

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.

data cleaning
and feature
engineering

choosing a
baseline

choosing a
model

building a
training
pipeline

tuning the
model

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.

vectorization

normalization /
standardization

handling
missing values

feature
engineering

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

original dataset

id	animal
1	cat
2	dog
3	horse
4	cat
5	horse

one-hot encoding replaces the original feature with a binary column for each category.

one-hot
encoding

transformed dataset

id	is_cat	is_dog	is_horse
1	1	0	0
2	0	1	0
3	0	0	1
4	1	0	0
5	0	0	1

encoded columns

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

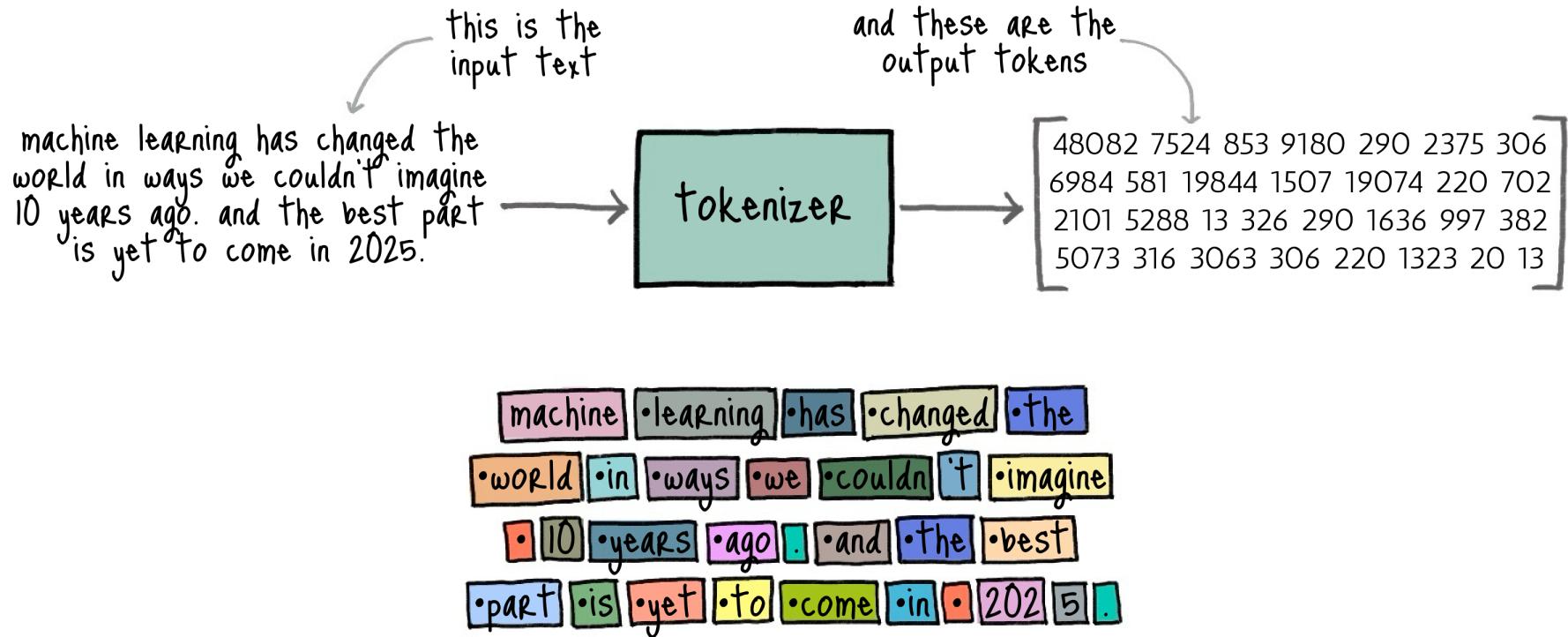
ip-address	target
4,743	5,200
8,443	5,385
4,743	3,799
8,443	11,500
4,743	5,230

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



vectorization

normalization /
standardization

handling
missing values

feature
engineering

Models work best with **small, homogeneous** features.

Normalization and **standardization** are techniques to scale numerical data into a similar range.

normalization

214	245	25	251	99	54	2		230	29	106	133	176	53	9	194
168	236	231	71	138	201	209	164	102	61	36	188	62	28	2	241
38	136	244	63	206	53	179	10	240	112	0	61	108	40	22	41
78	237	119	195	43	135	104	94	120	204	155	35	120	178	237	188
4	85	143	19		121	124	203	96	132	232	132	213	214	28	254
117	252	25	107	16	140	252	99	94	38	233	4	180	185	102	168
160	3	216	180	117	225	5	8	135	16	194	143	209	20	143	35
252	25	79	91	169	112	101	28	29	255	34	124	70	142	61	75
75	255		227	174	170	118	91	53	94	179	250	56	144	231	249
209	158	134		169	141	235	98	11	29	192	23	177	138	66	
64	176	210	155	100	85	249	110	10		255	24	223	237	19	
119	207	215	239	26	0	46	102	181	184	78	72	152	125	138	91
173	130	161	89	73	150	31	28	28	8	152	206	50	18	152	4
62	86	49	182	186	169	75	227	181	6	122	221	90	72	74	73
58	194	163	110	229	74	254	62	4	132	114	108	190	144	121	175
188	191	14	116	99	89	100	15	218	45	134	207	168	81	1	44

