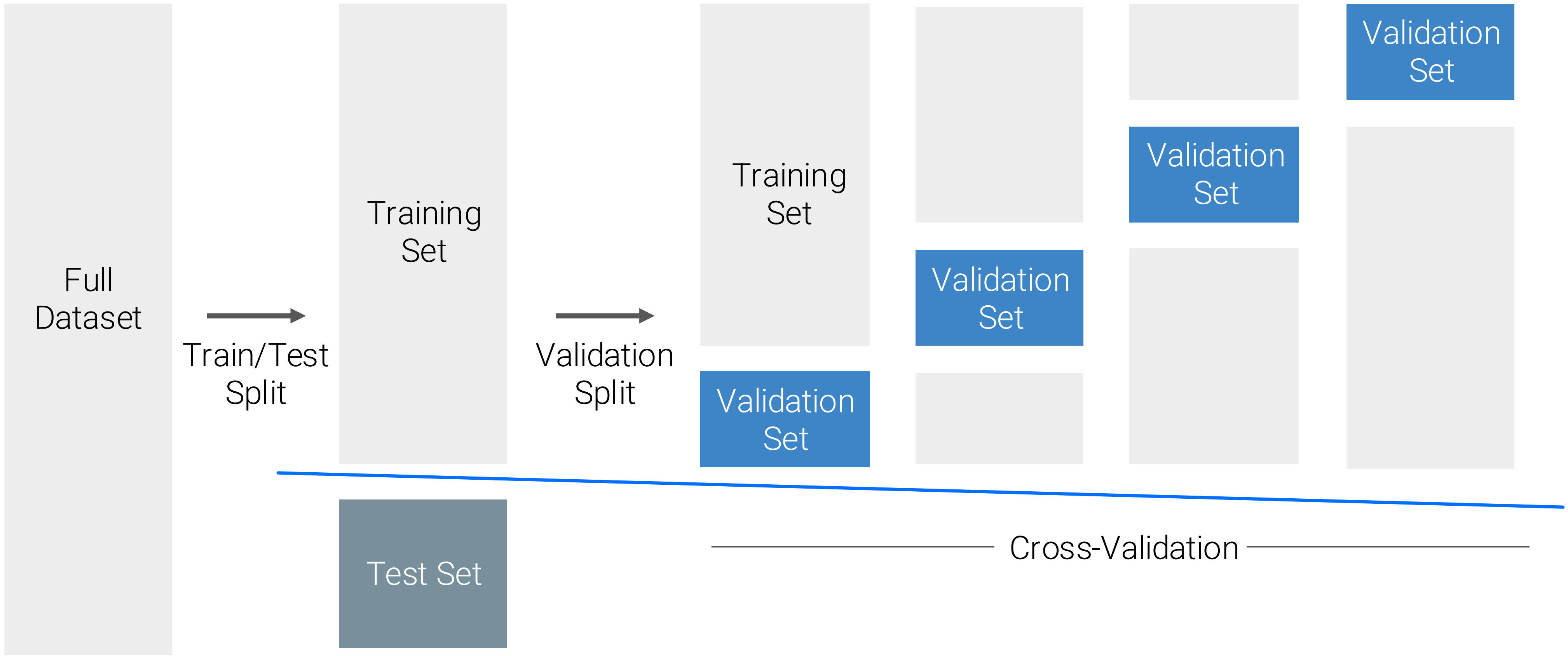
- Pick one fold to be the validation fold.
- Train model on data from every fold *other* than the validation fold.
- Compute the model's error on the validation fold and record it.
- Repeat for all K folds.

The **cross-validation error** is the average error across all K validation folds.



Cross-validation error = mean of validation errors #1 to #5

Model Selection Workflow



Set Aside

Hyperparameters

Cross-validation is often used for **hyperparameter** selection.

Hyperparameter: Value in a model chosen *before* the model is fit to data.

