

# Collecting, Labeling, and Validating Data



DeepLearning.AI

---

# Welcome

# The importance of data

*“Data is the hardest part of ML and the most important piece to get right...  
Broken data is the most common cause of problems in production ML systems”*  
- Scaling Machine Learning at Uber with Michelangelo - Uber

*“No other activity in the machine learning life cycle has a higher return on investment than improving the data a model has access to.”*  
- Feast: Bridging ML Models and Data - Gojek

# Introduction to Machine Learning Engineering for Production



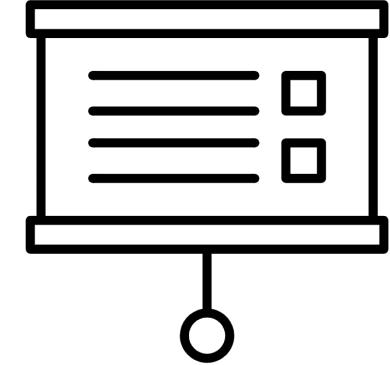
DeepLearning.AI

---

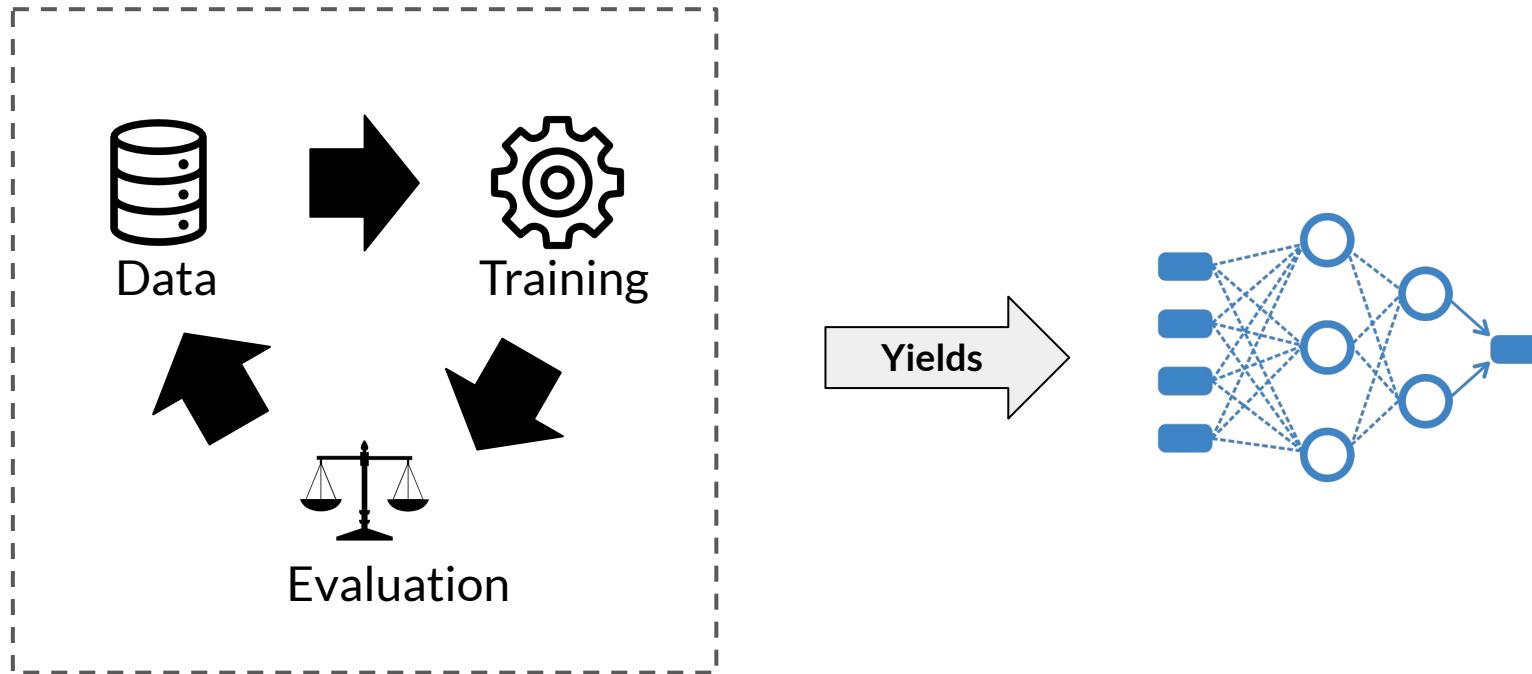
## Overview

# Outline

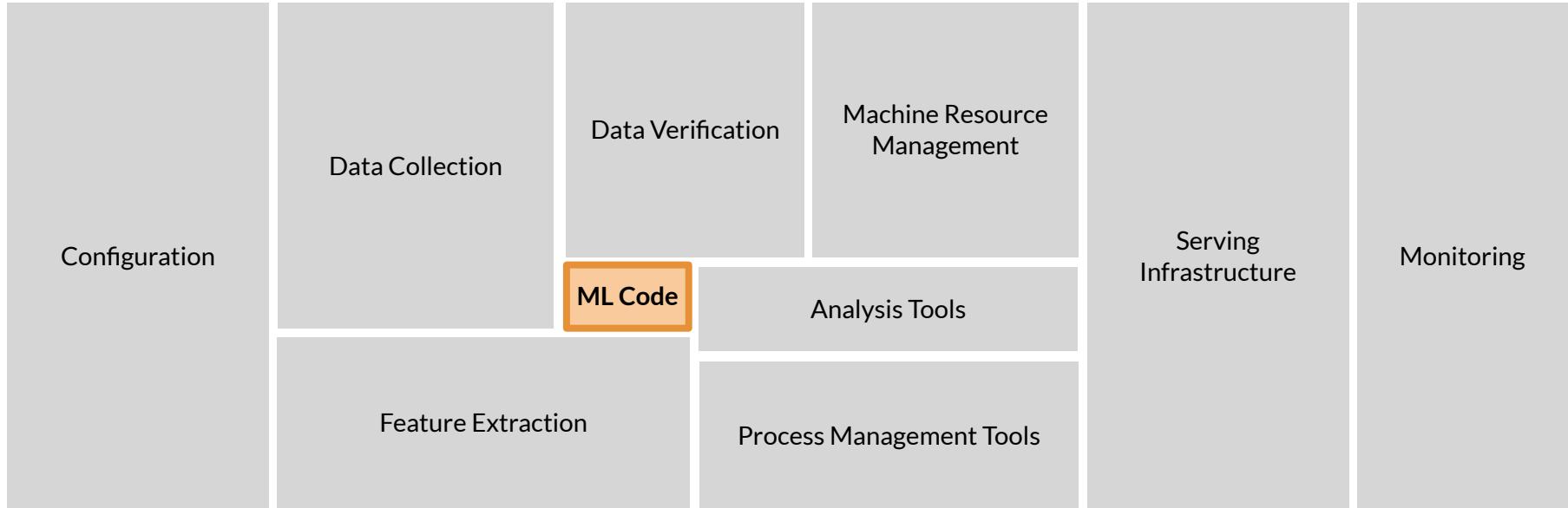
- Machine learning (ML) engineering for production: overview
- Production ML = ML development + software development
- Challenges in production ML



# Traditional ML modeling



# Production ML systems require so much more

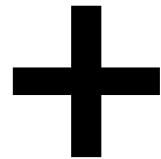


# ML modeling vs production ML

	Academic/Research ML	Production ML
Data	Static	Dynamic - Shifting
Priority for design	Highest overall accuracy	Fast inference, good interpretability
Model training	Optimal tuning and training	Continuously assess and retrain
Fairness	Very important	Crucial
Challenge	High accuracy algorithm	Entire system

# Production machine learning

Machine learning  
development



Modern software  
development

# Managing the entire life cycle of data

- Labeling
- Feature space coverage
- Minimal dimensionality
- Maximum predictive data
- Fairness
- Rare conditions

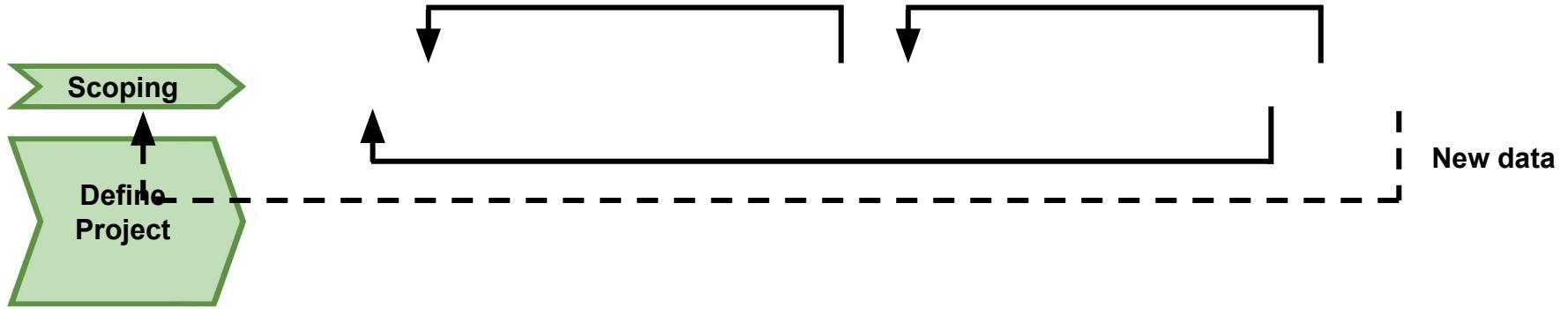
# Modern software development

Accounts for:

- Scalability
- Extensibility
- Configuration
- Consistency & reproducibility
- Safety & security
- Modularity
- Testability
- Monitoring
- Best practices



# Production machine learning system



# Challenges in production grade ML

- Build integrated ML systems
- Continuously operate it in production
- Handle continuously changing data
- Optimize compute resource costs

# Introduction to Machine Learning Engineering for Production



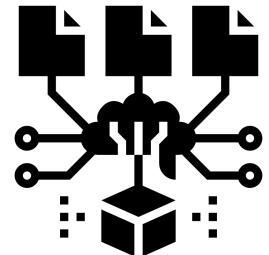
DeepLearning.AI

---

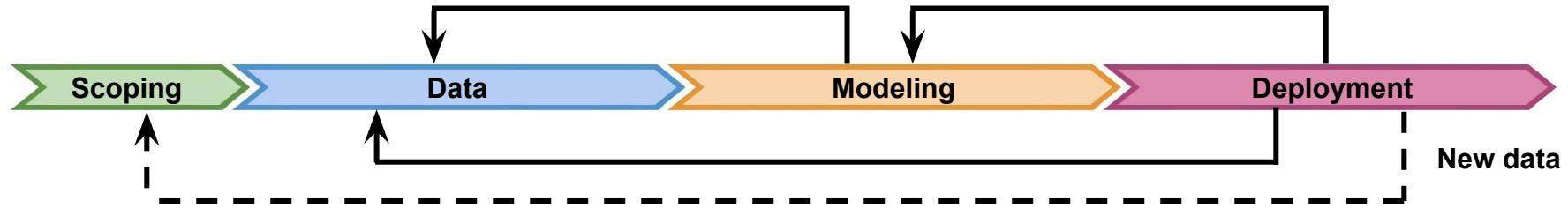
## ML Pipelines

# Outline

- ML Pipelines
- Directed Acyclic Graphs and Pipeline Orchestration Frameworks
- Intro to TensorFlow Extended (TFX)



