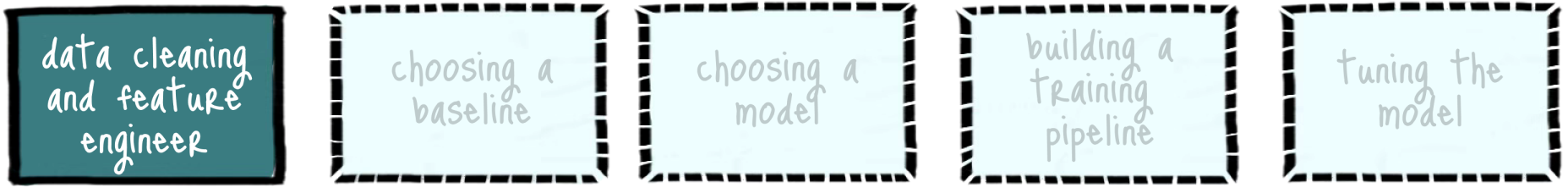
session 2

how to build a model (that works)

ml.school

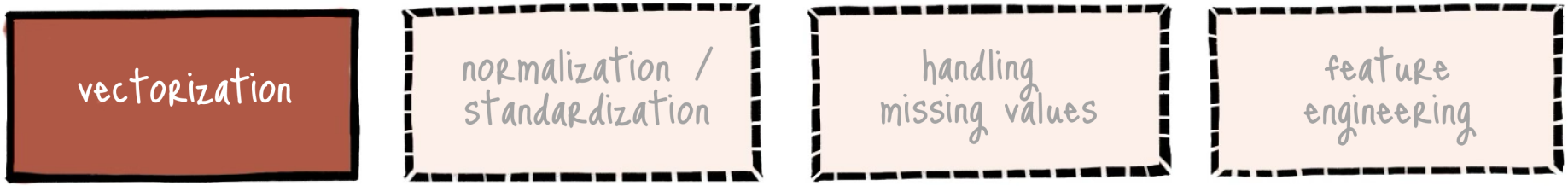
1. feature engineering
2. choosing a baseline
3. choosing a model
4. training pipeline
5. tuning the model

The hardest things in machine learning are **framing problems** and **collecting**, **annotating**, and **cleaning** data. — François Chollet.



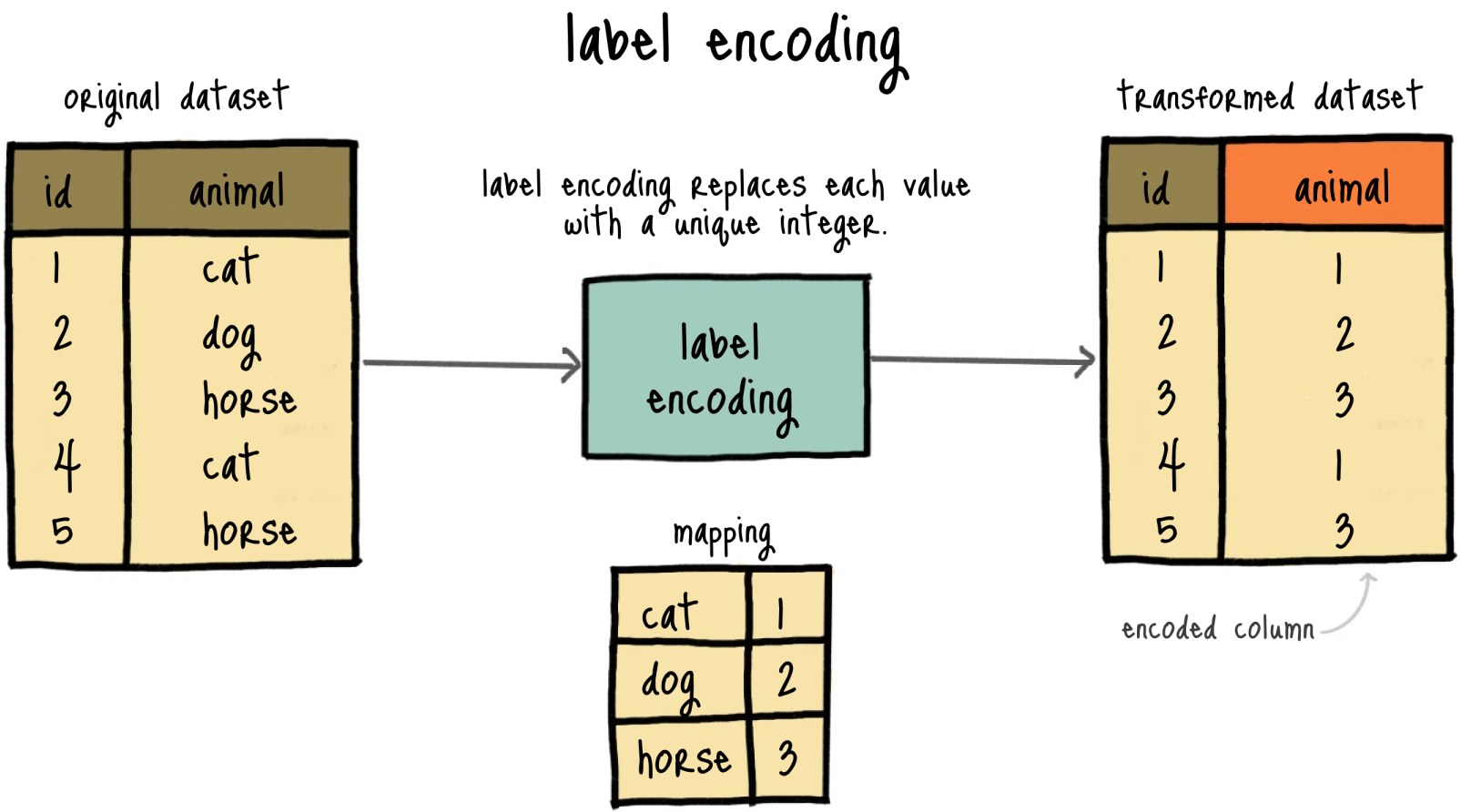
Better data produces **better models.**

The most important factor in the **quality** of a **model** is the **quality** of the **features** you use to train it.

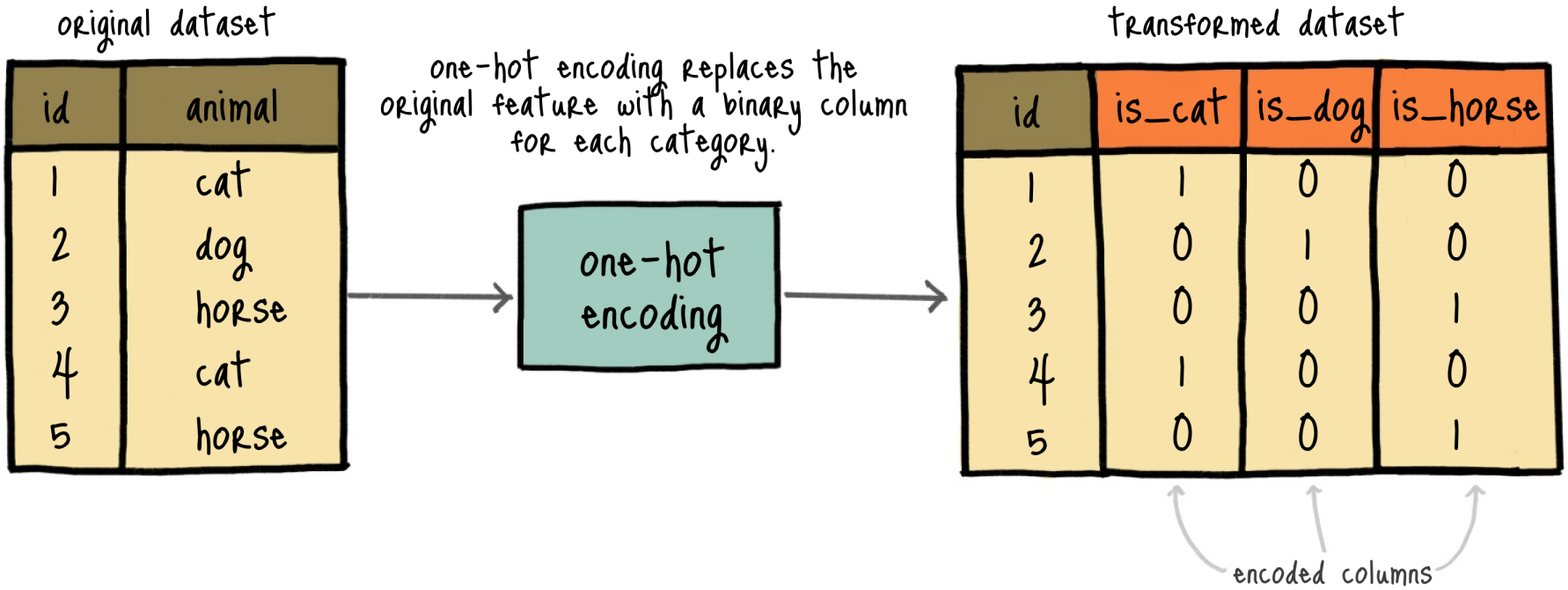


Most algorithms only work with **numerical data.**

Vectorization is the process of converting data from its original format into **numeric vectors** or **tensors.**


label encoding
original dataset
id
animal
1
cat
2
dog
3
horse
4
cat
5
horse
label encoding replaces each value with a unique integer.
label encoding
mapping
cat
1
dog
2
horse
3
transformed dataset
id
animal
1
1
2
2
3
3
4
1
5
3
encoded column

one-hot encoding



target encoding

original dataset

ip_address	target
172.16.254.3	5,200
203.0.113.45	5,385
172.16.254.3	3,799
203.0.113.45	11,500
172.16.254.3	5,230

target encoding replaces the original category with its expected target value.