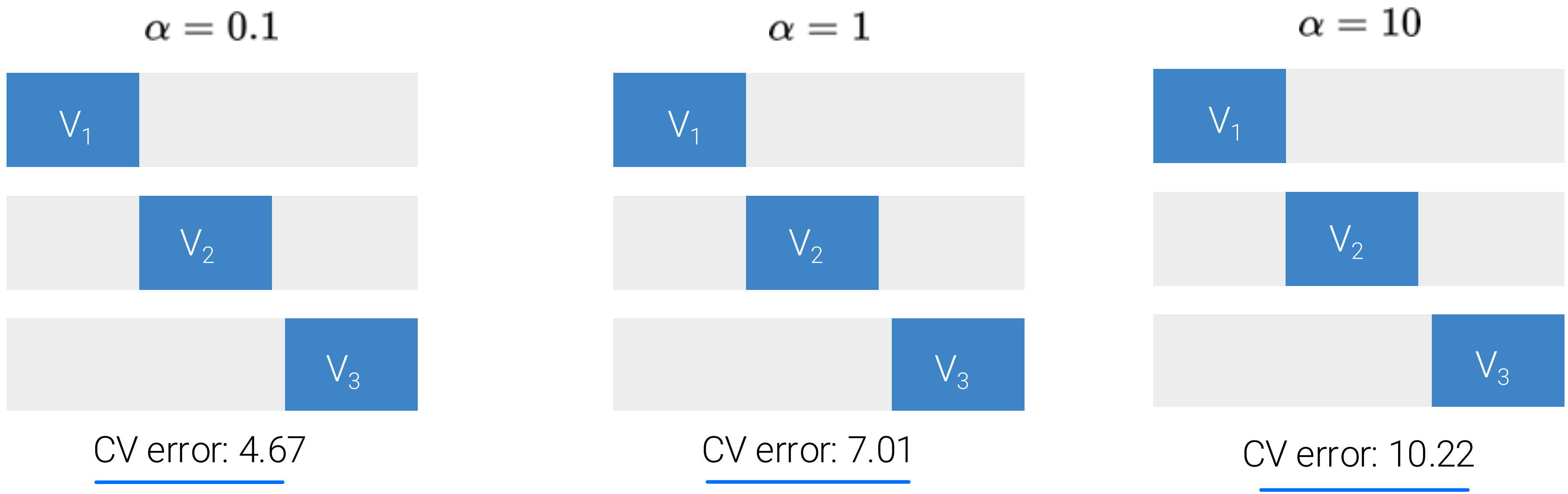
- Cannot solve for hyperparameters via calculus, OLS, gradient descent, etc – we must choose it ourselves.
- Examples
  - Degree of polynomial model α
  - Gradient descent learning rate,
  - Regularization penalty, λ (coming up)

Hyperparameter Tuning

To select a hyperparameter value via cross-validation:

- List out several different "guesses" for the best hyperparameter.
- For each guess, run cross-validation to compute the CV error for that choice of hyperparameter value.
- Select the hyperparameter value with lowest CV error.

Example: Guesses for learning rate are 0.1, 1, and 10. We decide to apply 3-fold cross-validation.



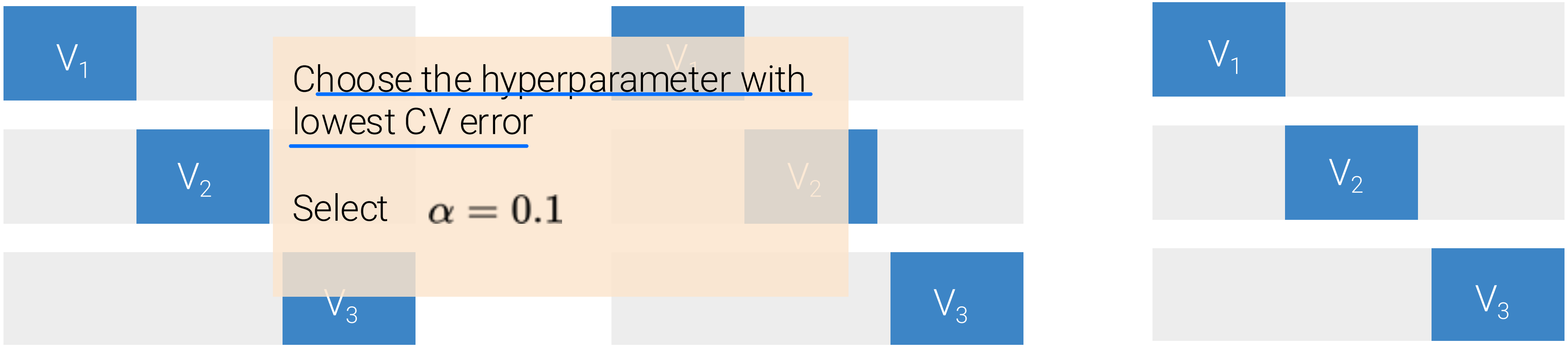
Hyperparameter Tuning

To select a hyperparameter value via cross-validation:

- List out several different "guesses" for the best hyperparameter
- For each guess, run cross-validation to compute the CV error for that choice of hyperparameter value
- Select the hyperparameter value with lowest CV error

Example: Guesses for learning rate are 0.1, 1, and 10. We decide to apply 3-fold cross-validation.