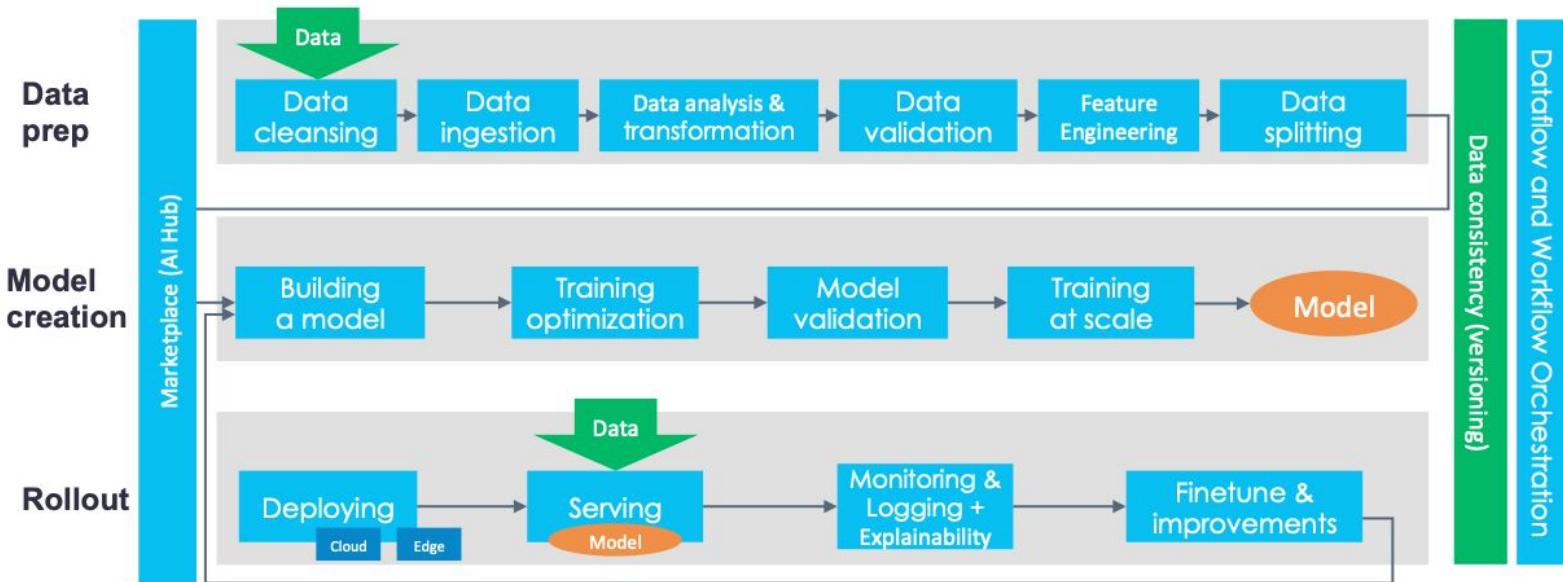
# ML pipelines



Infrastructure for  
automating, monitoring, and maintaining  
model training and deployment

# Production ML infrastructure

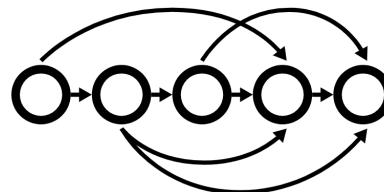
## CD Foundation MLOps reference architecture



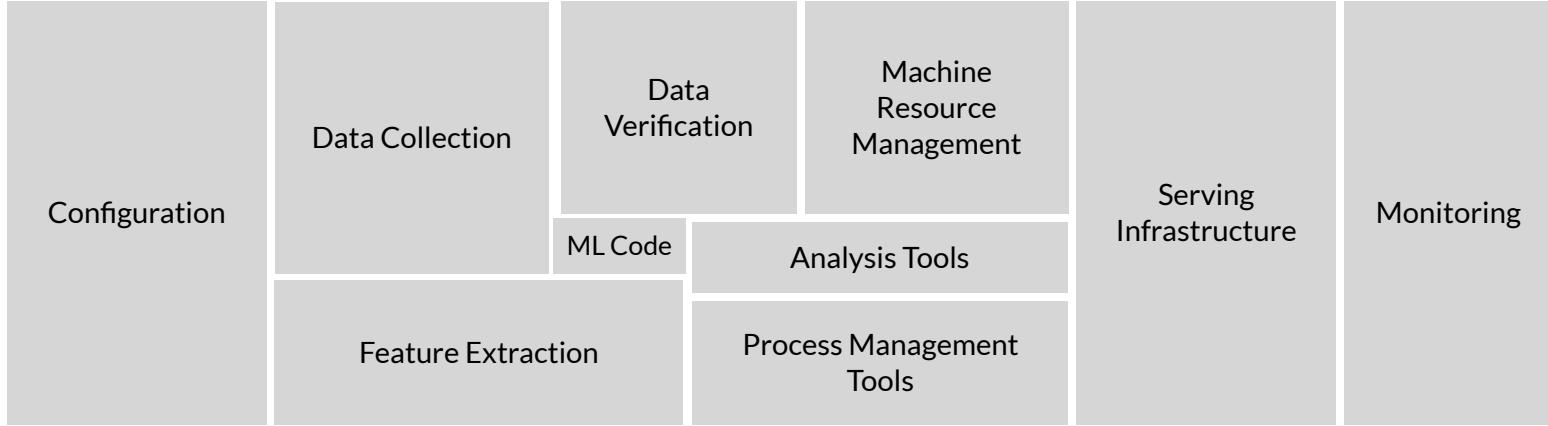
# Directed acyclic graphs



- A directed acyclic graph (DAG) is a directed graph that has no cycles
- ML pipeline workflows are usually DAGs
- DAGs define the sequencing of the tasks to be performed, based on their relationships and dependencies.



# Pipeline orchestration frameworks



- Responsible for scheduling the various components in an ML pipeline DAG dependencies
- Help with pipeline automation
- Examples: Airflow, Argo, Celery, Luigi, Kubeflow

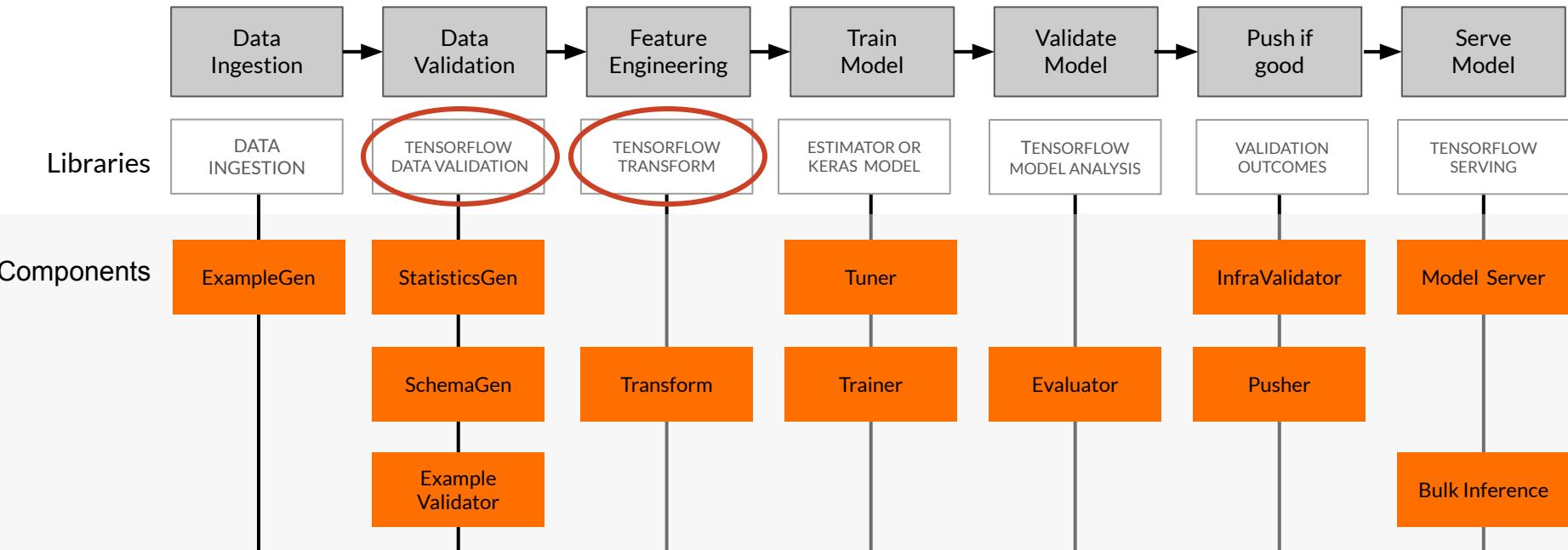
# TensorFlow Extended (TFX)

End-to-end platform for deploying production ML pipelines

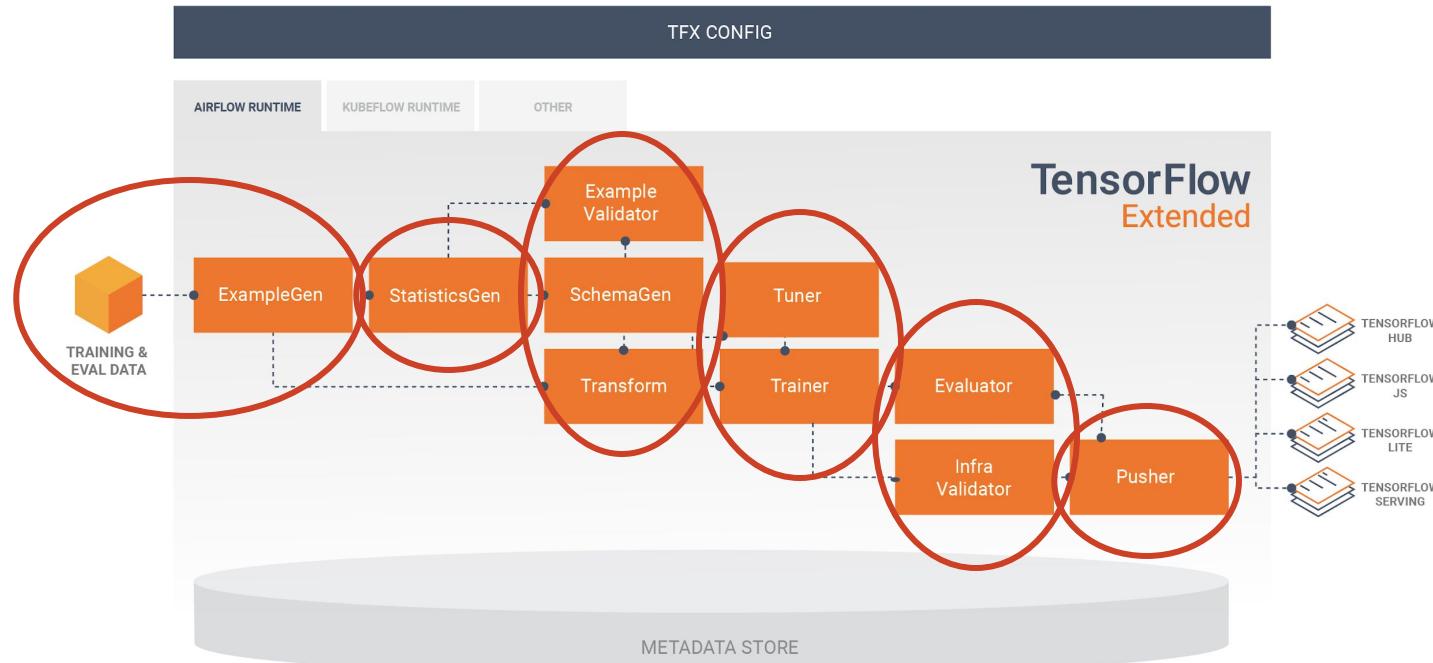


Sequence of components that are designed for scalable, high-performance machine learning tasks

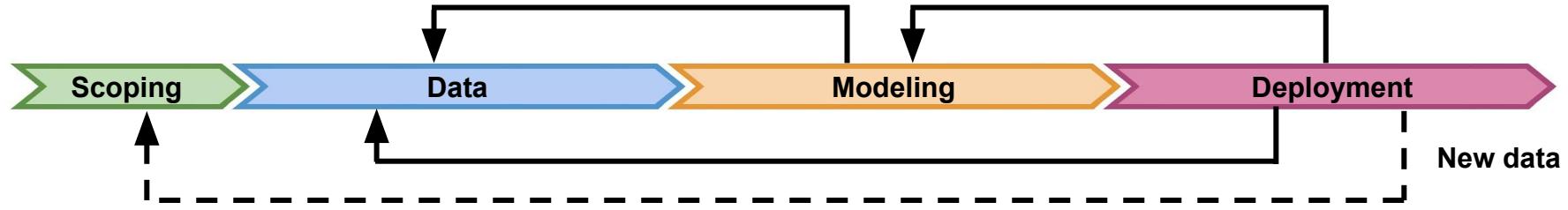
# TFX production components



# TFX Hello World



# Key points



- Production ML pipelines: automating, monitoring, and maintaining end-to-end processes
- Production ML is much more than just ML code
  - ML development + software development
- TFX is an open-source end-to-end ML platform



DeepLearning.AI

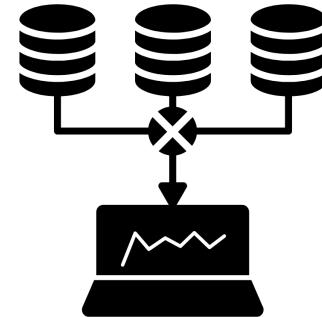
## Collecting Data

---

## Importance of Data

# Outline

- Importance of data quality
- Data pipeline: data collection, ingestion and preparation
- Data collection and monitoring