target encoding

transformed dataset

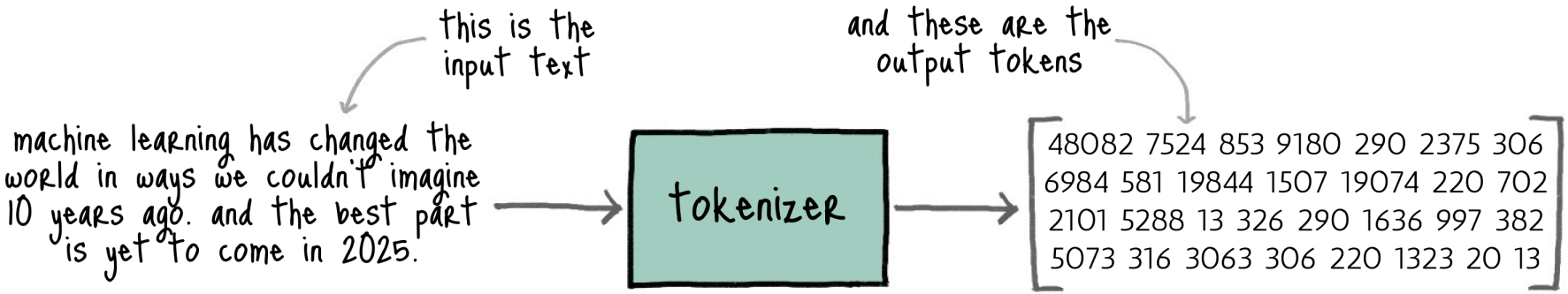
ip_address	target
4,743	5,200
8,443	5,385
4,743	3,799
8,443	11,500
4,743	5,230

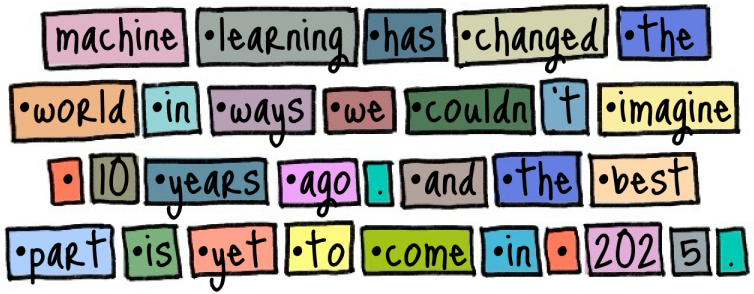
encoded column

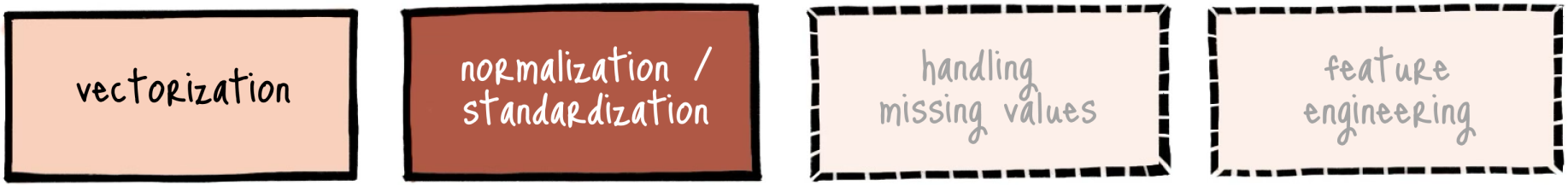
regression: expected value is the mean value for that category.

classification: expected value is the conditional probability given that category

tokenization for large language models



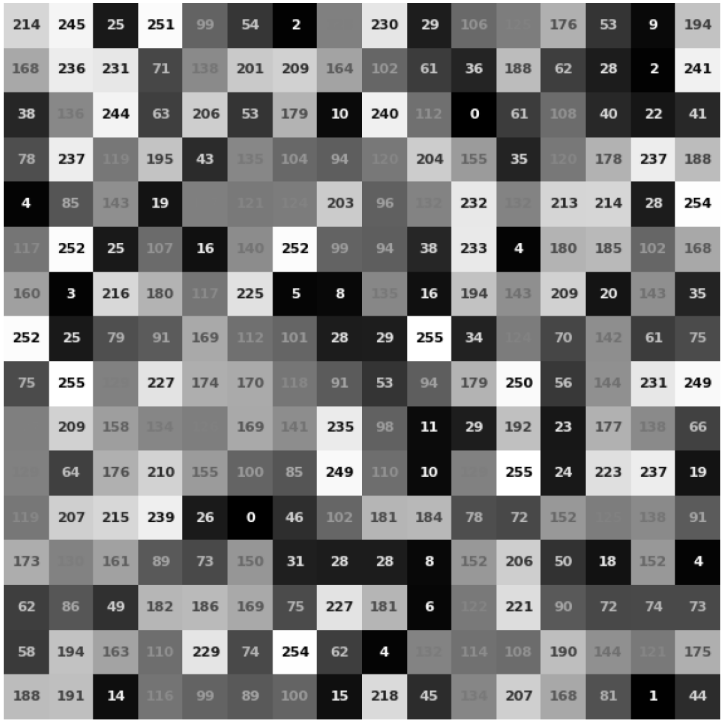




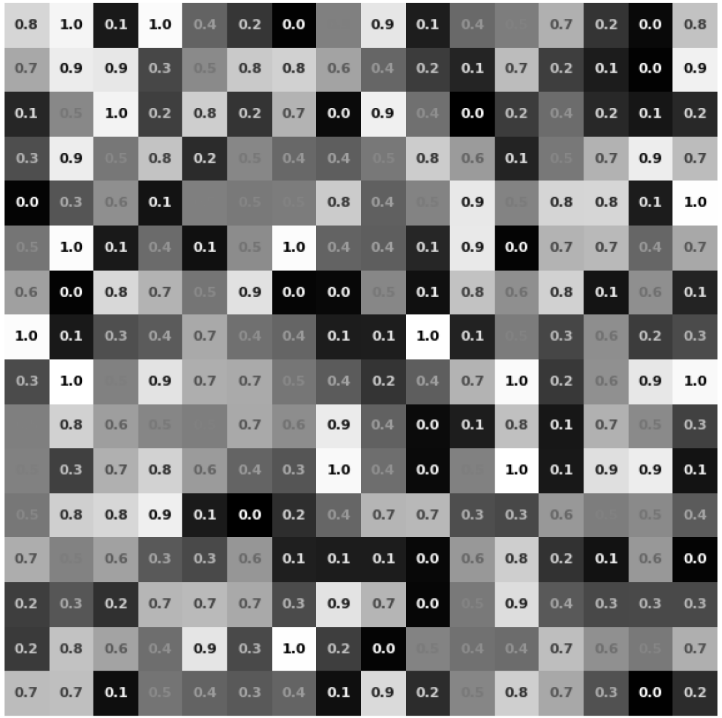
Models work best with **small**, **homogeneous** features. **Normalization** and **standardization** are techniques to scale numerical data into a similar range.