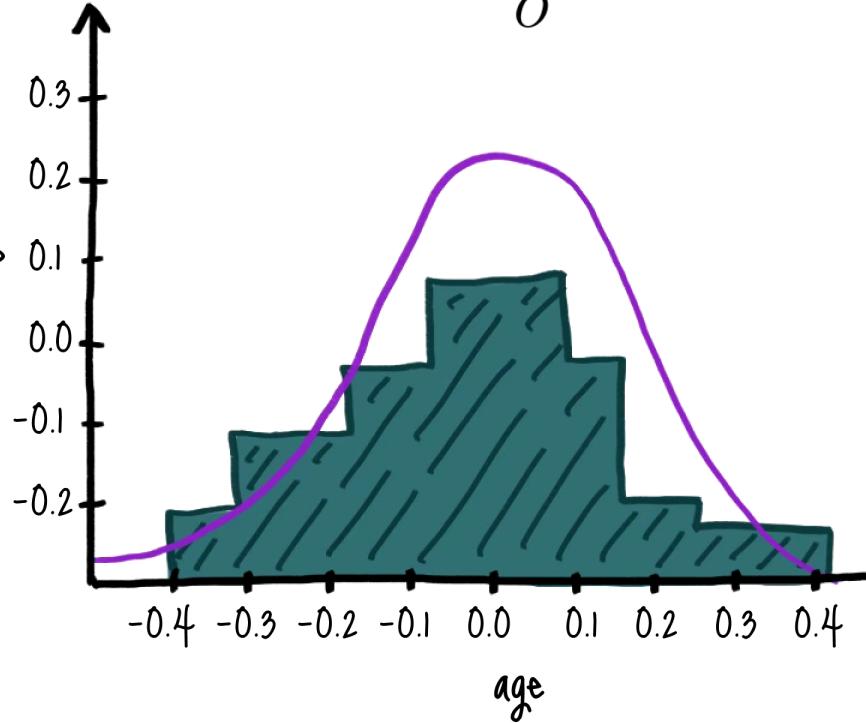
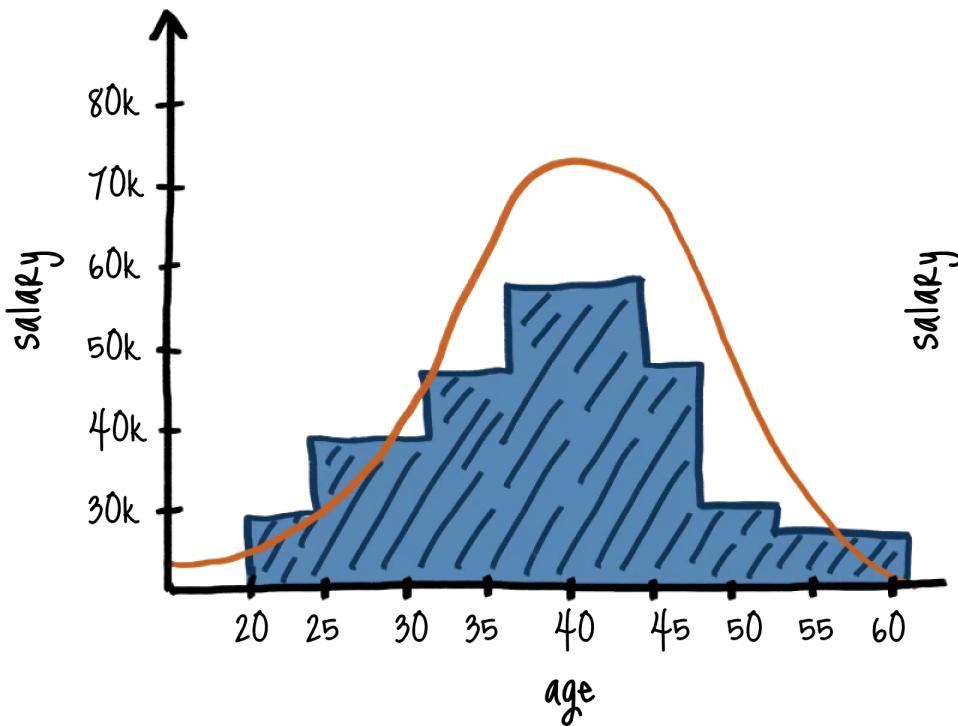
matrix of pixel values



0.8	1.0	0.1	1.0	0.4	0.2	0.0		0.9	0.1	0.4	0.5	0.7	0.2	0.0	0.8
0.7	0.9	0.9	0.3	0.5	0.8	0.8	0.6	0.4	0.2	0.1	0.7	0.2	0.1	0.0	0.9
0.1	0.5	1.0	0.2	0.8	0.2	0.7	0.0	0.9	0.4	0.0	0.2	0.4	0.2	0.1	0.2
0.3	0.9	0.5	0.8	0.2	0.5	0.4	0.4	0.5	0.8	0.6	0.1	0.5	0.7	0.9	0.7
0.0	0.3	0.6	0.1		0.5	0.5	0.8	0.4	0.5	0.9	0.5	0.8	0.8	0.1	1.0
0.5	1.0	0.1	0.4	0.1	0.5	1.0	0.4	0.4	0.1	0.9	0.0	0.7	0.7	0.4	0.7
0.6	0.0	0.8	0.7	0.5	0.9	0.0	0.0	0.5	0.1	0.8	0.6	0.8	0.1	0.6	0.1
1.0	0.1	0.3	0.4	0.7	0.4	0.4	0.1	0.1	1.0	0.1	0.5	0.3	0.6	0.2	0.3
0.3	1.0		0.9	0.7	0.7	0.5	0.4	0.2	0.4	0.7	1.0	0.2	0.6	0.9	1.0
	0.8	0.6	0.5		0.7	0.6	0.9	0.4	0.0	0.1	0.8	0.1	0.7	0.5	0.3
0.3	0.7	0.8	0.6	0.4	0.3	1.0	0.4	0.0		1.0	0.1	0.9	0.9	0.1	
0.5	0.8	0.8	0.9	0.1	0.0	0.2	0.4	0.7	0.7	0.3	0.3	0.6	0.5	0.4	
0.7	0.3	0.6	0.3	0.3	0.6	0.1	0.1	0.1	0.0	0.6	0.8	0.2	0.1	0.6	0.0
0.2	0.3	0.2	0.7	0.7	0.7	0.3	0.9	0.7	0.0	0.5	0.9	0.4	0.3	0.3	0.3
0.2	0.8	0.6	0.4	0.9	0.3	1.0	0.2	0.0	0.5	0.4	0.4	0.7	0.6	0.5	0.7
0.7	0.7	0.1	0.5	0.4	0.3	0.4	0.1	0.9	0.2	0.5	0.8	0.7	0.3	0.0	0.2

normalized image

standardization



vectorization

normalization /
standardization

handling
missing values

feature
engineering

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.