# The importance of data

*“Data is the hardest part of ML and the most important piece to get right... Broken data is the most common cause of problems in production ML systems”*

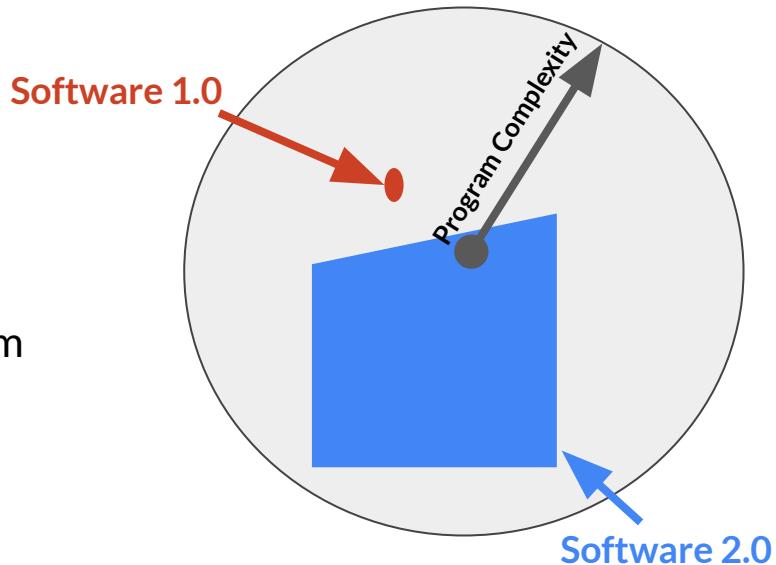
- Scaling Machine Learning at Uber with Michelangelo - Uber

*“No other activity in the machine learning life cycle has a higher return on investment than improving the data a model has access to.”*

- Feast: Bridging ML Models and Data - Gojek

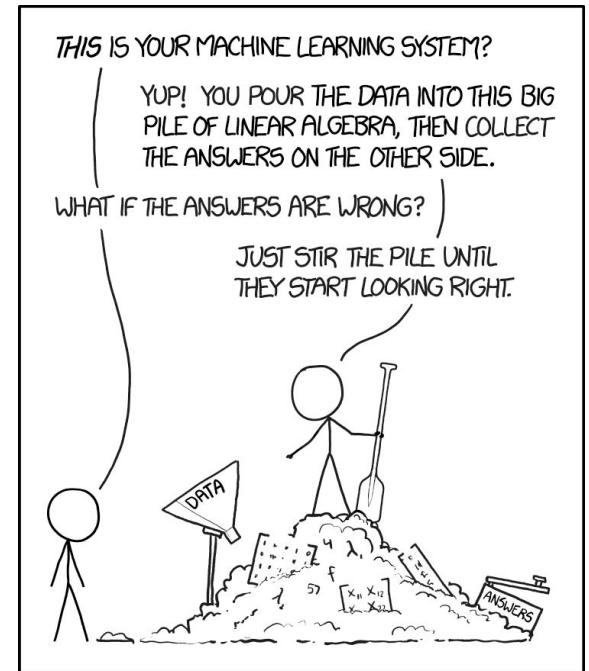
# ML: Data is a first class citizen

- Software 1.0
  - Explicit instructions to the computer
- Software 2.0
  - Specify some goal on the behavior of a program
  - Find solution using optimization techniques
  - Good data is key for success
  - Code in Software = Data in ML

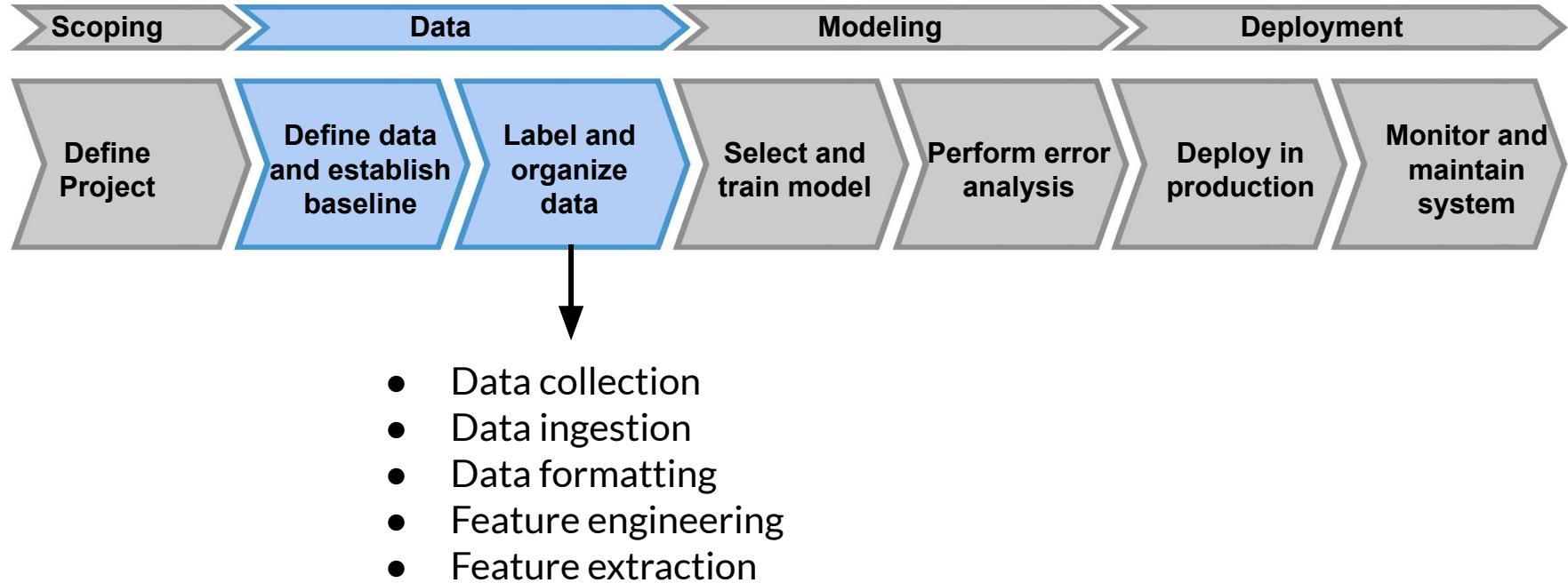


# Everything starts with data

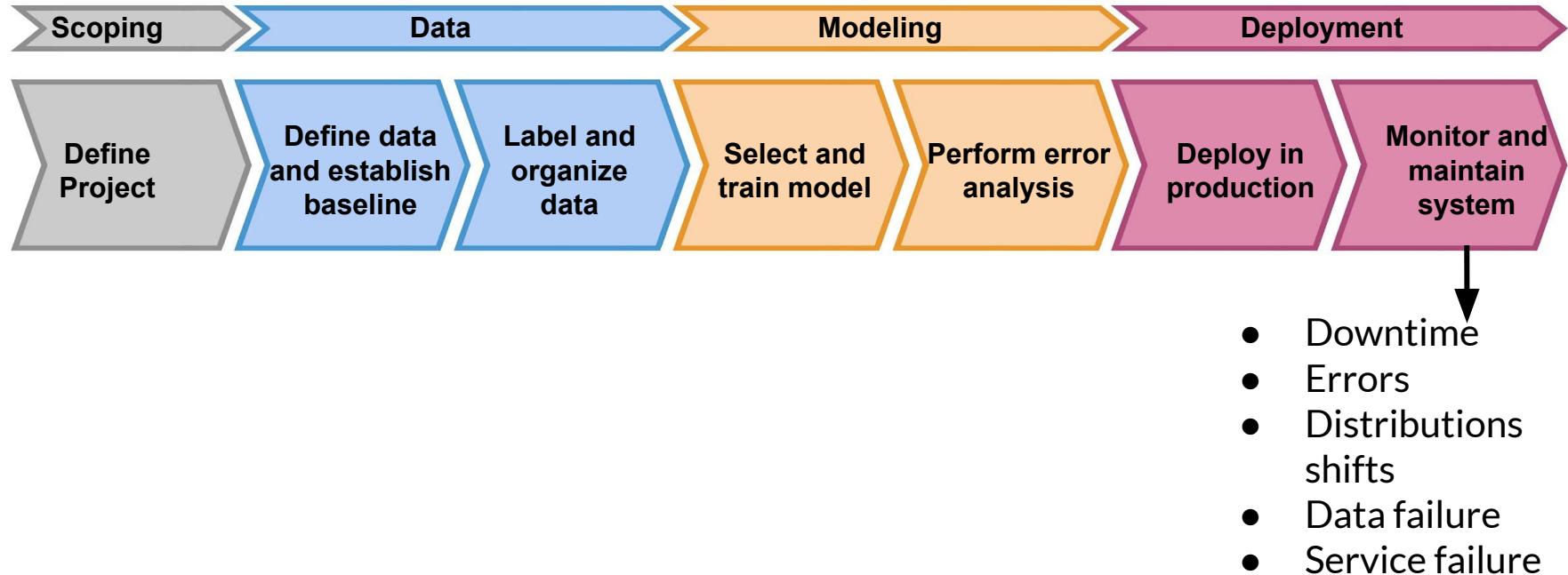
- Models aren't magic
- Meaningful data:
  - maximize predictive content
  - remove non-informative data
  - feature space coverage



# Data pipeline



# Data collection and monitoring



# Key Points

- Understand users, translate user needs into data problems
- Ensure data coverage and high predictive signal
- Source, store and monitor quality data responsibly



DeepLearning.AI

## Collecting Data

---

# Example Application: Suggesting Runs



# Example application: Suggesting runs

<b>Users</b>	Runners
<b>User Need</b>	Run more often
<b>User Actions</b>	Complete run using the app
<b>ML System Output</b>	<ul style="list-style-type: none"><li>• What routes to suggest</li><li>• When to suggest them</li></ul>
<b>ML System Learning</b>	<ul style="list-style-type: none"><li>• Patterns of behaviour around accepting run prompts</li><li>• Completing runs</li><li>• Improving consistency</li></ul>



# Key considerations

- Data availability and collection
  - What kind of/how much data is available?
  - How often does the new data come in?
  - Is it annotated?
    - If not, how hard/expensive is it to get it labeled?
- Translate user needs into data needs
  - Data needed
  - Features needed
  - Labels needed

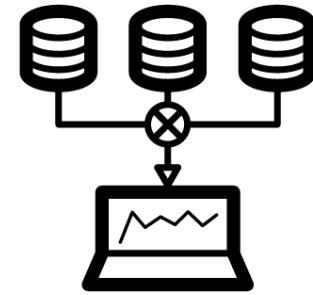


# Example dataset

FEATURES						
EXAMPLES	Runner ID	Run	Runner Time	Elevation	Fun	LABELS
EXAMPLES	AV3DE	Boston Marathon	03:40:32	1,300 ft	Low	LABELS
	X8KGF	Seattle Oktoberfest 5k	00:35:40	0 ft	High	
	BH9IU	Houston Half-marathon	02:01:18	200 ft	Medium	

# Get to know your data

- Identify data sources
- Check if they are refreshed
- Consistency for values, units, & data types
- Monitor outliers and errors



# Dataset issues

- Inconsistent formatting
  - Is zero “0”, “0.0”, or an indicator of a missing measurement
- Compounding errors from other ML Models
- Monitor data sources for system issues and outages

# Measure data effectiveness

- Intuition about data value can be misleading
  - Which features have predictive value and which ones do not?
- Feature engineering helps to maximize the predictive signals
- Feature selection helps to measure the predictive signals

# Translate user needs into data needs

<b>Data Needed</b>	<ul style="list-style-type: none"><li>• Running data from the app</li><li>• Demographic data</li><li>• Local geographic data</li></ul>
--------------------	--

# Translate user needs into data needs

Features Needed	
	<ul style="list-style-type: none"><li>• Runner demographics</li><li>• Time of day</li><li>• Run completion rate</li><li>• Pace</li><li>• Distance ran</li><li>• Elevation gained</li><li>• Heart rate</li></ul>

# Translate user needs into data needs

## Labels Needed

- Runner acceptance or rejection of app suggestions
- User generated feedback regarding why suggestion was rejected
- User rating of enjoyment of recommended runs

# Key points

- Understand your user, translate their needs into data problems
  - What kind of/how much data is available
  - What are the details and issues of your data
  - What are your predictive features
  - What are the labels you are tracking
  - What are your metrics