normalization

$$x' = \frac{x - min(x)}{max(x) - min(x)}$$



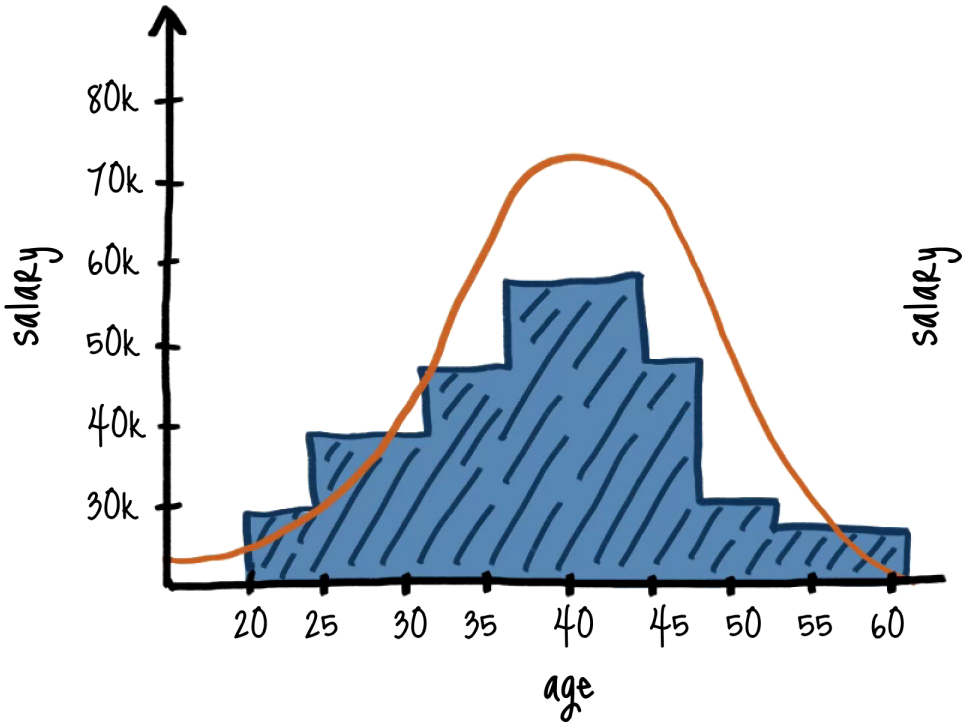
matrix of pixel values

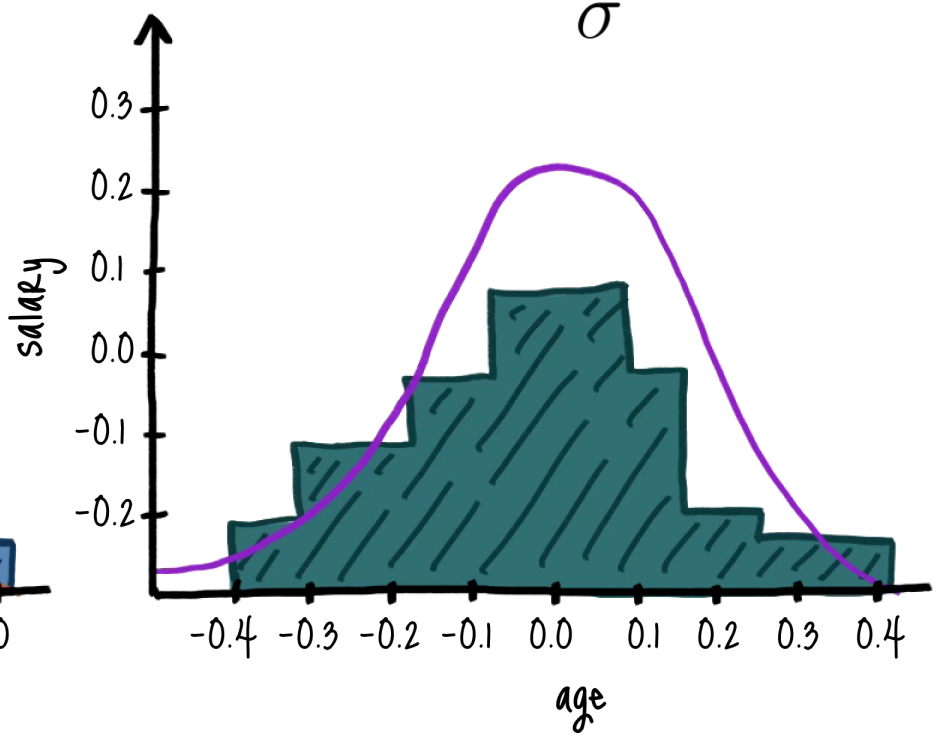


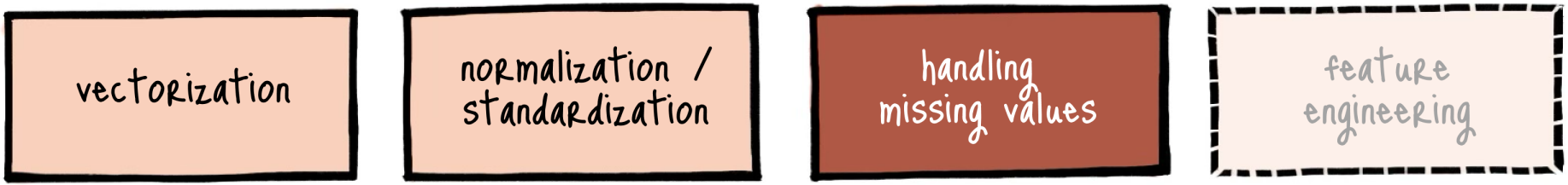
normalized image

standardization

$$x' = \frac{x - \mu}{\sigma}$$



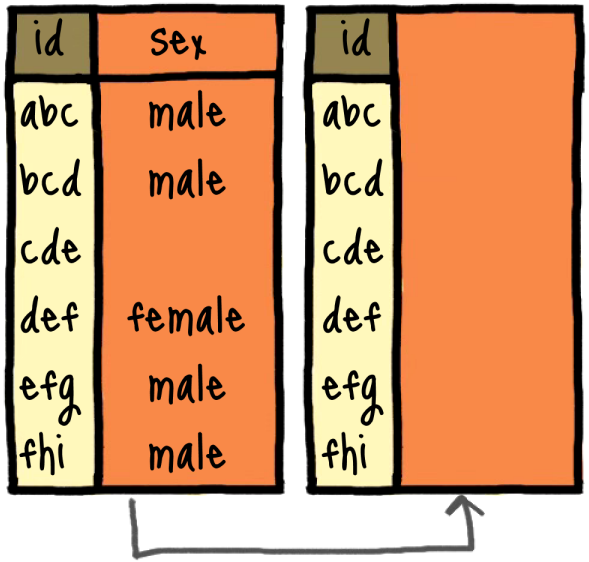




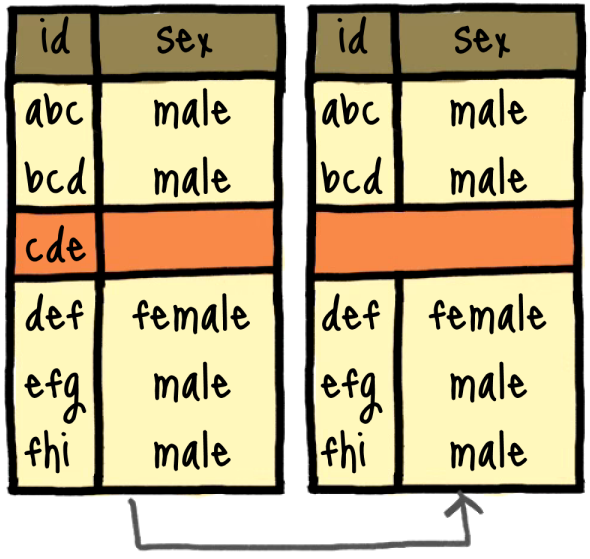
Handling **missing values** is critical to ensure the data is more **representative**, **consistent**, and **unbiased.** This improves model **accuracy** and **reliability**.

handling missing values

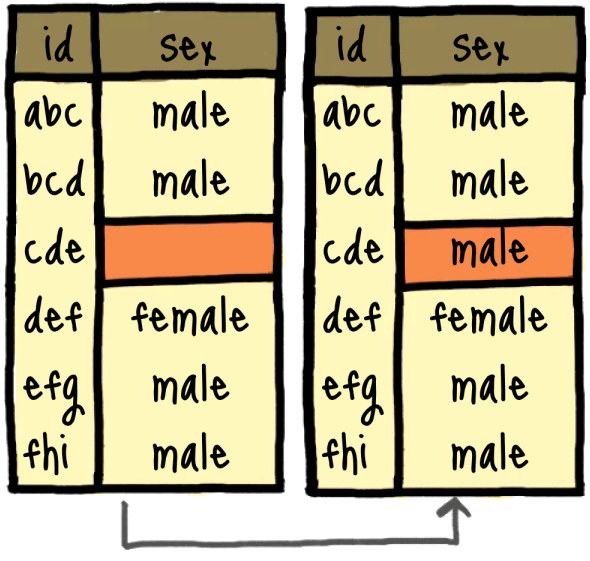
Remove the feature containing missing values.



Remove the sample containing missing values.



Replace missing values with the most frequent value.

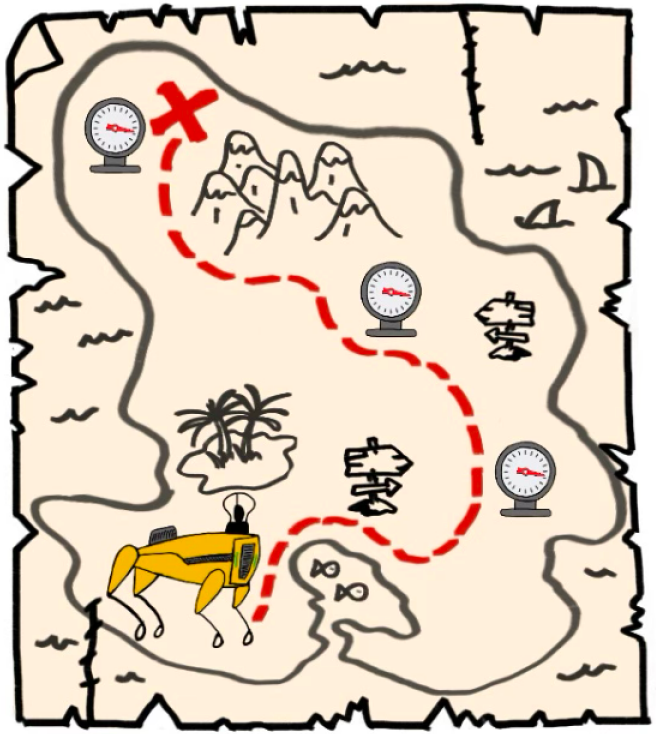


best practice

Be careful when removing missing values. **Frequency** and **distribution** patterns may reveal **key insights** about the data and its collection process.