id	sex
abc	male
bcd	male
cde	
def	female
efg	male
fhi	male

id	sex
abc	
bcd	
cde	
def	
efg	
fhi	

Remove the sample containing missing values.

id	sex
abc	male
bcd	male
cde	
def	female
efg	male
fhi	male

id	sex
abc	male
bcd	male
cde	
def	female
efg	
fhi	male

Replace missing values with the most frequent value.

id	sex
abc	male
bcd	male
cde	male
def	female
efg	male
fhi	male

id	sex
abc	male
bcd	male
cde	male
def	female
efg	male
fhi	male

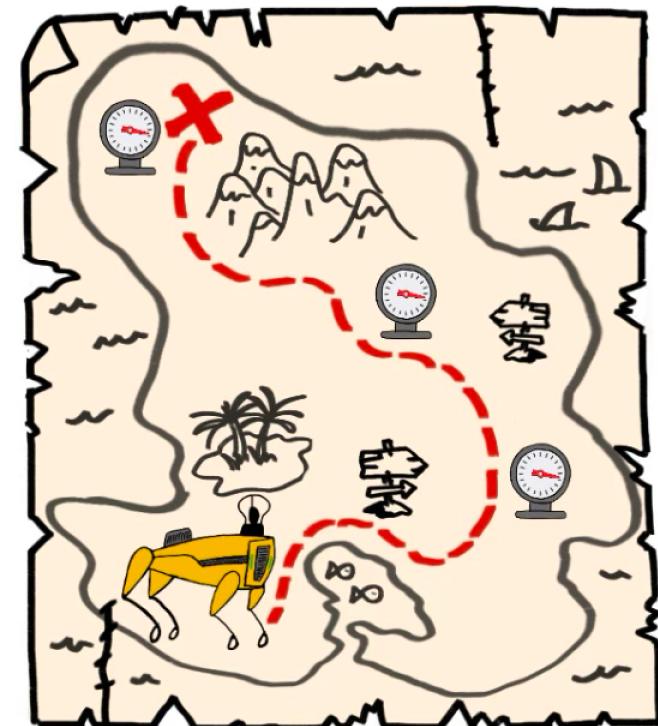
best practice

Be careful when removing missing values.

Frequency and **distribution** patterns may reveal **key insights** about the data and its collection process.