DeepLearning.AI

## Collecting Data

---

# Responsible Data: Security, Privacy & Fairness

# Outline

- Data Sourcing
- Data Security and User Privacy
- Bias and Fairness



# Avoiding problematic biases in datasets

Example: classifier trained on the Open Images dataset



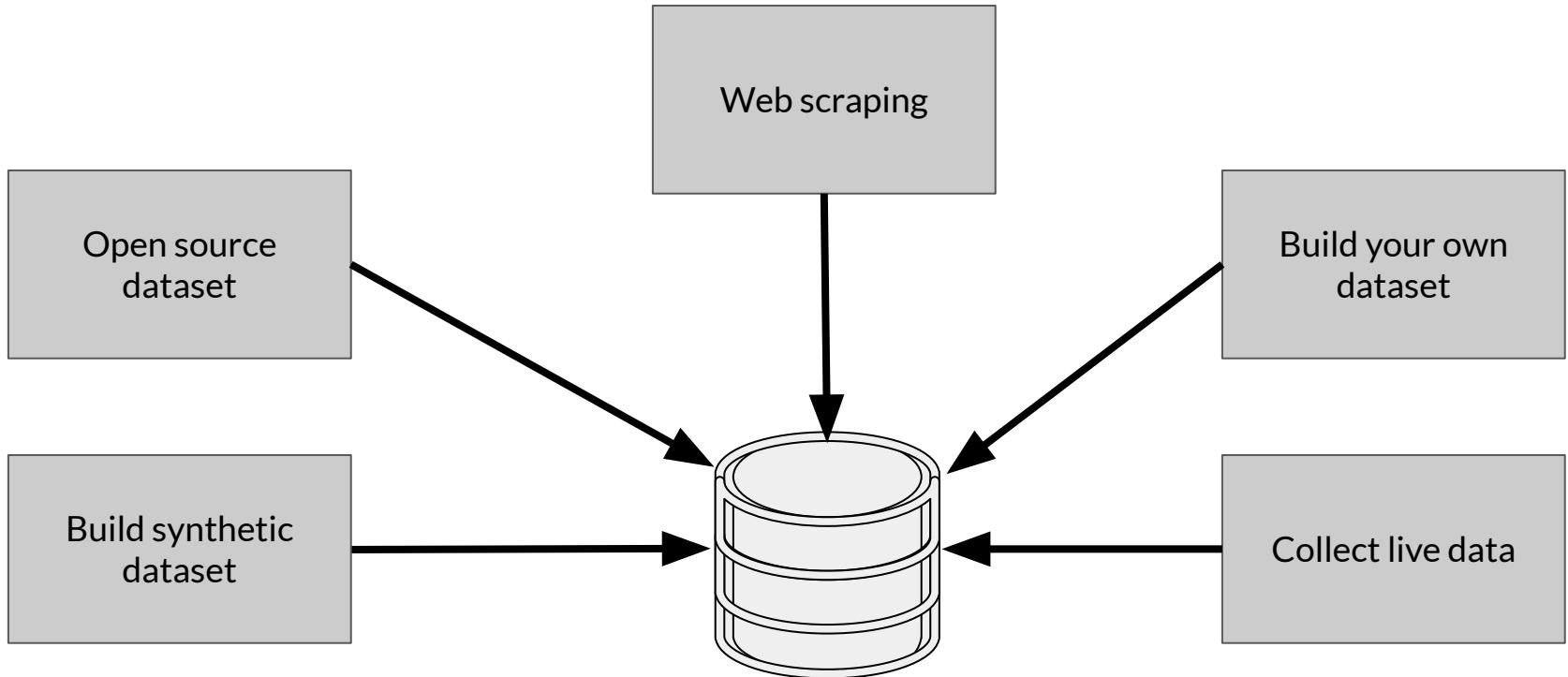
*ceremony,  
wedding, bride,  
man, groom,  
woman, dress*

*bride,  
ceremony,  
wedding, dress,  
woman*

*ceremony,  
bride, wedding,  
man, groom,  
woman, dress*

*person, people*

# Source Data Responsibly



# Data security and privacy

- Data collection and management isn't just about your model
  - Give user control of what data can be collected
  - Is there a risk of inadvertently revealing user data?
- Compliance with regulations and policies (e.g. GDPR)

# Users privacy

- Protect personally identifiable information
  - Aggregation - replace unique values with summary value
  - Redaction - remove some data to create less complete picture

# How ML systems can fail users



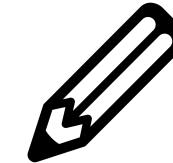
Fair



Accountable



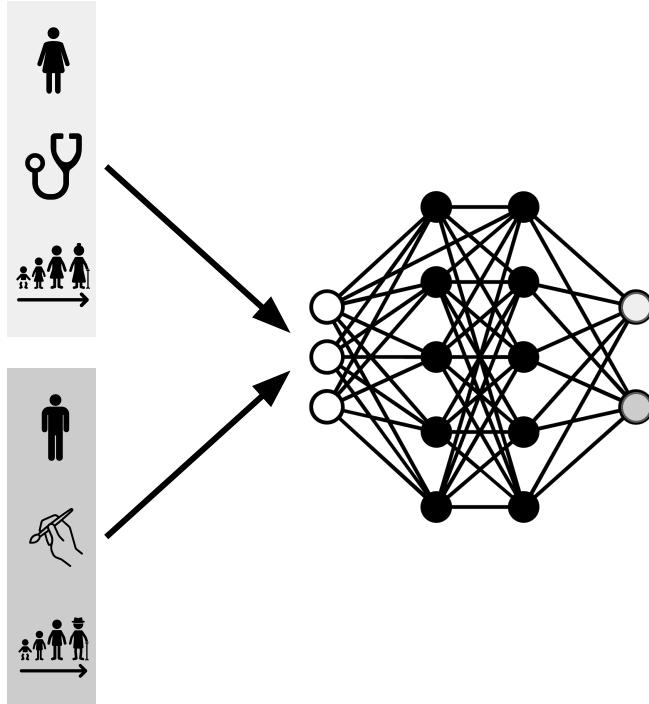
Transparent



Explainable

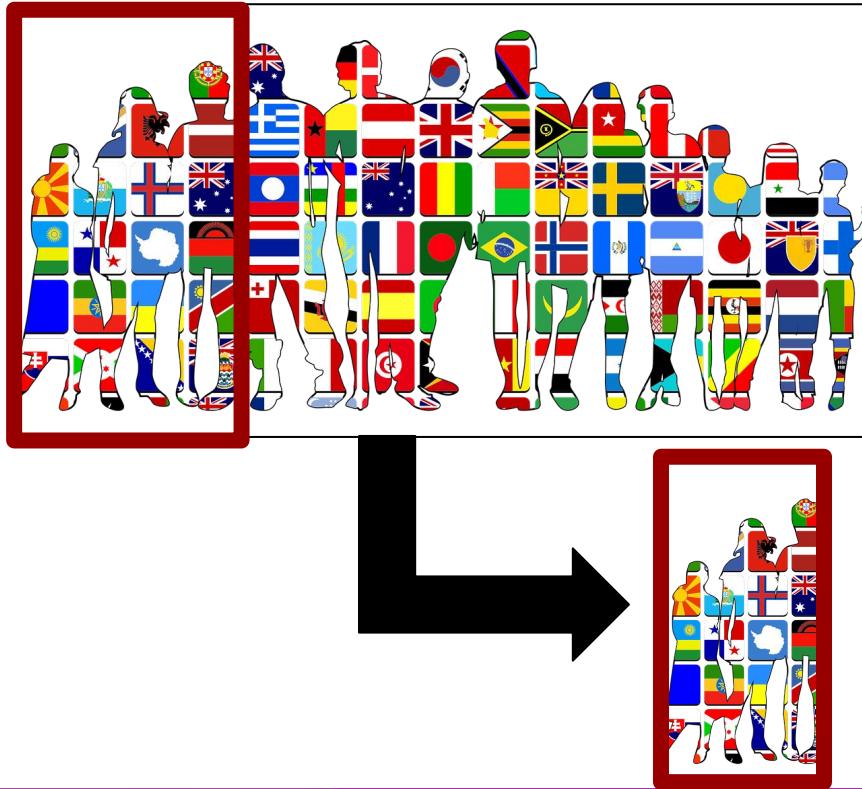
- Representational harm
- Opportunity denial
- Disproportionate product failure
- Harm by disadvantage

# Commit to fairness



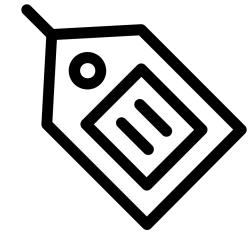
- Make sure your models are fair
  - Group fairness, equal accuracy
- Bias in human labeled and/or collected data
- ML Models can amplify biases

# Biased data representation

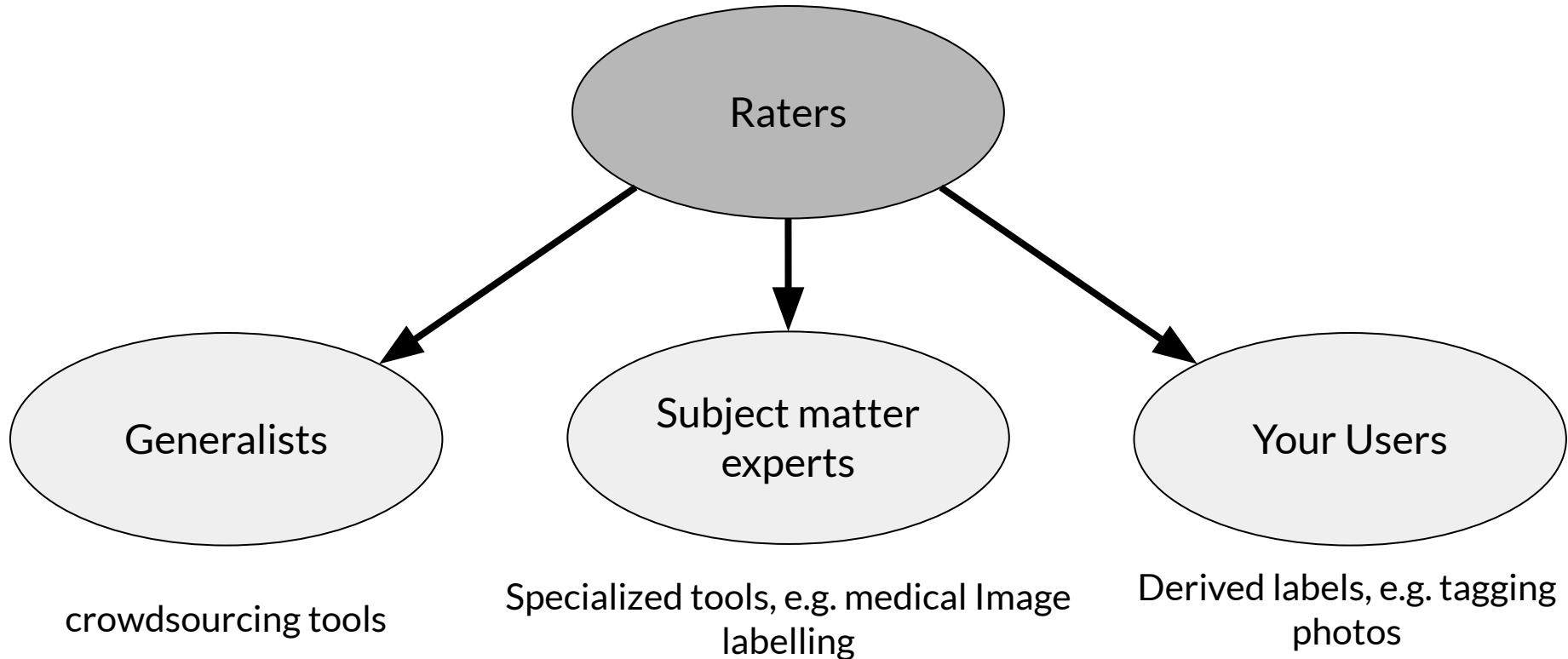


# Reducing bias: Design fair labeling systems

- Accurate labels are necessary for supervised learning
- Labeling can be done:
  - Automation (logging or weak supervision)
  - Humans (aka “Raters”, often semi-supervised)



# Types of human raters



# Key points

- Ensure rater pool diversity
- Investigate rater context and incentives
- Evaluate rater tools
- Manage cost
- Determine freshness requirements



DeepLearning.AI

## Labeling Data

---

# Case Study: Degraded Model Performance

# You're an Online Retailer Selling Shoes ...

Your model predicts  
**click-through rates**  
**(CTR)**, helping you decide  
how much inventory to  
order



# When suddenly

Your AUC and prediction accuracy  
have dropped on men's dress shoes!