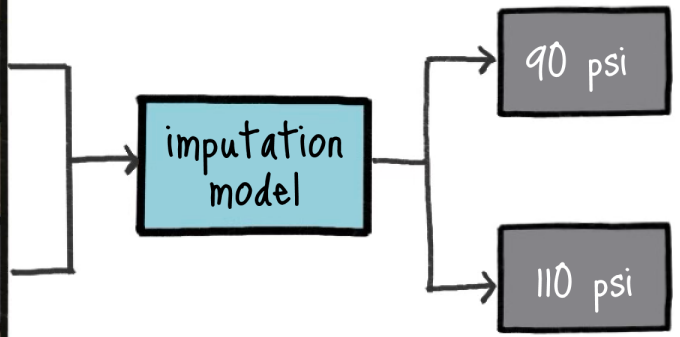
handling missing values

spot mission

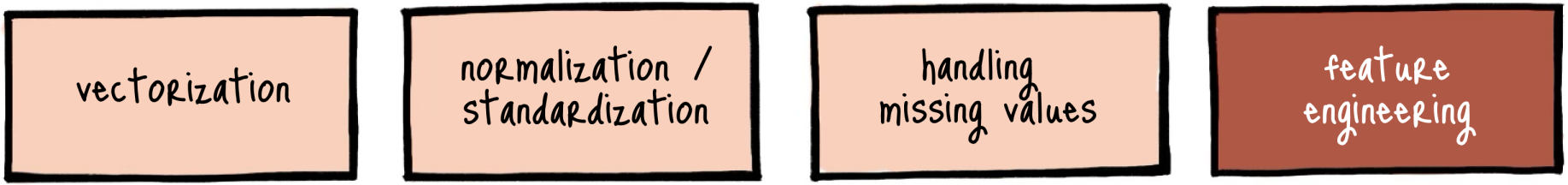


mission results

image	location	reading
	north isle	120 psi
	warehouse	
	south isle	60 psi
	warehouse	
	south isle	88 psi
	entrance	60 psi

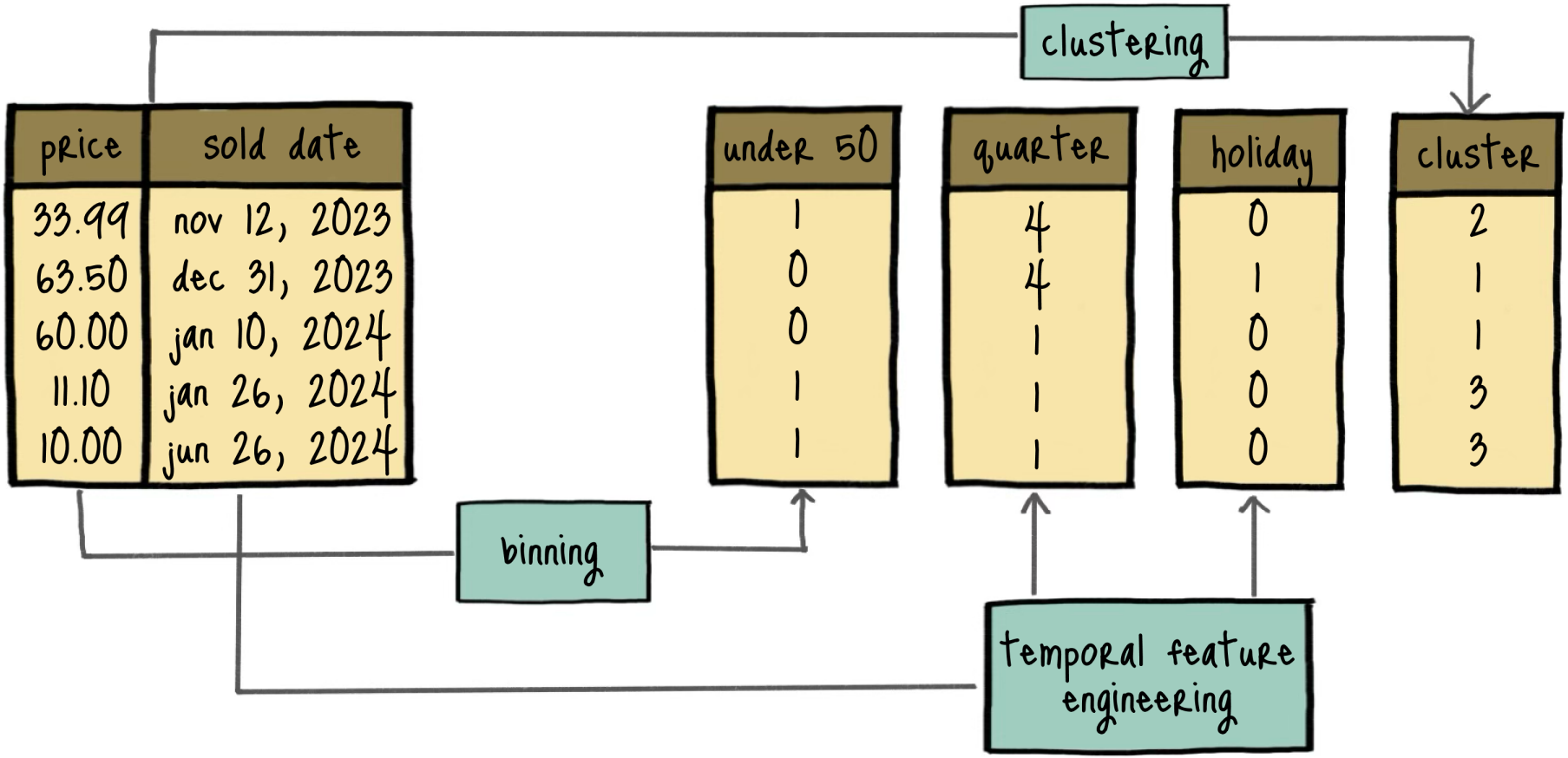


the model predicts missing values using historical data

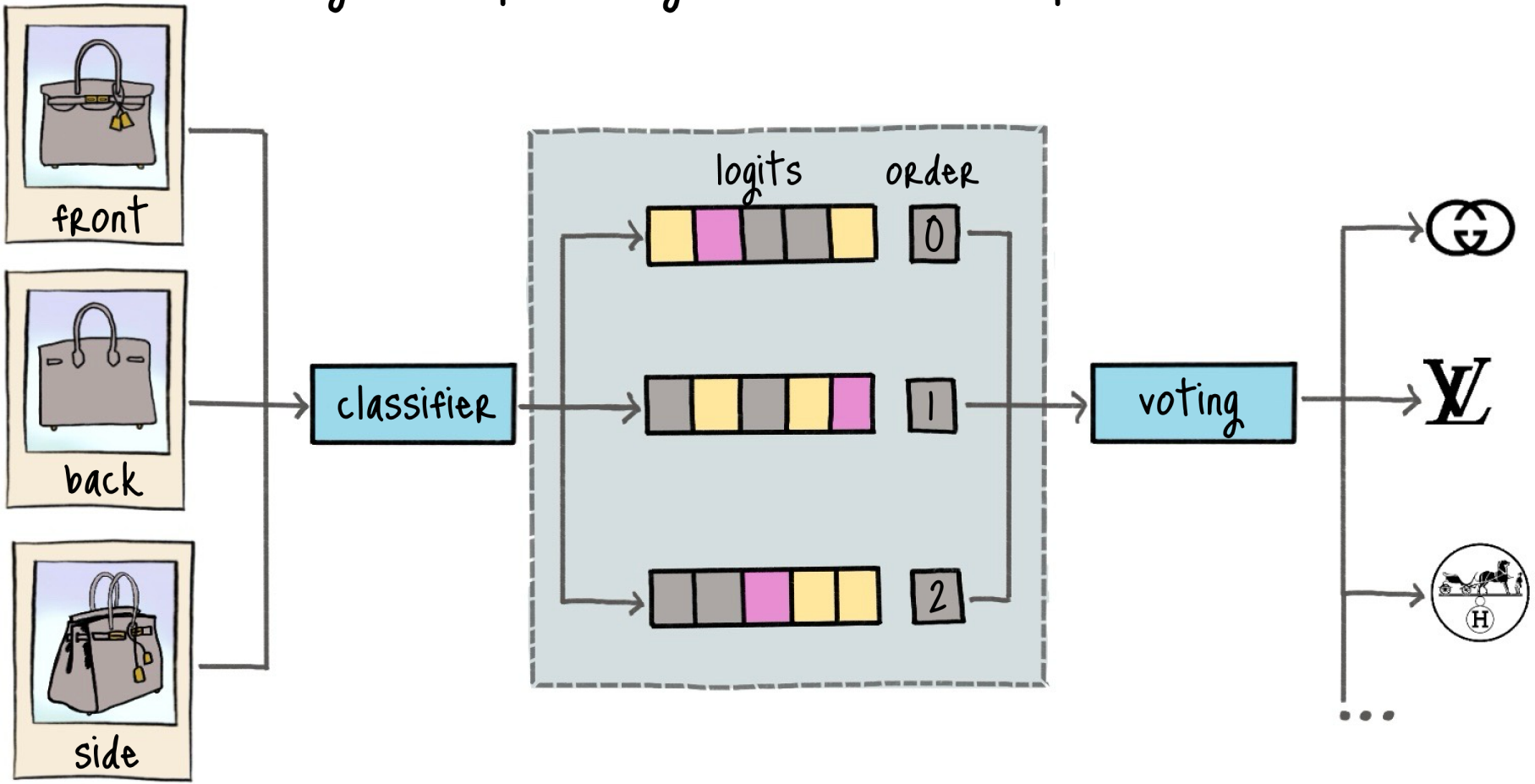


Feature engineering is the process of **transforming** the original data into **more useful**, **informative** features, helping a model **learn** more **effectively**.

feature engineering examples



using multiple images to make a prediction



best practice

The data you have is often not the data you need. **Engineering features** is one of the smartest ways to add **structure** and help your model **learn** what matters.