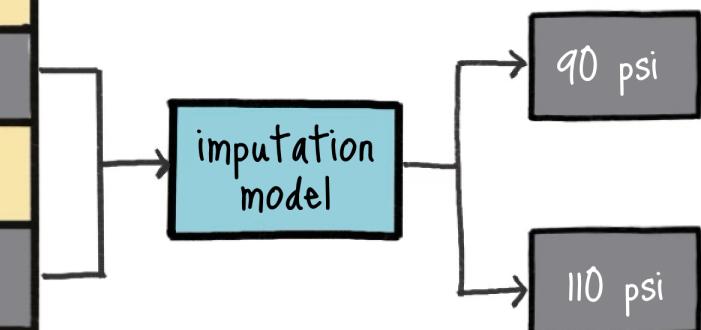
handling missing values

spot mission



mission results

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



the model predicts missing values using historical data

vectorization

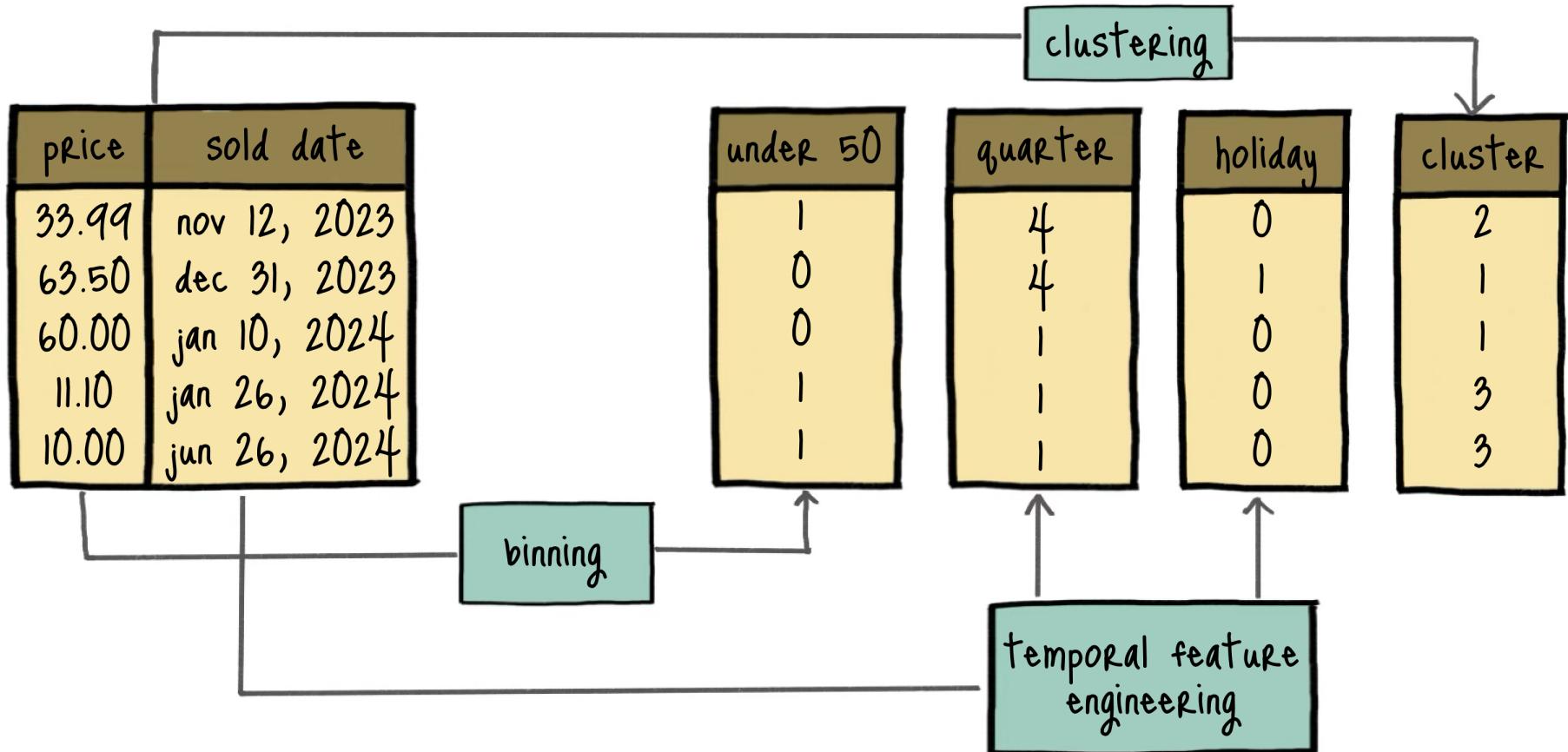
normalization /
standardization

handling
missing values

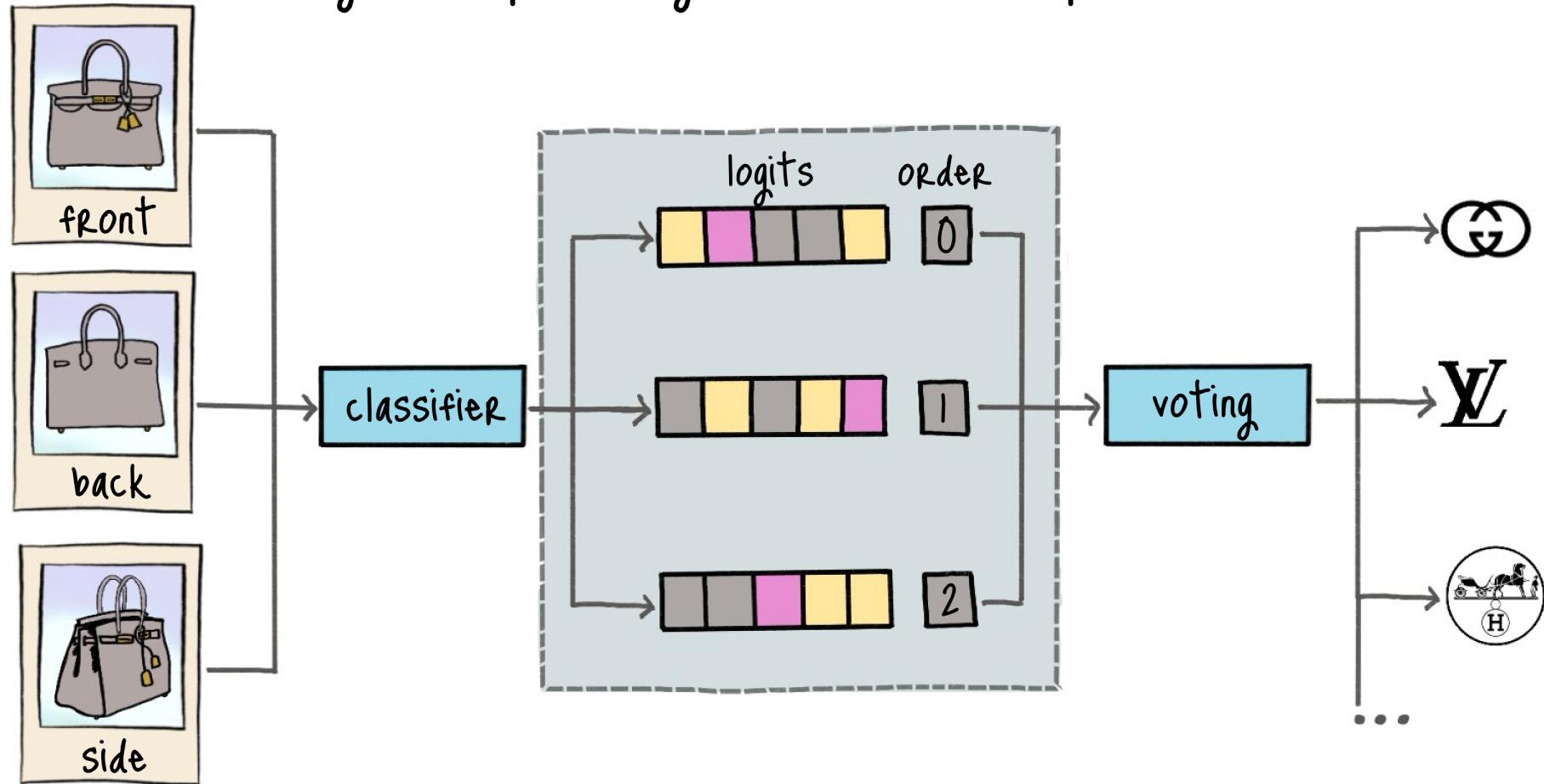
feature
engineering

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.