# Why?



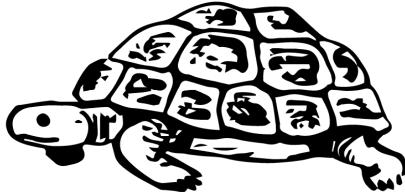
# Case study: taking action

- How to detect problems early on?
- What are the possible causes?
- What can be done to solve these?

# What causes problems?

Kinds of problems:

- Slow - example: drift
- Fast - example: bad sensor, bad software update



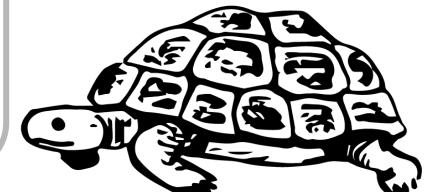
# Gradual problems

## Data changes

- Trend and seasonality
- Distribution of features changes
- Relative importance of features changes

## World changes

- Styles change
- Scope and processes change
- Competitors change
- Business expands to other geos



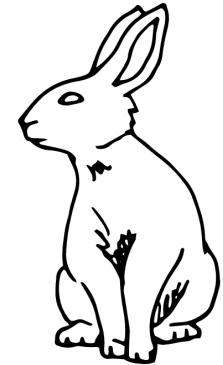
# Sudden problems

## Data collection problem

- Bad sensor/camera
- Bad log data
- Moved or disabled sensors/cameras

## Systems problem

- Bad software update
- Loss of network connectivity
- System down
- Bad credentials



# Why “Understand” the model?

- Mispredictions do not have uniform **cost** to your business
- The **data you have** is rarely the data you wish you had
- Model objective is nearly always a **proxy** for your business objectives
- Some percentage of your customers may have a **bad experience**

**The real world does not stand still!**



DeepLearning.AI

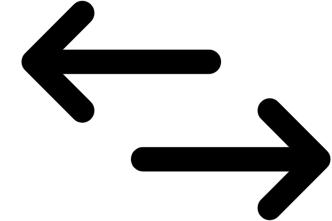
## Labeling Data

---

Data and Concept  
Change in  
Production ML

# Outline

- Detecting problems with deployed models
  - Data and concept change
- Changing ground truth
  - Easy problems
  - Harder problems
  - Really hard problems



# Detecting problems with deployed models

- Data and scope changes
- Monitor models and validate data to find problems early
- Changing ground truth: **label** new training data

# Easy problems

- Ground truth changes slowly (months, years)
- Model retraining driven by:
  - Model improvements, better data
  - Changes in software and/or systems
- Labeling
  - Curated datasets
  - Crowd-based



# Harder problems

- Ground truth changes faster (weeks)
- Model retraining driven by:
  - Declining model performance
  - Model improvements, better data
  - Changes in software and/or system
- Labeling
  - Direct feedback
  - Crowd-based



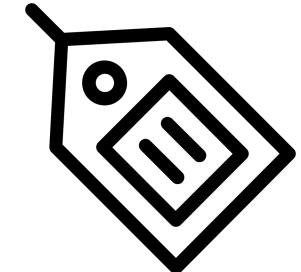
# Really hard problems

- Ground truth changes very fast (days, hours, min)
- Model retraining driven by:
  - Declining model performance
  - Model improvements, better data
  - Changes in software and/or system
- Labeling
  - Direct feedback
  - Weak supervision



# Key points

- Model performance decays over time
  - Data and Concept Drift
- Model retraining helps to improve performance
  - Data labeling for changing ground truth and scarce labels





DeepLearning.AI

## Labeling Data

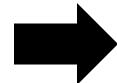
---

# Process Feedback and Human Labeling

# Data labeling

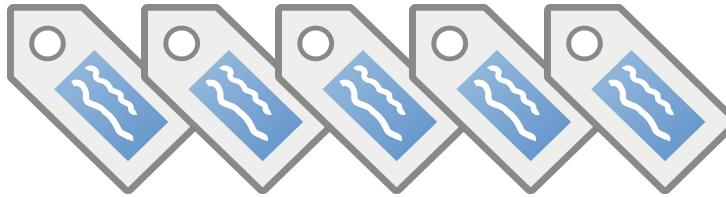
## Variety of Methods

- Process Feedback (Direct Labeling)
- Human Labeling
- ~~Semi Supervised Labeling~~
- ~~Active Learning~~
- ~~Weak Supervision~~



Practice later as advanced  
labeling methods

# Data labeling



**Process Feedback**

Example: Actual vs predicted click-through

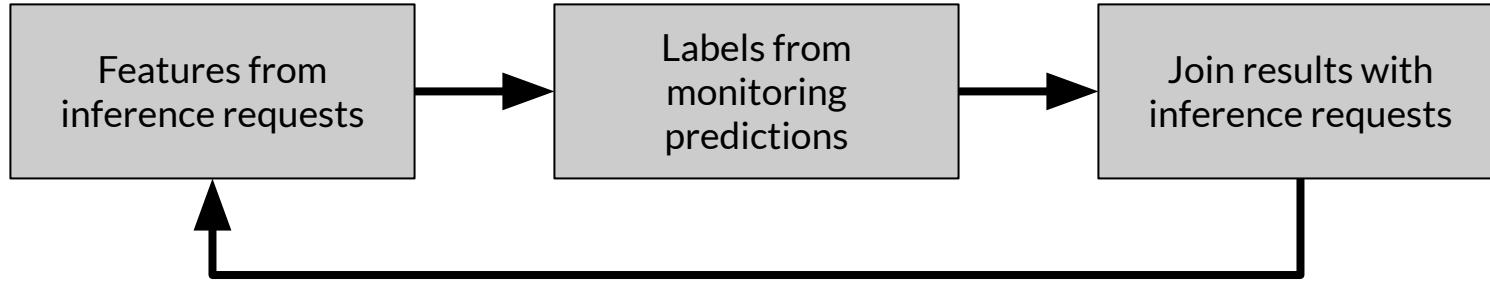
**Human Labeling**

Example: Cardiologists labeling MRI images

# Why is labeling important in production ML?

- Using business/organisation available data
- Frequent model retraining
- Labeling ongoing and critical process
- Creating a training datasets requires labels

# Direct labeling: continuous creation of training dataset



Similar to reinforcement learning  
rewards

# Process feedback - advantages

- Training dataset continuous creation
- Labels evolve quickly
- Captures strong label signals

# Process feedback - disadvantages

- Hindered by inherent nature of the problem
- Failure to capture ground truth
- Largely bespoke design

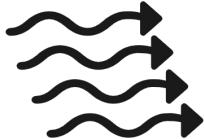
# Process feedback - Open-Source log analysis tools



## Logstash

Free and open source data processing pipeline

- Ingests data from a multitude of sources
- Transforms it
- Sends it to your favorite "stash."

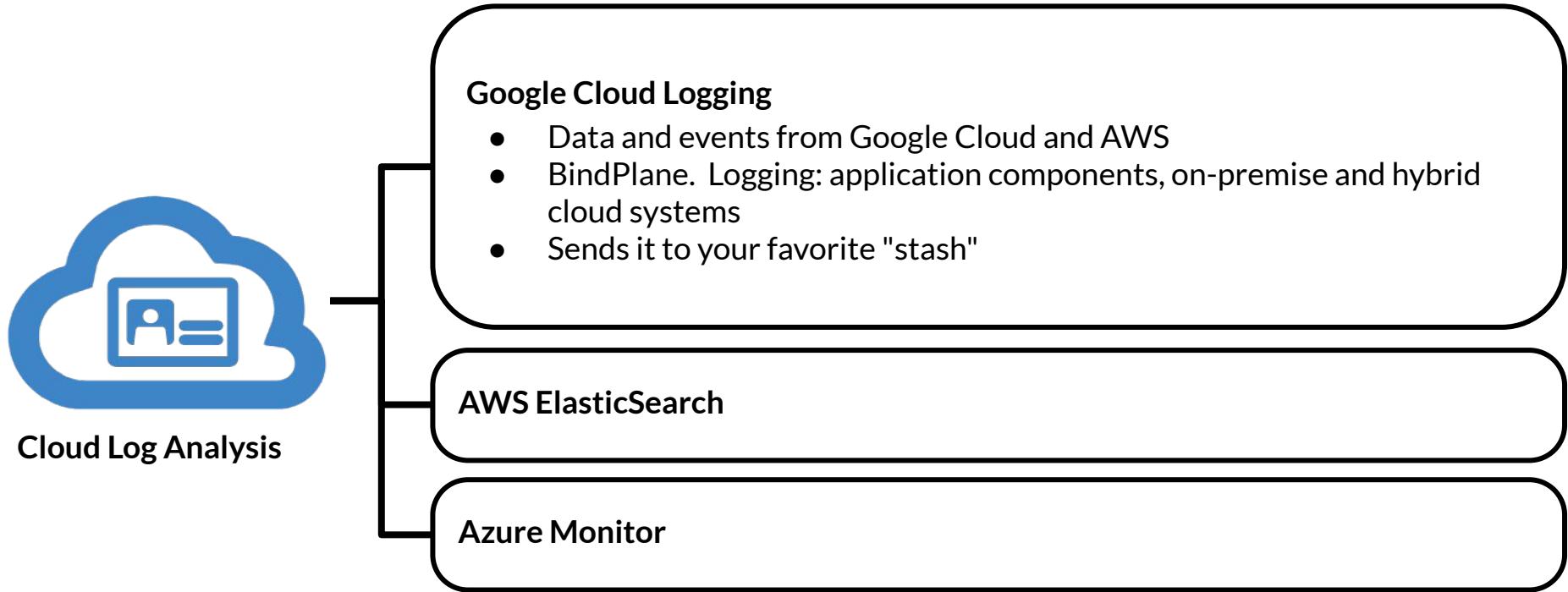


## Fluentd

Open source data collector

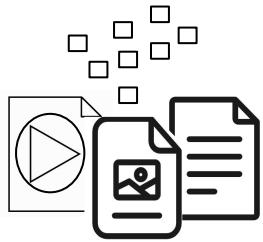
Unify the data collection and consumption