A **baseline** is a measuring stick to **track progress.**

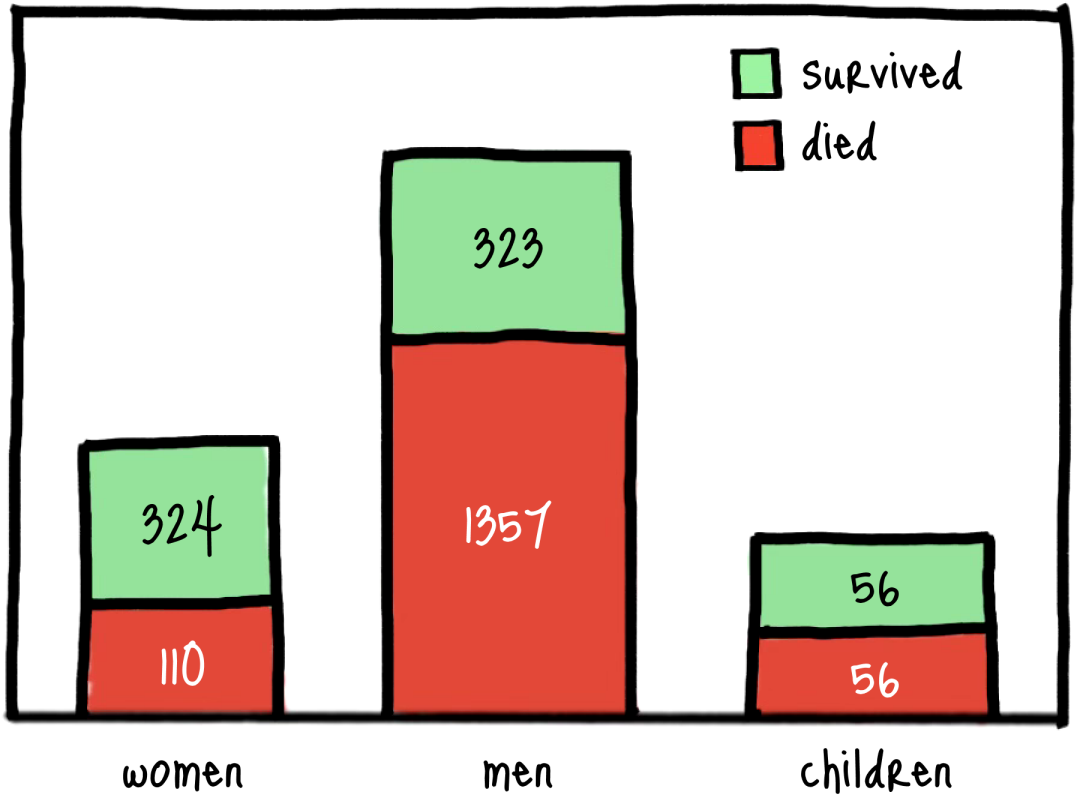
You can't tell whether a model is **good** or **useful**

unless you compare it against a **baseline.**

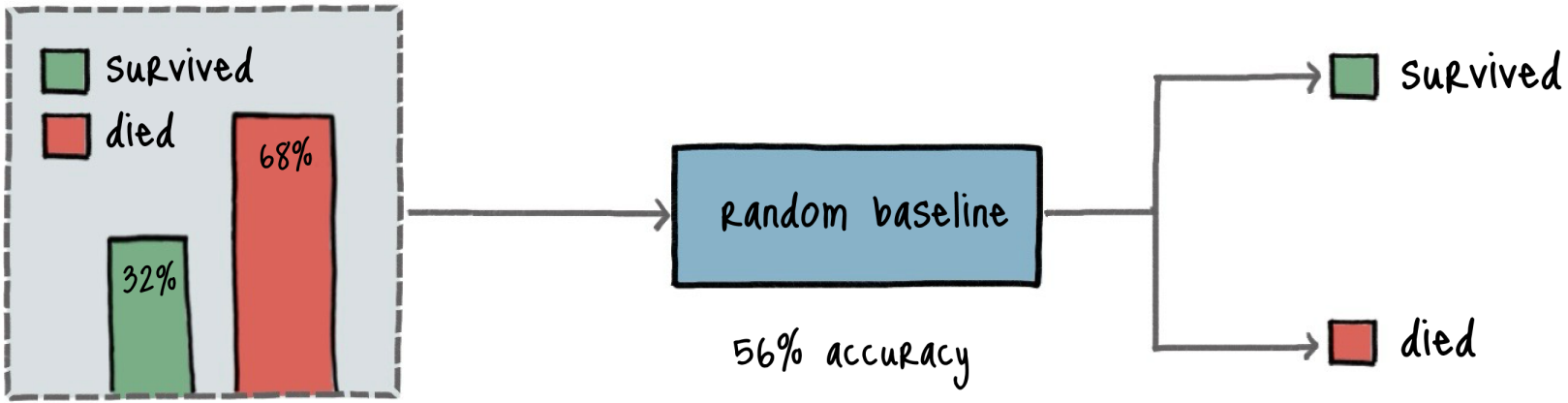
best practice

Always start with a **simple baseline**.
If your model can’t **outperform** a simple baseline,
it hasn’t learned **anything useful** yet.

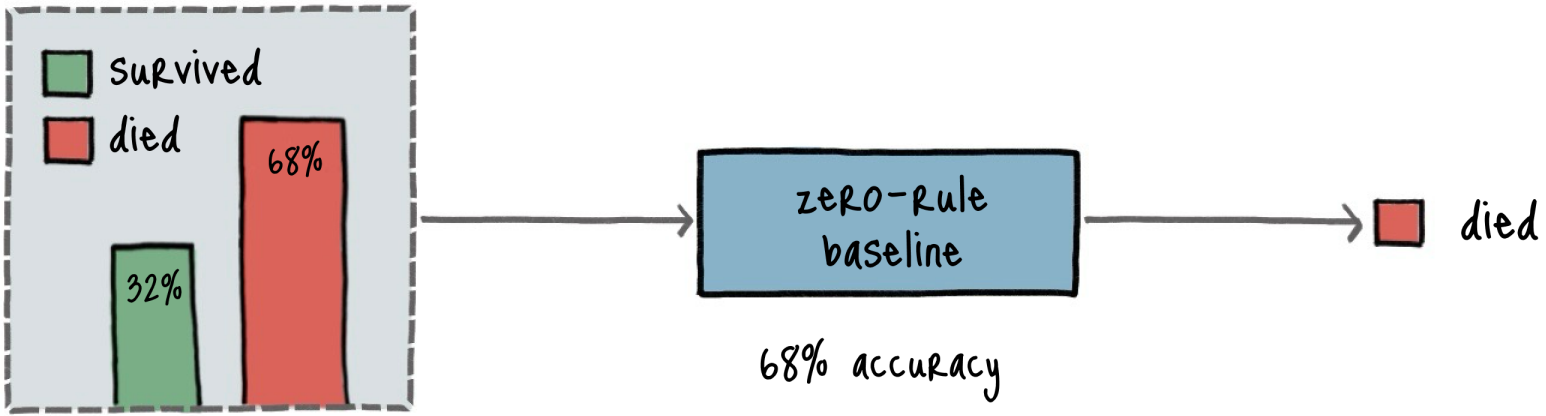
titanic's survival rate



A **random baseline** makes predictions by **chance**. It gives us a **straightforward** and **unbiased** starting point for comparison.