data cleaning
and feature
engineering

choosing a
baseline

choosing a
model

building a
training
pipeline

tuning the
model

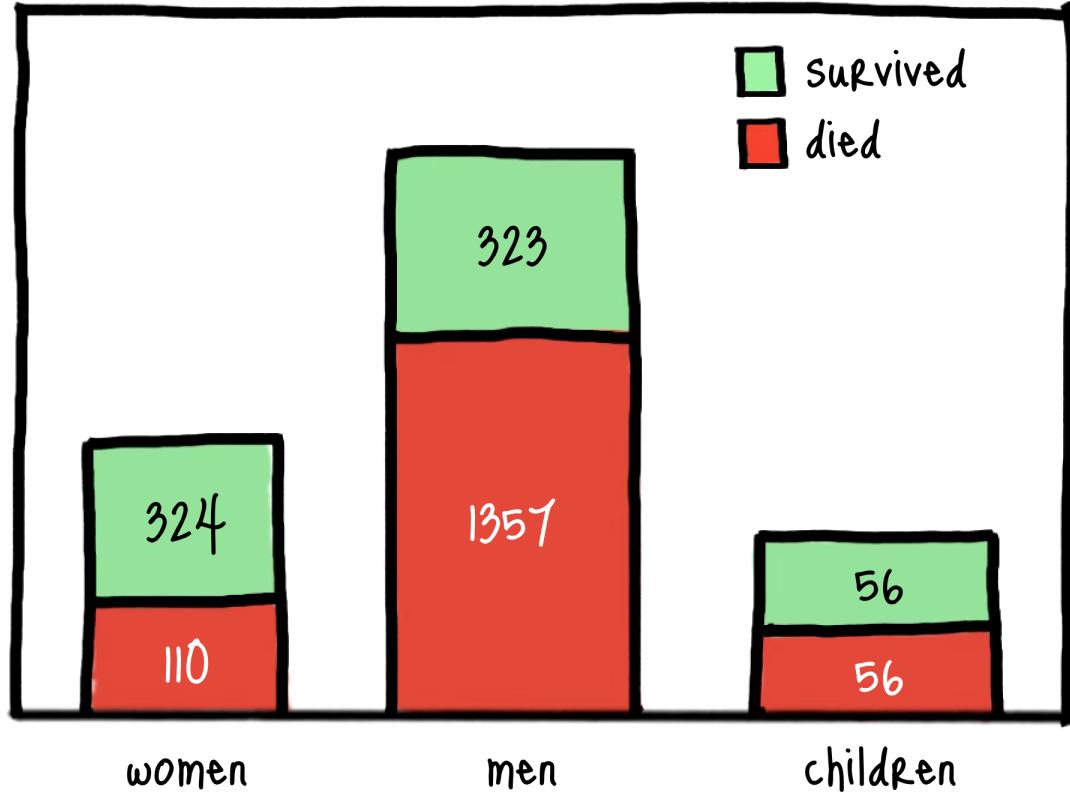
A **baseline** is a measuring stick to **track progress**.

You can't tell whether a model is **good** or **useful**
unless you compare it against a **baseline**.

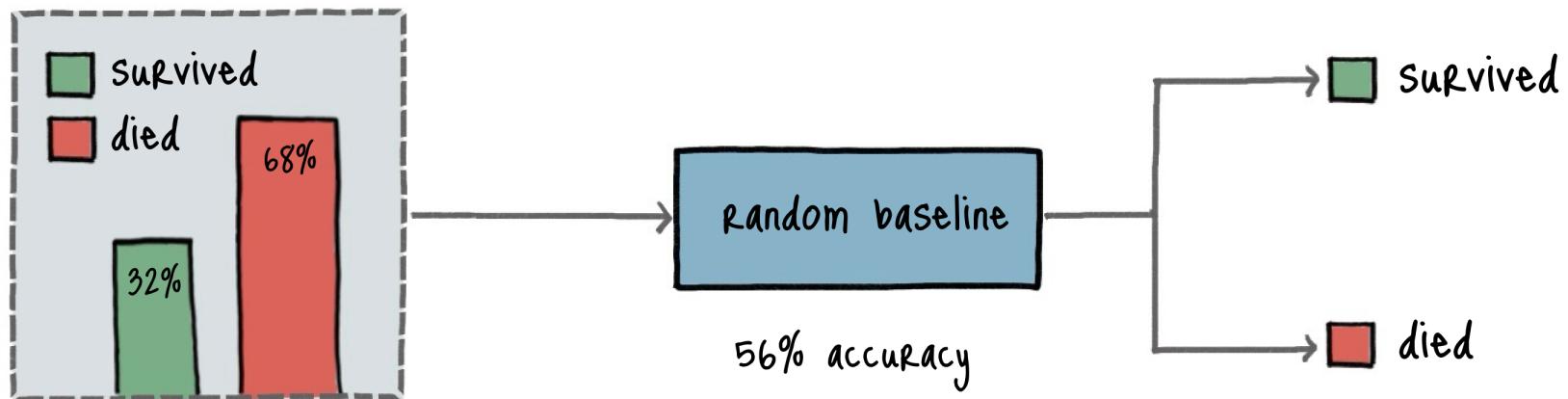
best practice

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

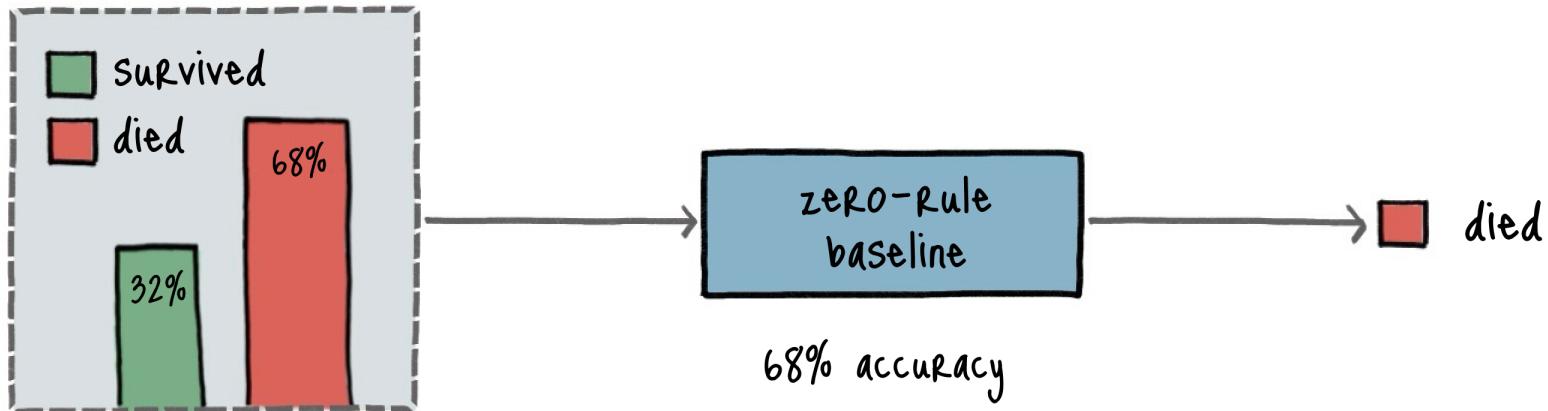


A **random baseline** makes predictions by **chance**.
It gives us a **straightforward** and **unbiased**
starting point for comparison.