$\alpha = 0.1$ | $\alpha = 1$ | $\alpha = 10$



Choose the hyperparameter with lowest CV error

Select $\alpha = 0.1$

CV error: 4.67 | CV error: 7.01 | CV error: 10.22

Data leakage

- Some form of the label “leaks” into the features
- This same information is not available during inference

Data leakage: example 1

- Problem: detect lung cancer from CT scans
- Data: collected from hospital A
- Performs well on test data from hospital A
- Performs poorly on test data from hospital B

Patient ID	Date	Doctor note	Medical record	Scanner type	CT scan

Data leakage: example 1

- Problem: detect lung cancer from CT scans
- Data: collected from hospital A
- Performs well on test data from hospital A
- Performs poorly on test data from hospital B

Patient ID	Date	Doctor note	Medical record	Scanner type	CT scan

At hospital A, when doctors suspect that a patient has lung cancer, they send that patient to a higher-quality scanner

Data leakage: example 2

- Problem: predicting how many views an article will get
- Data: historical data on the site
- Where might data leakage come from?

150%. Careful

Article ID	Date	Title	Article	Author	Language	Translations

Data leakage: example 2

- Problem: predicting how many views an article will get
- Data: historical data on the site

Not leakage because author popularity also available during inference

Article ID	Date	Title	Article	Author	Language	Translations

The site only translate articles that are already gaining attention

Causes of data leakage

1. Splitting time-correlated data randomly instead of by time

Partition: shuffle then split

	Week 1	Week 2	Week 3	Week 4	Week 5
Test split	X11	X21	X31	X41	X51
Valid split	X12	X22	X32	X42	X52
Train split	X13	X23	X33	X43	X53
	X14	X24	X34	X44	X54
	...	...	...	...	...

Aim for similar distributions of labels across splits
e.g. each split has 90% NEGATIVE, 10% POSITIVE

Partition: shuffle then split

	Week 1	[illegible]	[illegible]	Week 4	Week 5
Test split	X11	[illegible]	[illegible]	X41	X51
Valid split	X12	[illegible]	[illegible]	X42	X52
Train split	X13	[illegible]	[illegible]	X43	X53
	X14	[illegible]	[illegible]	X44	X54
	...	...	...	...	...

A source of data leakage. Examples:

- stock price prediction
- song recommendation

Solution: split data by time

Train split

Week 1	Week 2	Week 3	Week 4	Week 5
X11	X21	X31	X41	X51
X12	X22	X32	X42	X52
X13	X23	X33	X43	X53
X14	X24	X34	X44	X54
...	...	...	...	...

Valid split

Test split

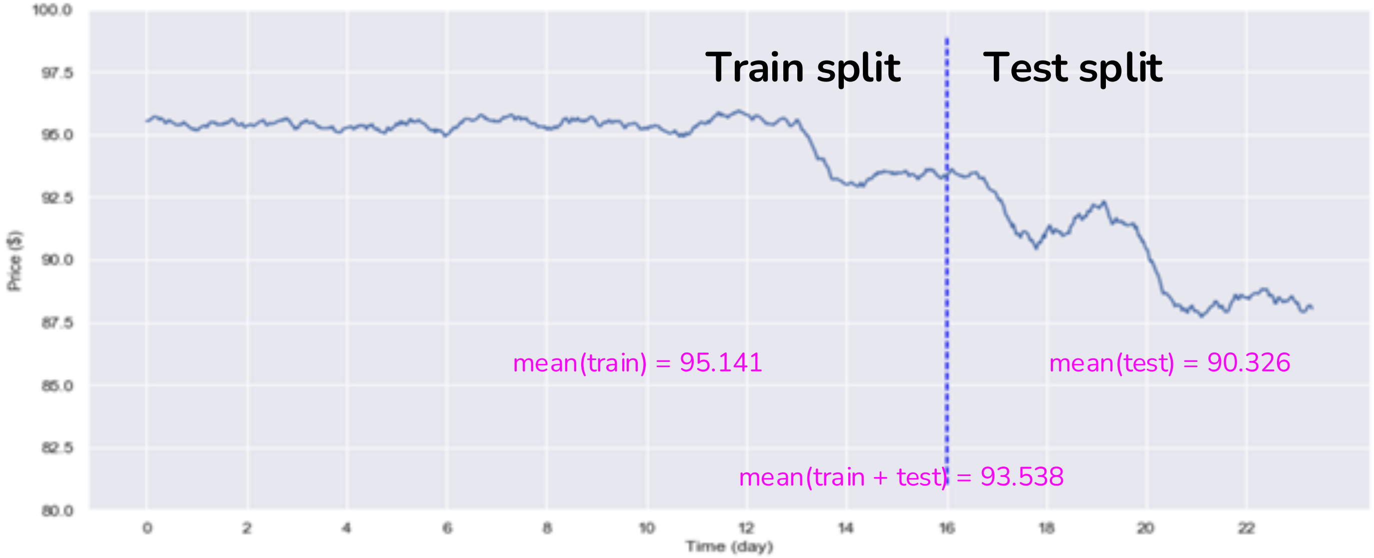
Also forces you to think about
the cold-start problem