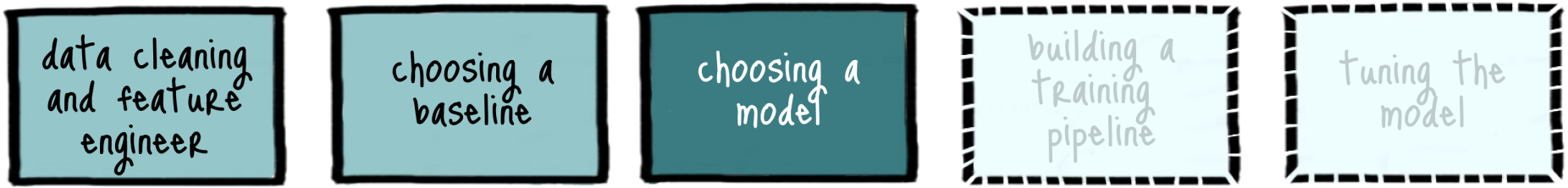


The **zero-rule algorithm** predicts the **most frequent class** (classification), or the **average** (regression). It's another simple baseline we can try to beat.



best practice

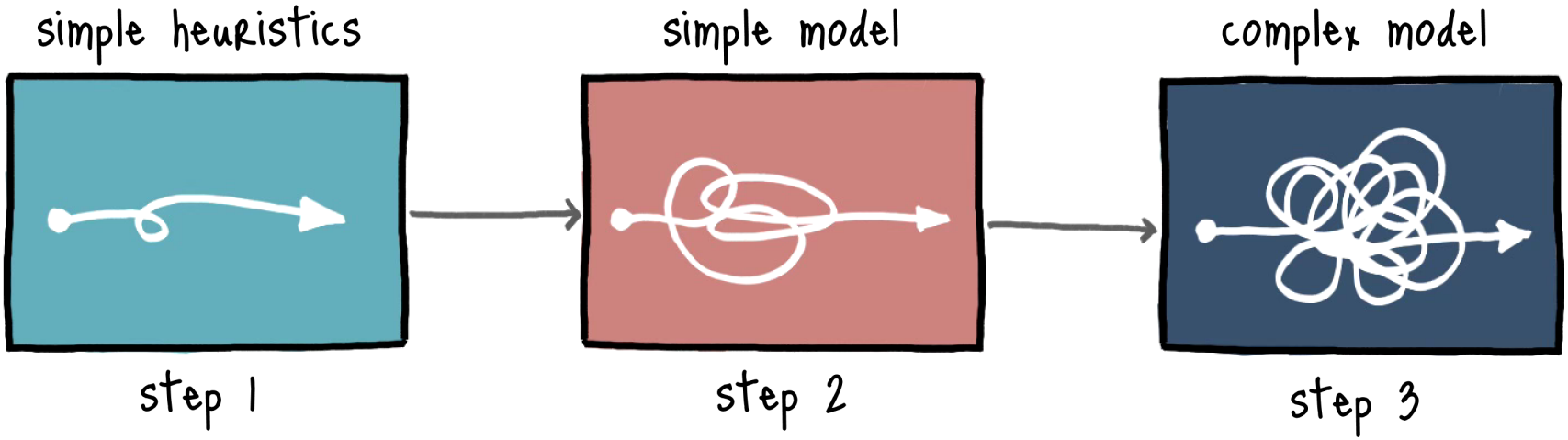
Benchmark against the **real world.**
Determine how **people** or **existing systems** do the task you want to solve and use this as your **baseline.**



Complex rule-base system become **unmaintainable.**

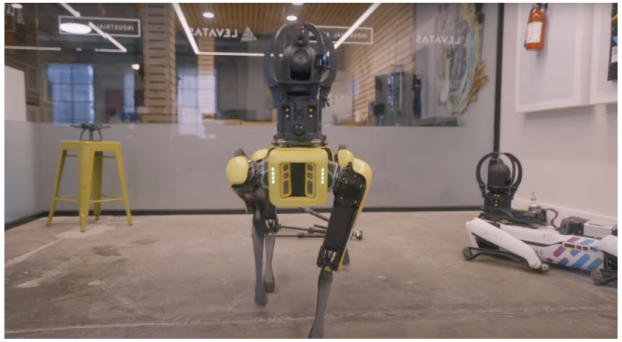
A **simple** machine learning **model** is easier to update and maintain than a **complex** set of **heuristics.**

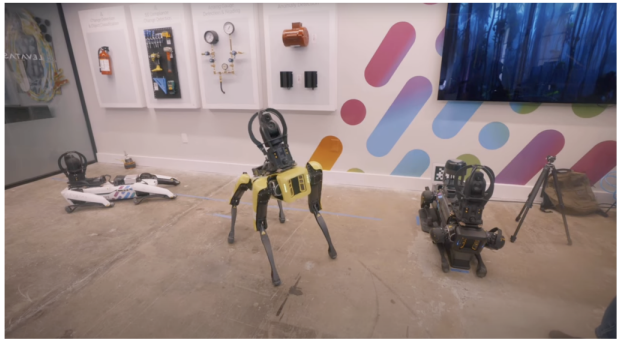
you have to earn the right to introduce complexity



additional complexity is only worth it if the extra performance makes a difference

scope creep and a simple rule-based system

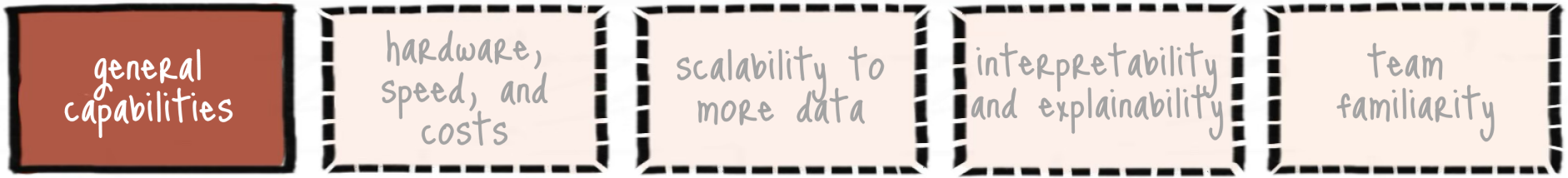





move back, go backwards, walk backwards, step back for me → → back

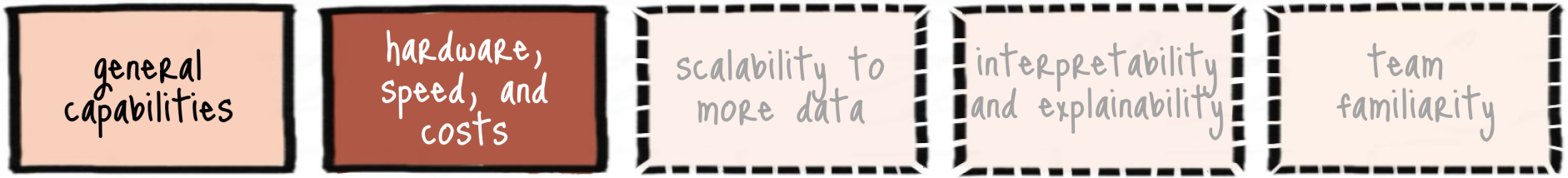
best practice

Choose the **simplest model** that could possibly work. Simple models are **easier** to **use** and **maintain**, **faster** to iterate on, and often **all you need**.



Consider what your model can and can't do.

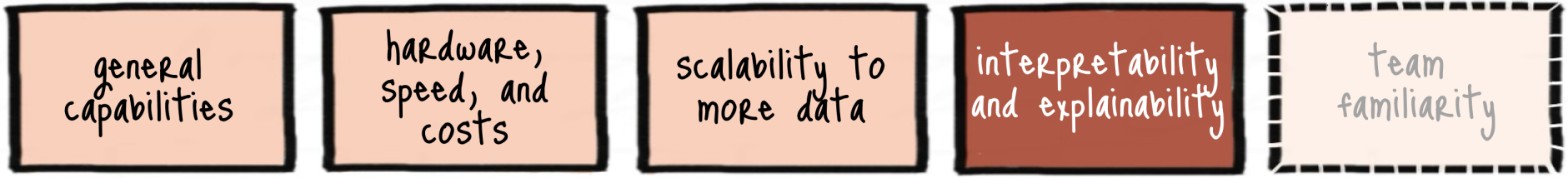
Think about the **patterns** it can capture, its **sensibility** to **noise** and **outliers**, and how well it **generalizes**.



Consider hardware, inference speed, and cost. Smaller models are **faster**, **cheaper**, and **dumber**. Large models **cost more**, run **slower**, and are more **accurate**.

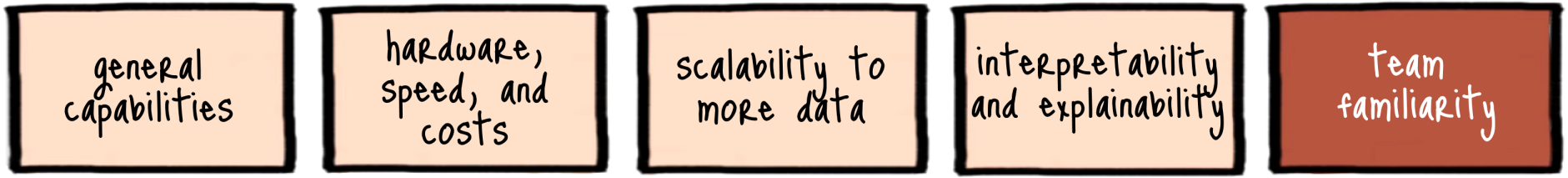
Consider how the model scales to more data.

As the **volume** and **complexity** of your data increase over time, your model will need to **scale** appropriately.



Consider black box versus white box models.

This is critical in scenarios where it's important to **understand** the **logic** behind a model's predictions.