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**.

data cleaning
and feature
engineering

choosing a
baseline

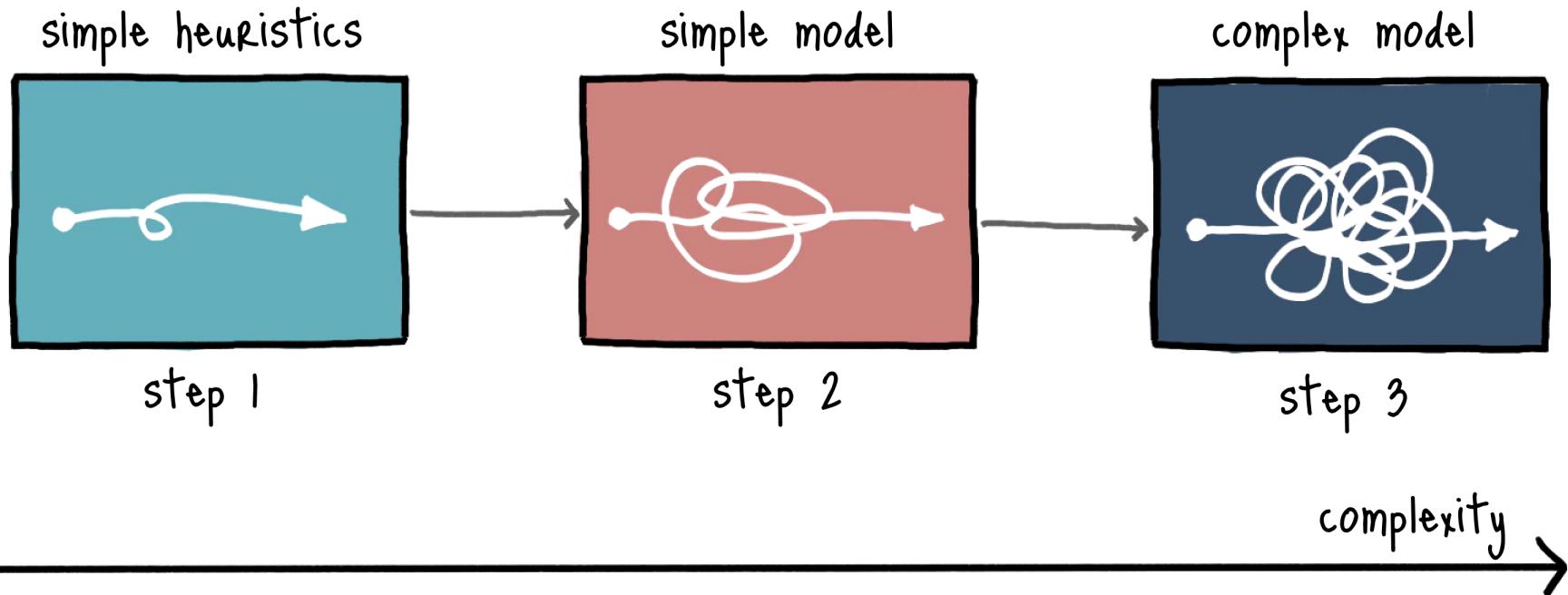
choosing a
model

building a
training
pipeline

tuning the
model

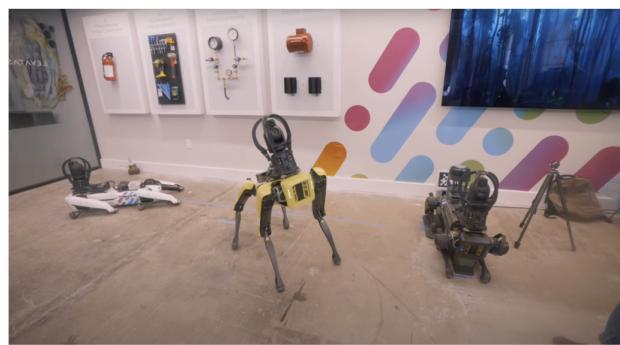
Complex rule-base system become **unmaintainable**.
A **simple** machine learning **model** is easier to update
and maintain than a **complex** set of **heuristics**.

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.

general
capabilities

hardware,
speed, and
costs

scalability to
more data

interpretability
and explainability

team
familiarity

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**.

general
capabilities

hardware,
speed, and
costs

scalability to
more data

interpretability
and explainability

team
familiarity

Consider hardware, inference speed, and cost.

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

general
capabilities

hardware,
speed, and
costs

scalability to
more data

interpretability
and explainability

team
familiarity

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.

general
capabilities

hardware,
speed, and
costs

scalability to
more data

interpretability
and explainability

team
familiarity

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.

general
capabilities

hardware,
speed, and
costs

scalability to
more data

interpretability
and explainability

team
familiarity

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.