Causes of data leakage

1. Splitting time-correlated data randomly instead of by time
2. Data processing before splitting
    a. Use the whole dataset (including valid/test) to generate global statistics/info

2. Data processing before splitting

- Use the whole dataset (including valid/test) to generate global statistics/info
  - mean, variance, min, max, n-gram count, vocabulary, etc.
- Statistics are then used to process test data
  - scale, fill in missing values, etc.



- Solution:
  - Split your data before scaling/filling in missing values
  - Split even before any EDA to ensure you're blind to the test set

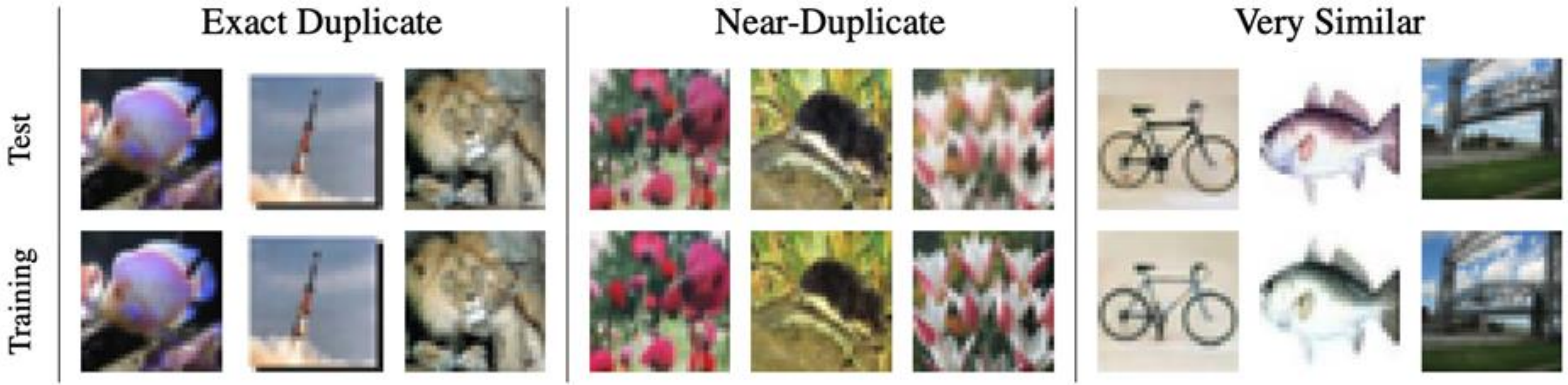
EDA

Causes of data leakage

1. Splitting time-correlated data randomly instead of by time
2. Data processing before splitting
3. Poor handling of data duplication before splitting
    a. Test set includes data from the train set

3. Poor handling of data duplication before splitting

- Datasets come with duplicates & near-duplicates
  - 3.3% CIFAR-10 and 10% CIFAR-100 test images have dups in training set
  - Removing dups increases errors 17.05% -> 19.38% on CIFAR-100 [PyramidNet-272-200}



Do we train on test data? Purging CIFAR of near-duplicates (Barz & Denzler, 2019)

3. Poor handling of data duplication before splitting

- Datasets come with duplicates & near-duplicates
- Oversampling can cause duplications (relevant for clinic 1?)

- Solution:
  - Deduplicate data before splitting
  - Oversample after splitting (we will discuss this when we cover class imbalances)

Causes of data leakage

1. Splitting time-correlated data randomly instead of by time
2. Data processing before splitting
3. Poor handling of data duplication before splitting
4. Group leakage
    - A group of examples have strongly correlated labels but are divided into different splits
    - Example: CT scans of the same patient a week apart
    - Solution: Understand your data and keep track of its metadata

Causes of data leakage

1. Splitting time-correlated data randomly instead of by time
2. Data processing before splitting
3. Poor handling of data duplication before splitting
4. Group leakage
5. Leakage from data generation & collection process
    - Example: doctors send high-risk patients to a better scanner
    - Solution: Data normalization + subject matter expertise

How to detect leakage?

1. Measure correlation of a feature with labels
   - A feature alone might not cause leakage, but 2 features together might
2. Feature ablation study
   - If removing a feature causes the model performance to decrease significantly, figure out why.
3. Monitor model performance as more features are added
   - Sudden increase: either a very good feature or leakage!