# Process feedback - Cloud log analytics



# Human labeling

People (“raters”) to examine data and assign labels manually



Raw data



Unlabeled and ambiguous data  
is sent to raters for annotation



A training data set is  
ready for use

# Human labeling - Methodology



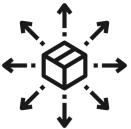
Unlabeled data is collected



Human “raters” are recruited



Instructions to guide raters are created



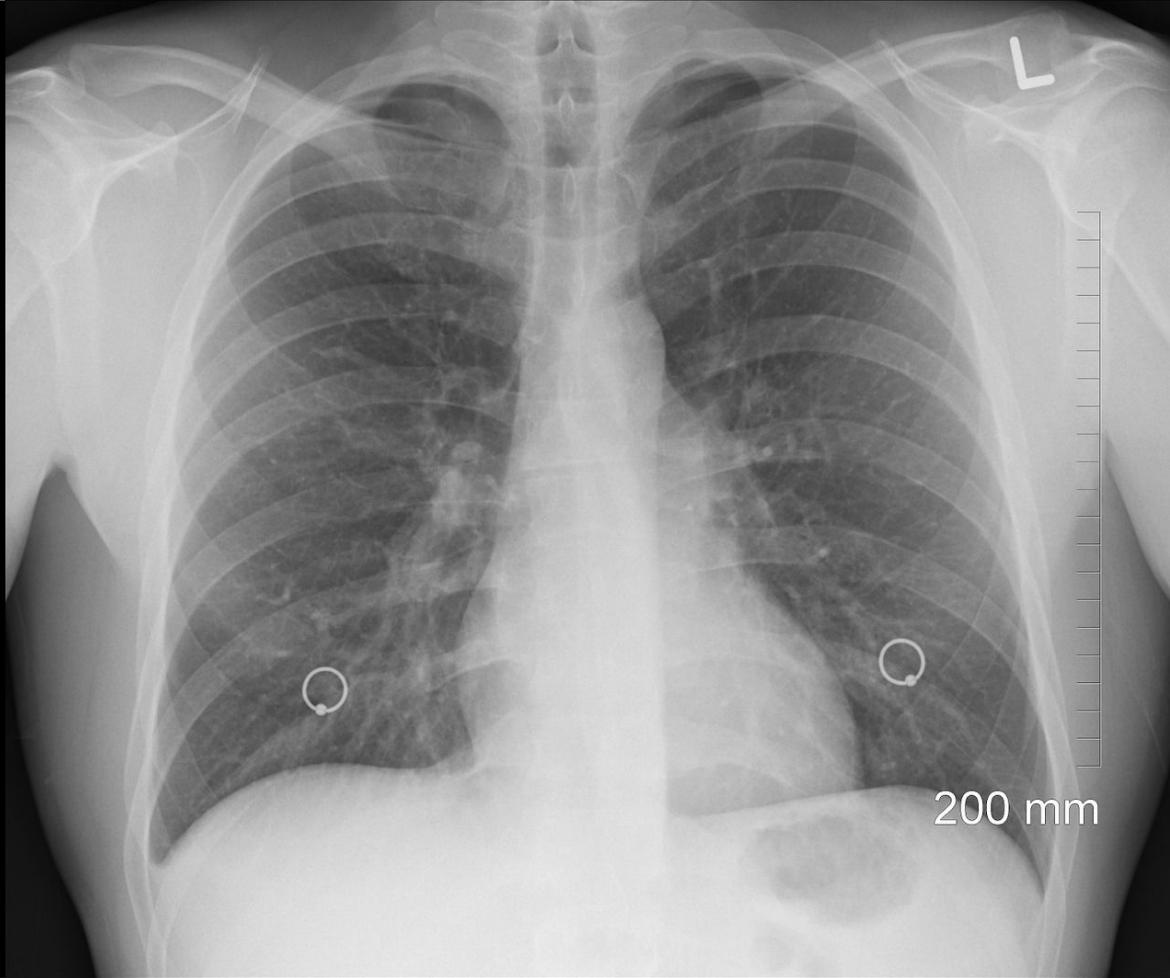
Data is divided and assigned to raters



Labels are collected and conflicts resolved

# Human labeling - advantages

- More labels
- Pure supervised learning



# Human labeling - Disadvantages



Quality consistency: Many datasets  
difficult for human labeling



Slow

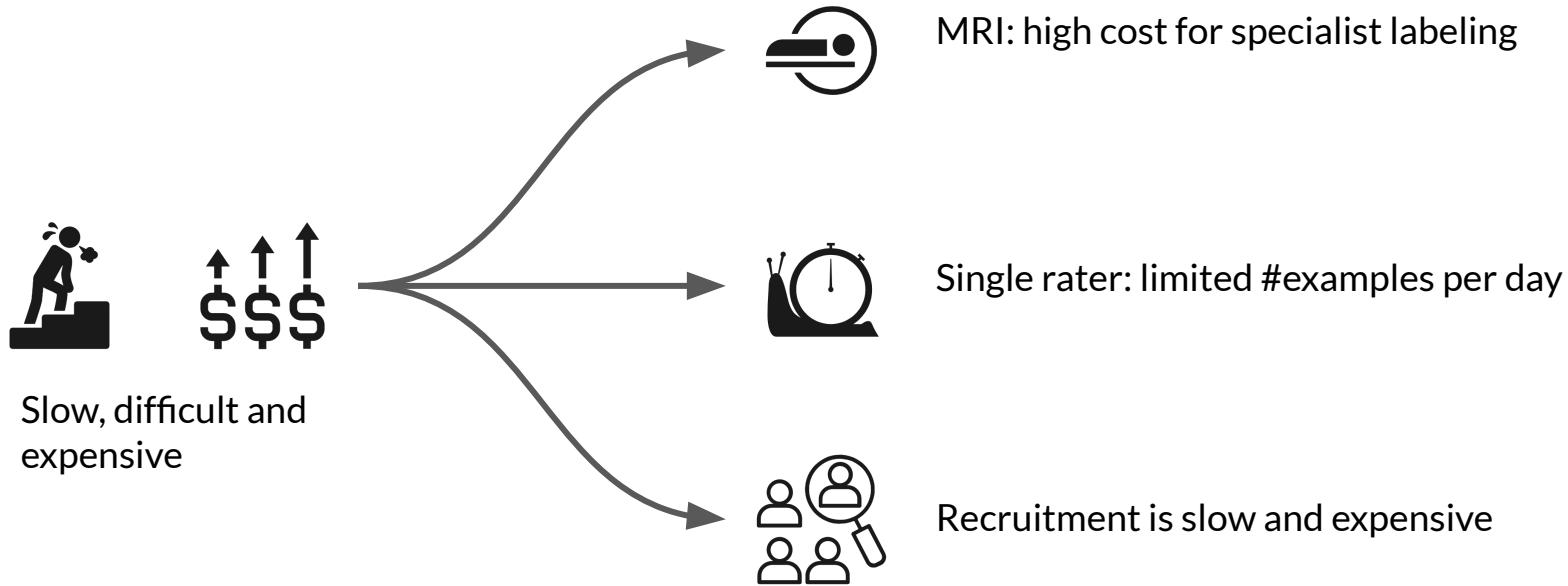


Expensive



Small dataset curation

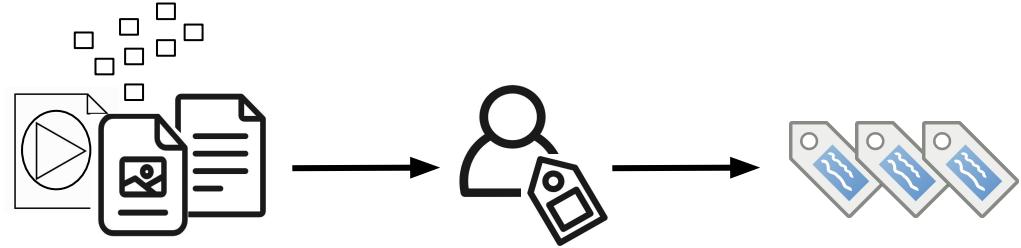
# Why is human labeling a problem?



Slow, difficult and  
expensive

# Key points

- Various methods of data labeling
  - Process feedback
  - Human labeling
- Advantages and disadvantages of both





DeepLearning.AI

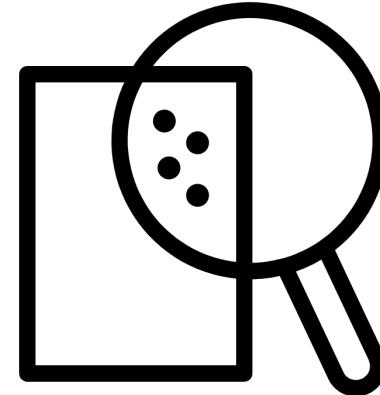
## Validating Data

---

## Detecting Data Issues

# Outline

- Data issues
  - Drift and skew
    - Data and concept Drift
    - Schema Skew
    - Distribution Skew
- Detecting data issues



# Drift and skew

## Drift

Changes in data over time, such as data collected once a day

## Skew

Difference between two static versions, or different sources, such as training set and serving set

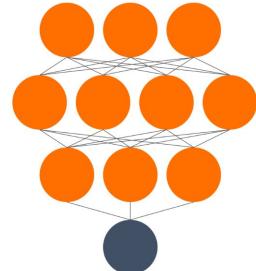
# Typical ML pipeline

During **training**

Data



Batch processing

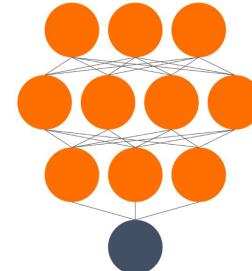


During **serving**

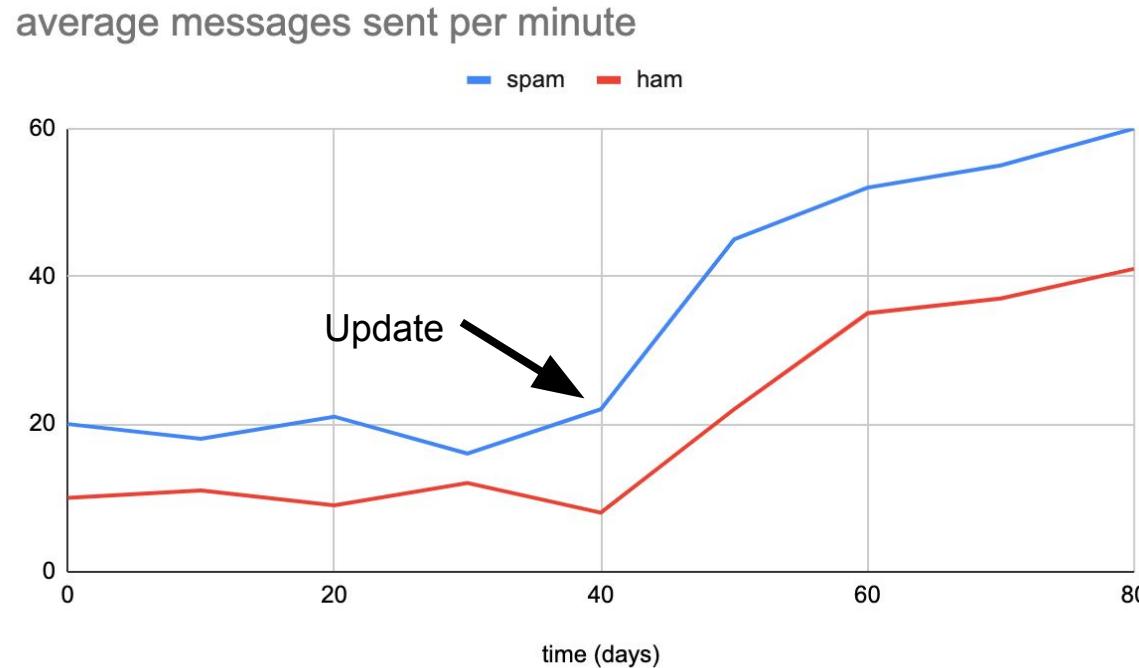
Request



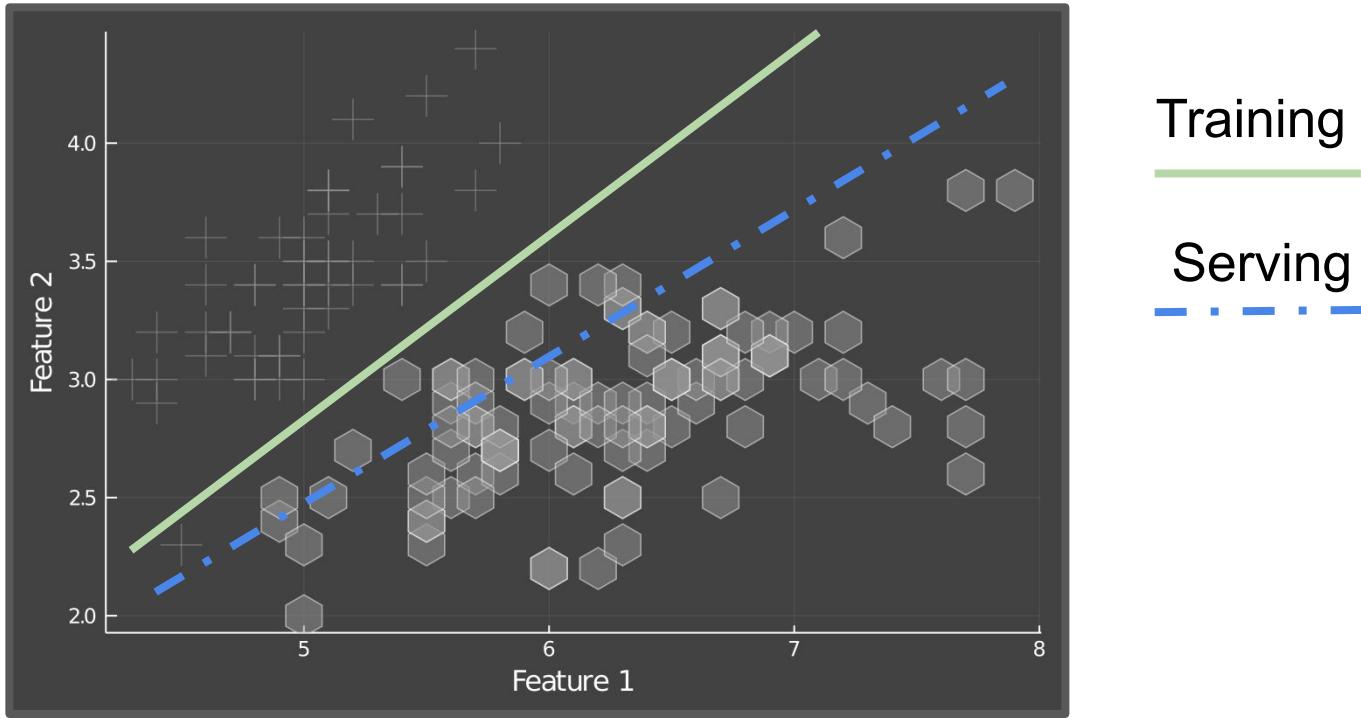
Real-time  
processing



# Model Decay : Data drift



# Performance decay : Concept drift



# Detecting data issues

- Detecting schema skew
  - Training and serving data do not conform to the same schema
- Detecting distribution skew
  - Dataset shift → covariate or concept shift
- Requires continuous evaluation