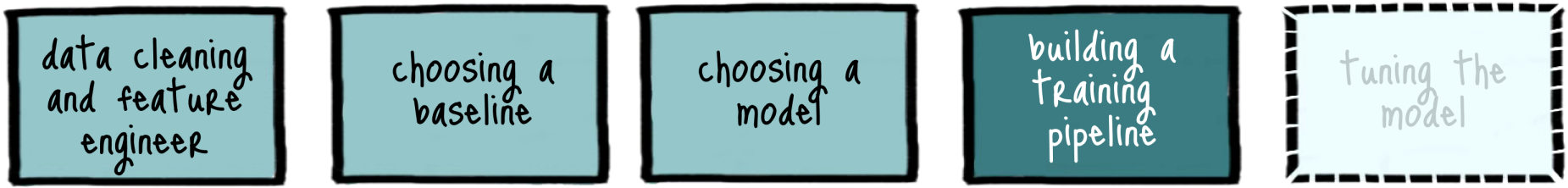
Consider how familiar is your team with the model.

Lack of technical **expertise** is a massive **roadblock.**

Prioritize using models you **know well.**

best practice

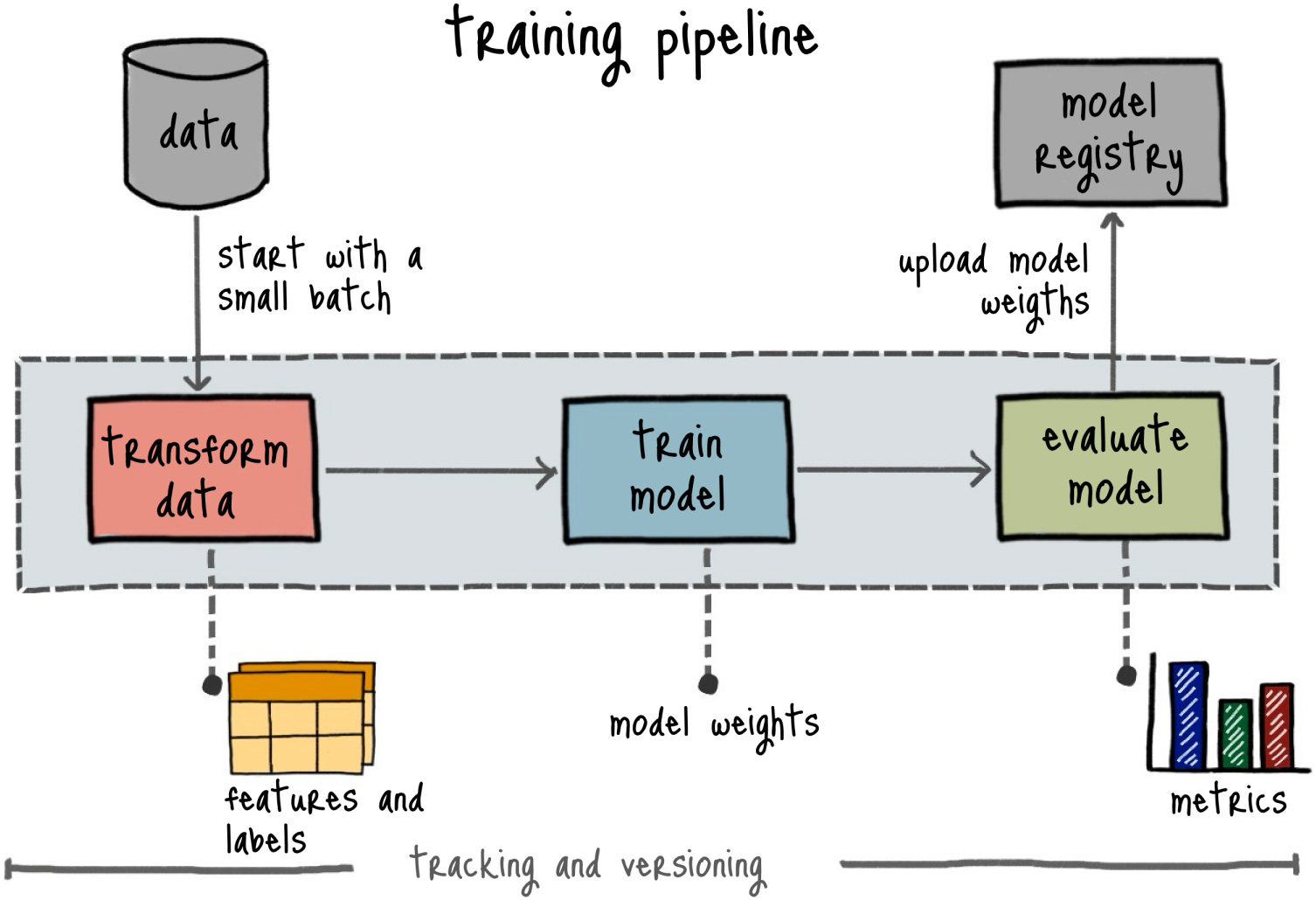
Beware of “**state-of-the-art**” models.
Many of these models are the result of a **popularity contest** and often struggle with **real-world** data.



Before iterating with the model you chose,
focus on **writing** and **wiring** the components you need
to train and evaluate a model.

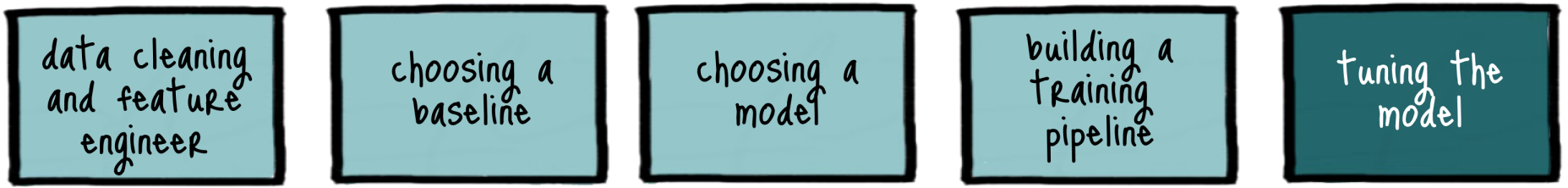
best practice

Build a working, **end-to-end** training pipeline.
Connecting all the pieces helps you **catch issues**,
measure **progress**, and get to a working solution faster.


training pipeline
data
model registry
start with a small batch
upload model weigths
transform data
train model
evaluate model
features and labels
model weights
metrics
tracking and versioning

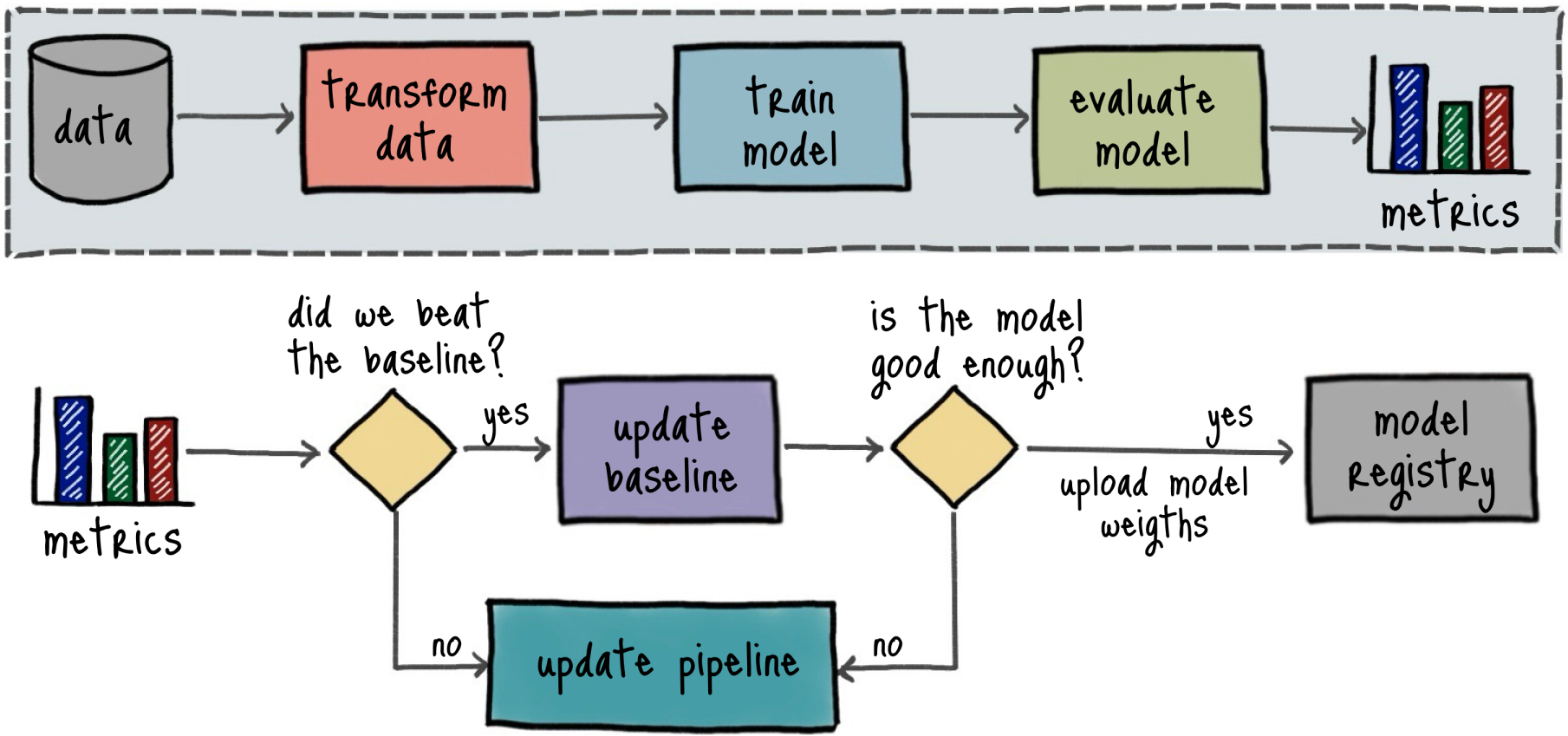
best practice

Track as many **metrics** and **logs** as you can to simplify the **debugging** and **analysis** of your pipelines. **Instrumentation** is key for building reliable pipelines.