data cleaning
and feature
engineering

choosing a
baseline

choosing a
model

building a
training
pipeline

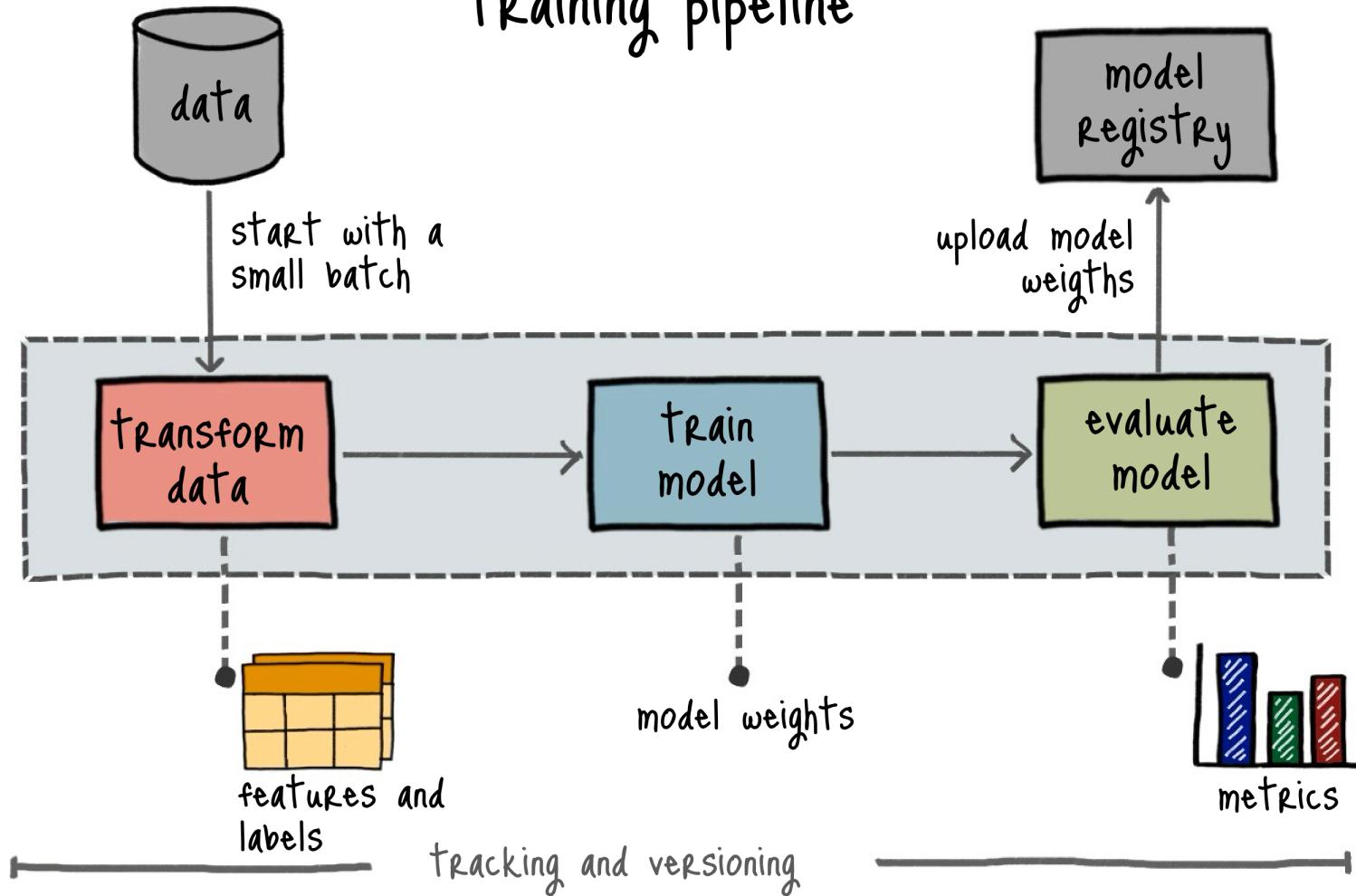
tuning the
model

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



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.

data cleaning
and feature
engineering

choosing a
baseline

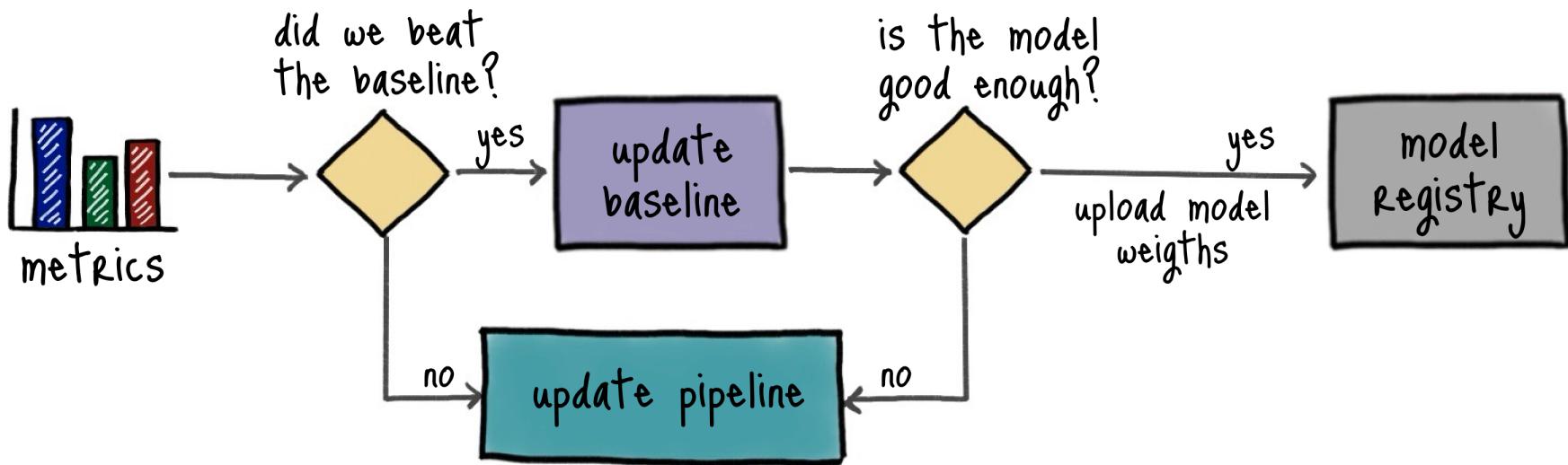
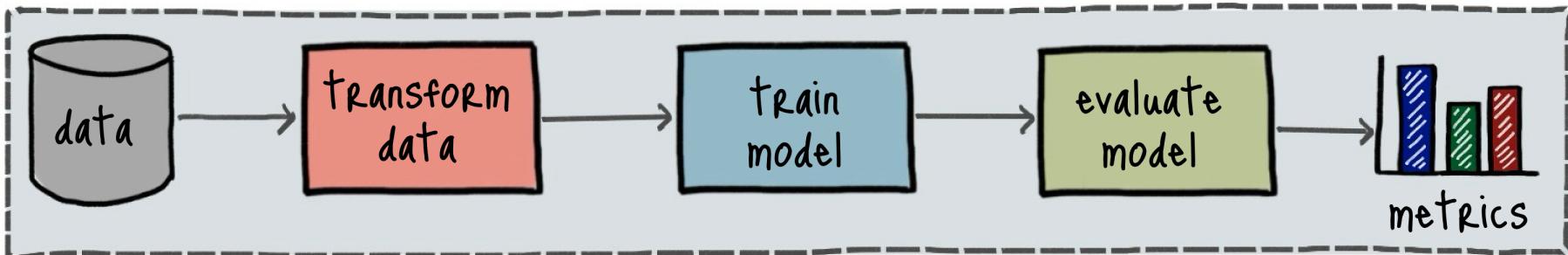
choosing a
model