# Detecting distribution skew

	Training	Serving
Joint	$P_{\text{train}}(y, x)$	$P_{\text{serve}}(y, x)$
Conditional	$P_{\text{train}}(y x)$	$P_{\text{serve}}(y x)$
Marginal	$P_{\text{train}}(x)$	$P_{\text{serve}}(x)$

Dataset shift

$$P_{\text{train}}(y, x) \neq P_{\text{serve}}(y, x)$$

Covariate shift

$$P_{\text{train}}(y|x) = P_{\text{serve}}(y|x)$$

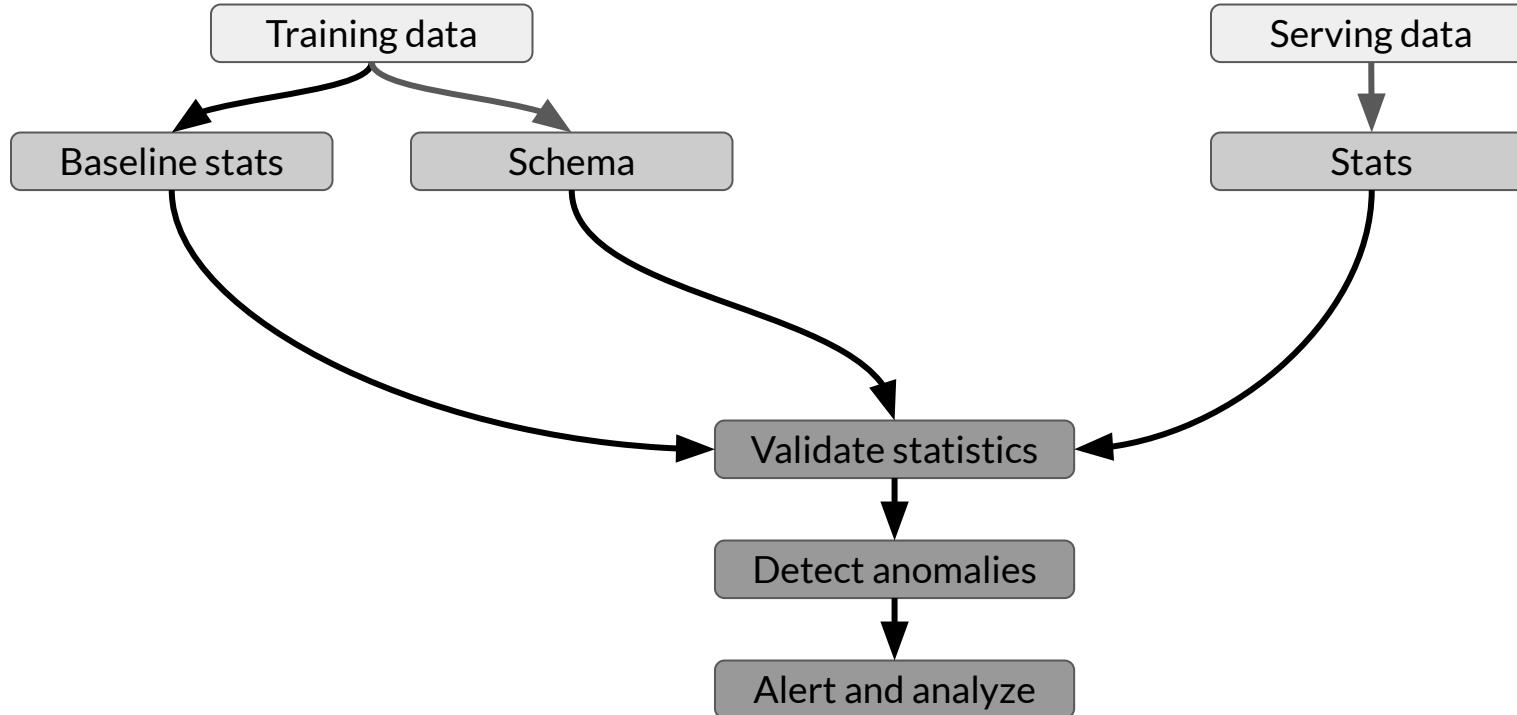
$$P_{\text{train}}(x) \neq P_{\text{serve}}(x)$$

Concept shift

$$P_{\text{train}}(y|x) \neq P_{\text{serve}}(y|x)$$

$$P_{\text{train}}(x) = P_{\text{serve}}(x)$$

# Skew detection workflow





DeepLearning.AI

## Validating Data

---

TensorFlow  
Data Validation

# TensorFlow Data Validation (TFDV)

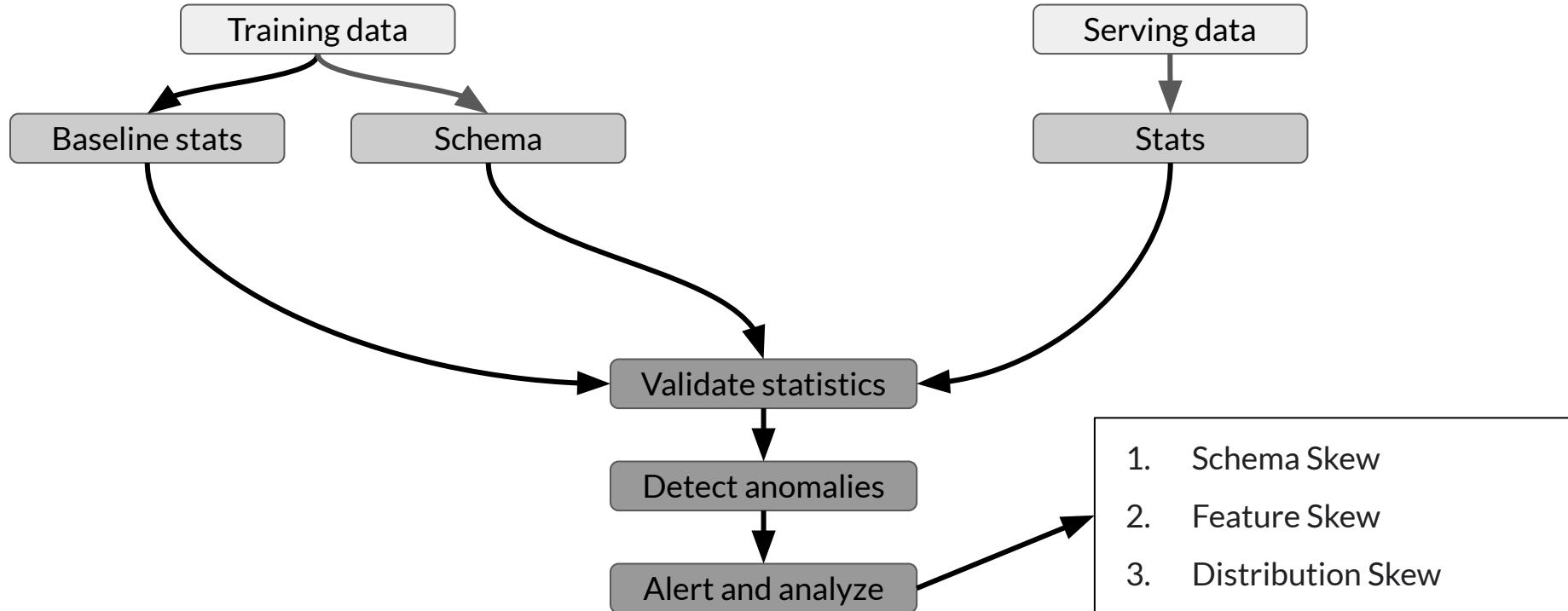


- Understand, validate, and monitor ML data at scale
- Used to analyze and validate petabytes of data at Google every day
- Proven track record in helping TFX users maintain the health of their ML pipelines

# TFDV capabilities

- Generates data statistics and browser visualizations
- Infers the data schema
- Performs validity checks against schema
- Detects training/serving skew

# Skew detection - TFDV

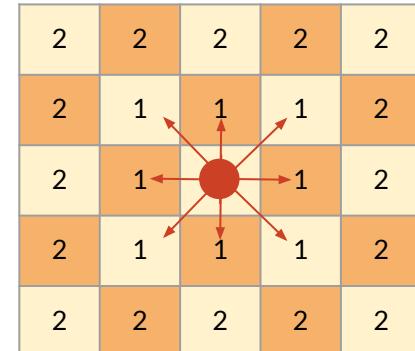


# Skew - TFDV

- Supported for categorical features
- Expressed in terms of L-infinity distance (Chebyshev Distance):

$$D_{\text{Chebyshev}}(x, y) = \max_i(|x_i - y_i|)$$

- Set a threshold to receive warnings



# Schema skew

Serving and training data don't conform to same schema:

- For example, `int != float`

# Feature skew

Training **feature values** are different than the serving **feature values**:

- Feature values are modified between training and serving time
- Transformation applied only in one of the two instances

# Distribution skew

**Distribution** of serving and training dataset is significantly different:

- Faulty sampling method during training
- Different data sources for training and serving data
- Trend, seasonality, changes in data over time

# Key points

- TFDV: Descriptive statistics at scale with the embedded facets visualizations
- It provides insight into:
  - What are the underlying statistics of your data
  - How does your training, evaluation, and serving dataset statistics compare
  - How can you detect and fix data anomalies

# Wrap up

- Differences between ML modeling and a production ML system
- Responsible data collection for building a fair production ML system
- Process feedback and human labeling
- Detecting data issues

**Practice data validation with TFDV in this week's exercise notebook**

**Test your skills with the programming assignment**

# Feature Engineering, Transformation and Selection



DeepLearning.AI

---

# Welcome

# Feature Engineering

---



DeepLearning.AI

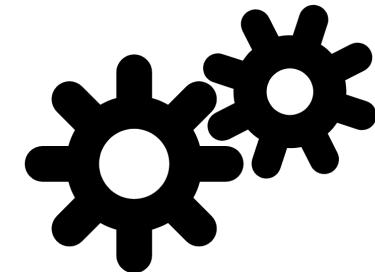
# Introduction to Preprocessing

*“Coming up with features is difficult,  
time-consuming, and requires expert knowledge.  
Applied machine learning often requires careful  
engineering of the features and dataset.”*

— Andrew Ng

# Outline

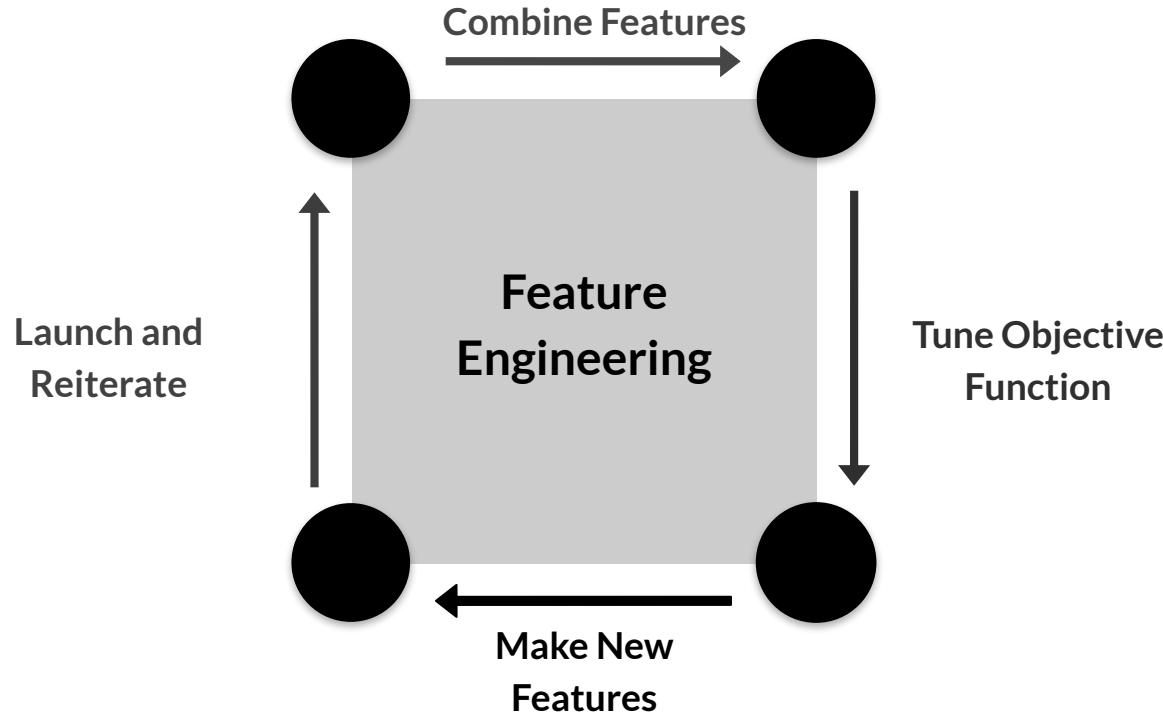
- Squeezing the most out of data
- The art of feature engineering
- Feature engineering process
- How feature engineering is done in a typical ML pipeline