With an end-to-end training pipeline in place, you can focus on **iteratively improving** your **data** and **tuning** the **model** until you have a **working solution.**

iterative model updates



model-centric

build the best model
to process a given dataset

- hyperparameter tuning
- implementing ensemble models
- using custom loss functions
- transfer learning
- using regularization techniques
- modifying model architecture

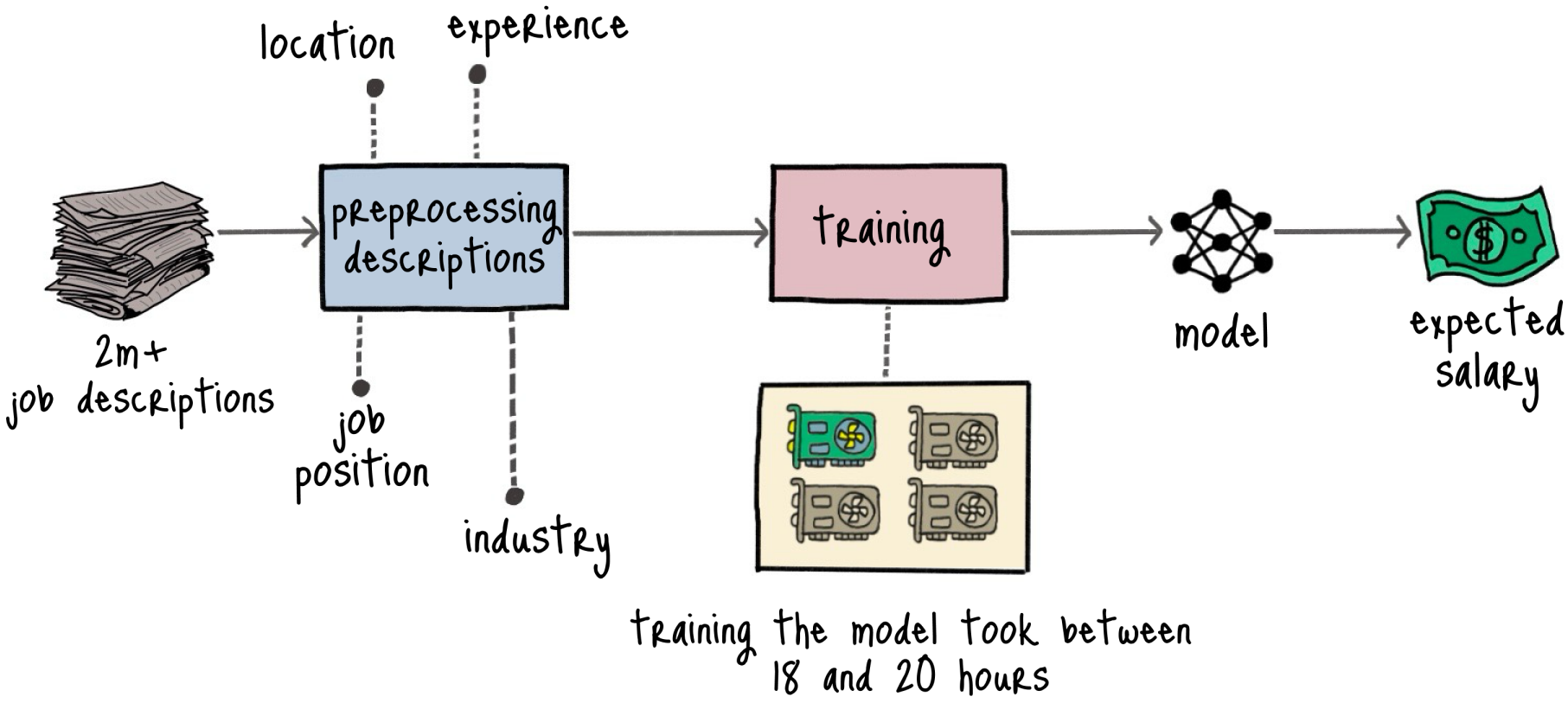
data-centric

produce the best dataset
to feed a given model

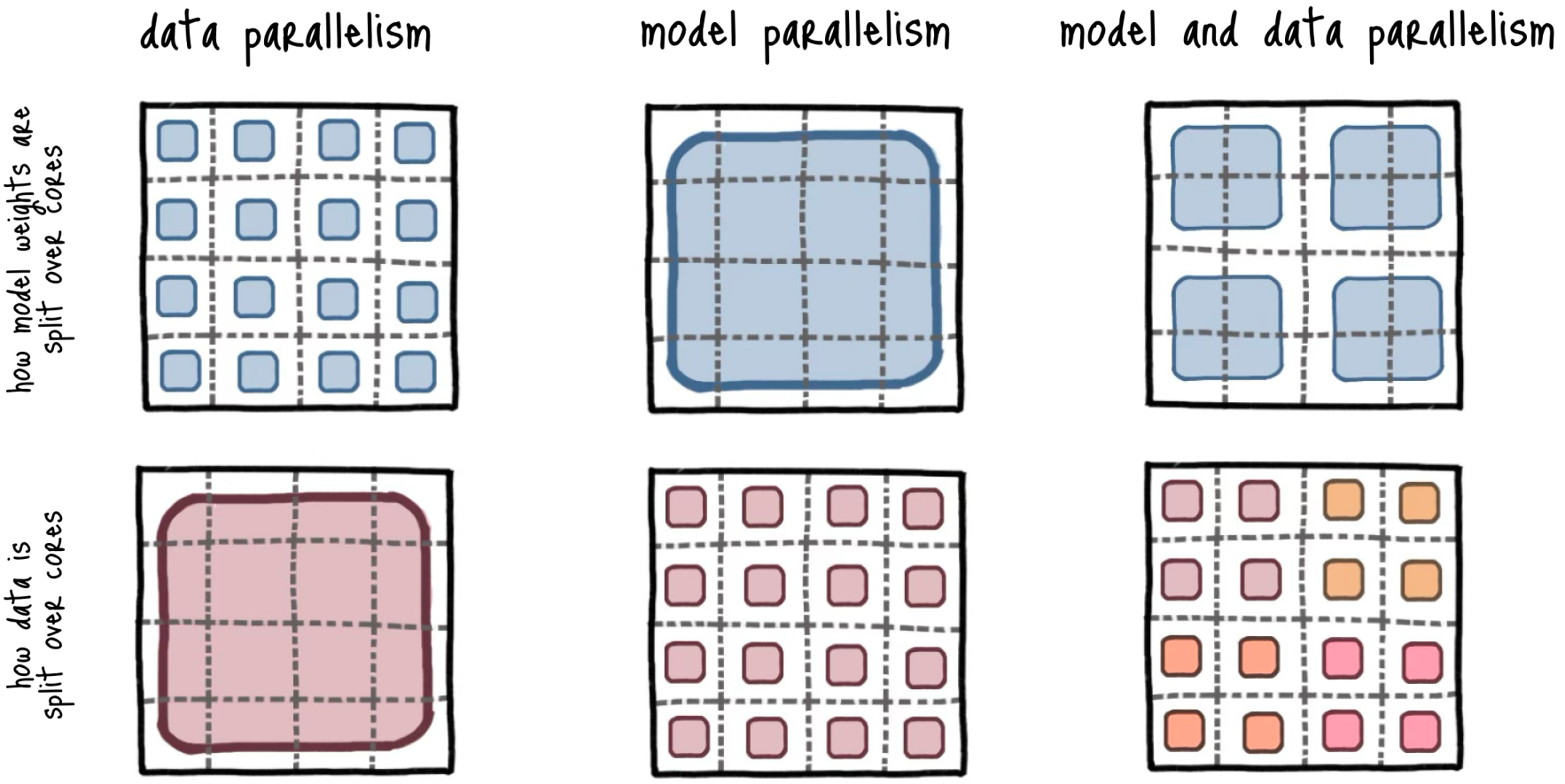
- feature engineering
- improving labels
- data augmentation
- balancing dataset
- synthetic data generation
- collecting additional data

Distributed training offers scalability, faster training time, and improved resource utilization when handling **large datasets** and **complex models**.

processing job descriptions



Data parallelism and **model parallelism** (with **pipeline execution**) are the two **distributed training** techniques used in production applications.


data parallelism
model parallelism
model and data parallelism
how model weights are split over cores
how data is split over cores

the end