building a
training
pipeline

tuning the
model

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

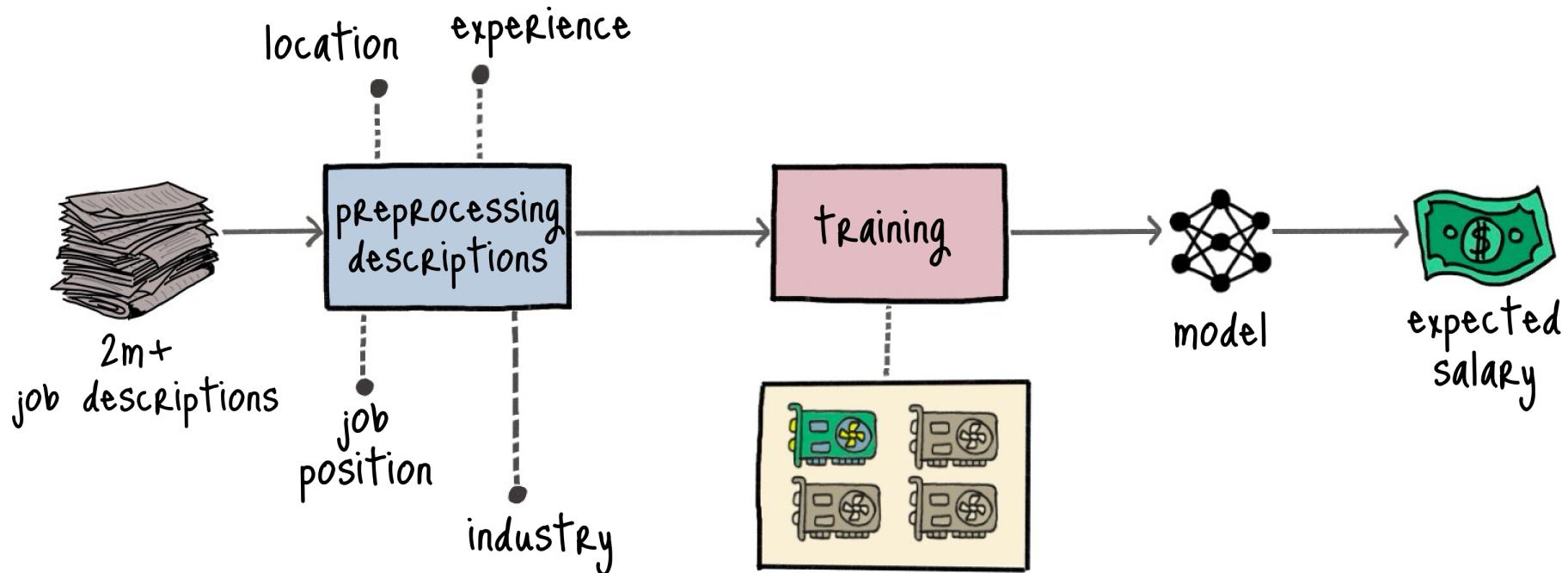
data-centric

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**.

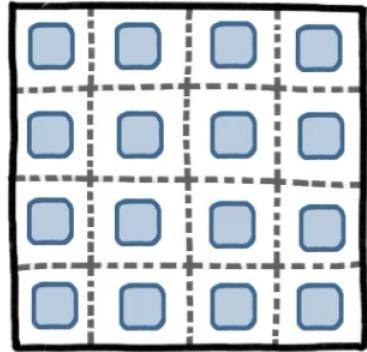
processing job descriptions



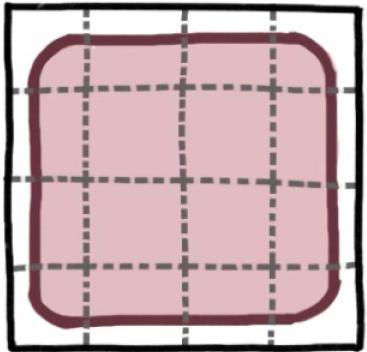
training the model took between
18 and 20 hours

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

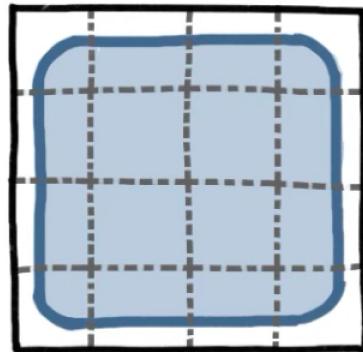
how model weights are split over cores



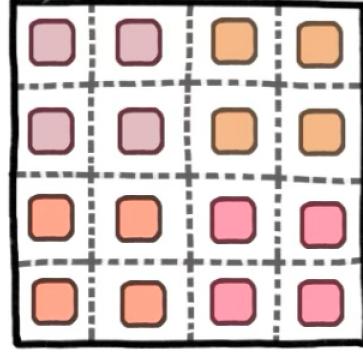
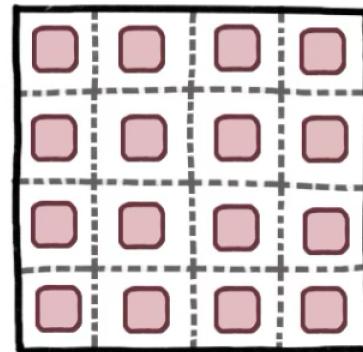
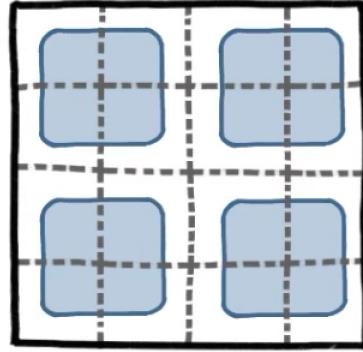
how data is split over cores



model parallelism



model and data parallelism



the end