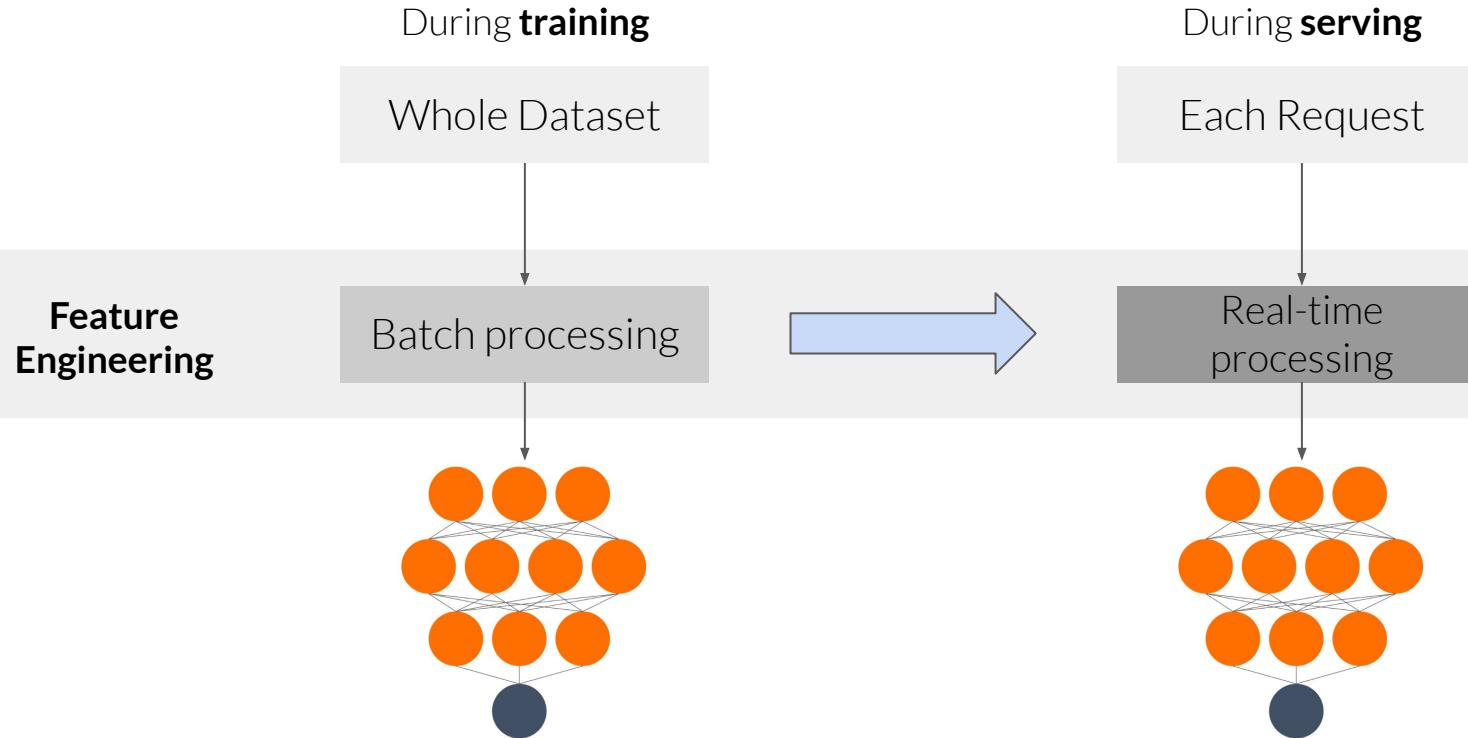
# Squeezing the most out of data

- Making data useful before training a model
- Representing data in forms that help models learn
- Increasing predictive quality
- Reducing dimensionality with feature engineering

# Art of feature engineering



# Typical ML pipeline



# Key points

- Feature engineering can be difficult and time consuming, but also very important to success
- Squeezing the most out of data through feature engineering enables models to learn better
- Concentrating predictive information in fewer features enables more efficient use of compute resources
- Feature engineering during training must also be applied correctly during serving

# Feature Engineering

---

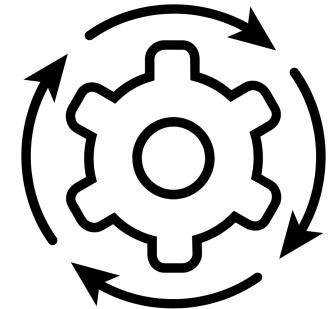


DeepLearning.AI

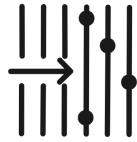
# Preprocessing Operations

# Outline

- Main preprocessing operations
- Mapping raw data into features
- Mapping numeric values
- Mapping categorical values
- Empirical knowledge of data



# Main preprocessing operations



Data cleansing



Feature tuning



Representation  
transformation

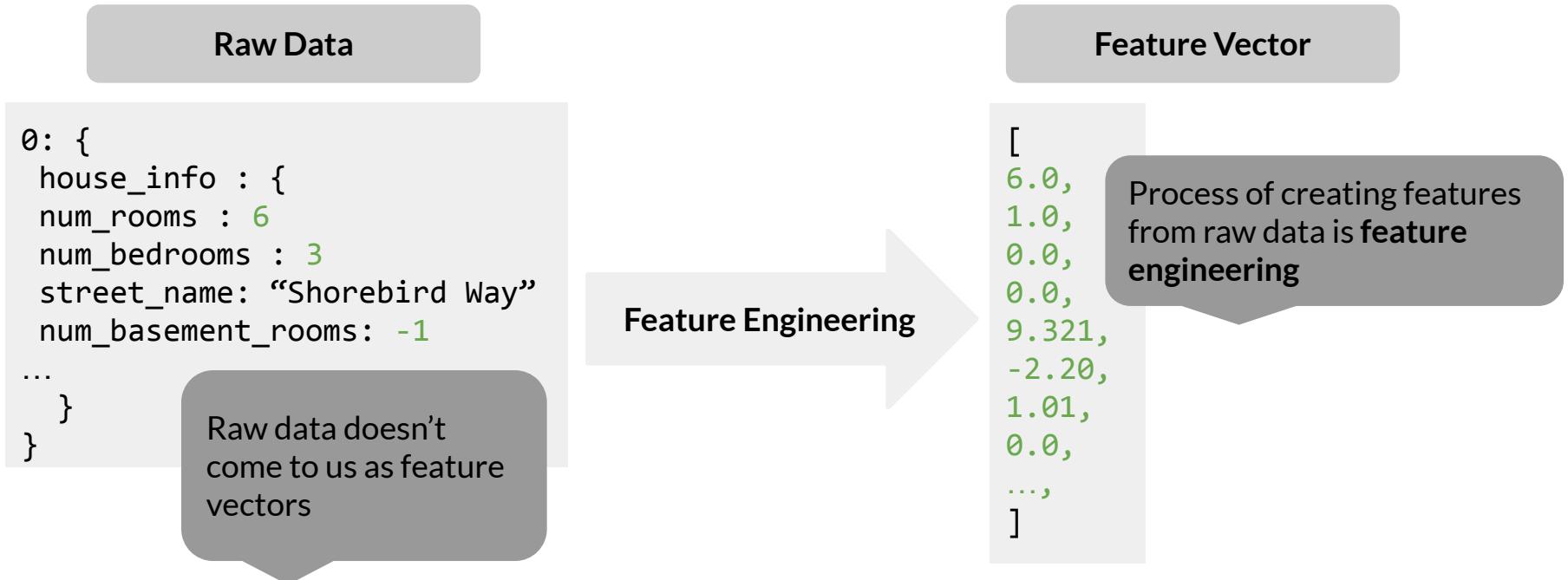


Feature  
extraction



Feature  
construction

# Mapping raw data into features



# Mapping categorical values

## Street names

{'Charleston Road', 'North Shoreline Boulevard', 'Shorebird Way', 'Rengstorff Avenue'}

Raw Data

```
0: {  
    house_info : {  
        num_rooms : 6  
        num_bedrooms : 3  
        street_name: "Shorebird Way"  
        num_basement_rooms: -1  
    ...  
    }  
}
```

String Features can be handled with one-hot encoding

Feature Engineering

Feature Vector

One-hot encoding  
This has a 1 for “Shorebird way” and 0 for all others

street\_name feature=  
[0,0, ..., 0, 1, 0, ..., 0]

# Categorical Vocabulary

```
# From a vocabulary list

vocabulary_feature_column = tf.feature_column.categorical_column_with_vocabulary_list(
    key=feature_name,
    vocabulary_list=["kitchenware", "electronics", "sports"])

# From a vocabulary file

vocabulary_feature_column = tf.feature_column.categorical_column_with_vocabulary_file(
    key=feature_name,
    vocabulary_file="product_class.txt",
    vocabulary_size=3)
```

# Empirical knowledge of data



**Text** - stemming, lemmatization, TF-IDF, n-grams, embedding lookup



**Images** - clipping, resizing, cropping, blur, Canny filters, Sobel filters, photometric distortions

# Key points

- Data preprocessing: transforms raw data into a clean and training-ready dataset
- Feature engineering maps:
  - Raw data into feature vectors
  - Integer values to floating-point values
  - Normalizes numerical values
  - Strings and categorical values to vectors of numeric values
  - Data from one space into a different space



DeepLearning.AI

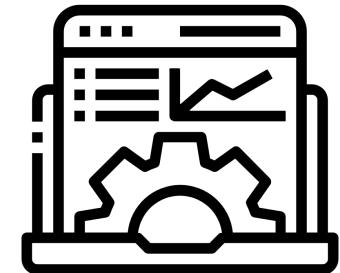
# Feature Engineering

---

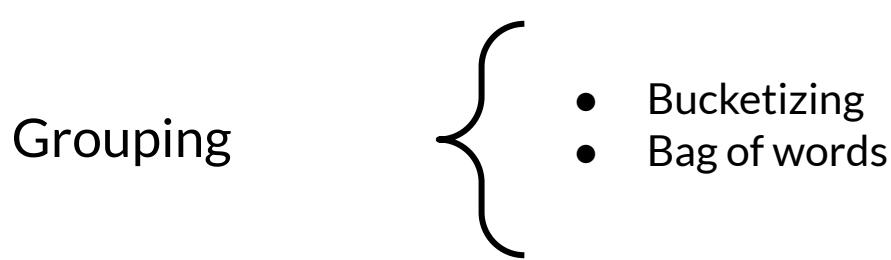
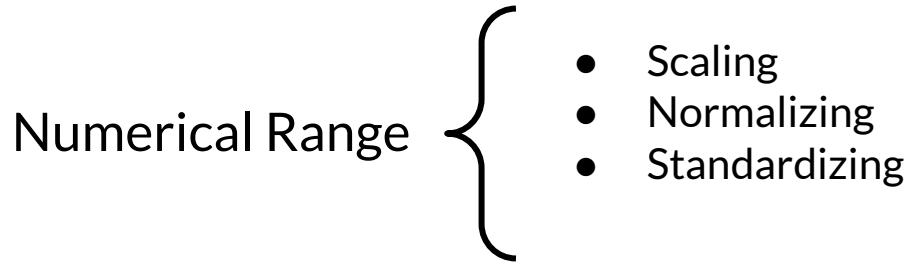
## Feature Engineering Techniques

# Outline

- Feature Scaling
- Normalization and Standardization
- Bucketizing / Binning
- Other techniques



# Feature engineering techniques



# Scaling

- Converts values from their natural range into a prescribed range
  - E.g. Grayscale image pixel intensity scale is [0,255] usually rescaled to [-1,1]

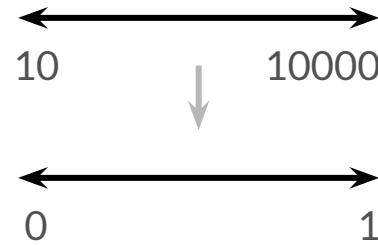
```
image = (image - 127.5) / 127.5
```

- Benefits
  - Helps neural nets converge faster
  - Do away with NaN errors during training
  - For each feature, the model learns the right weights

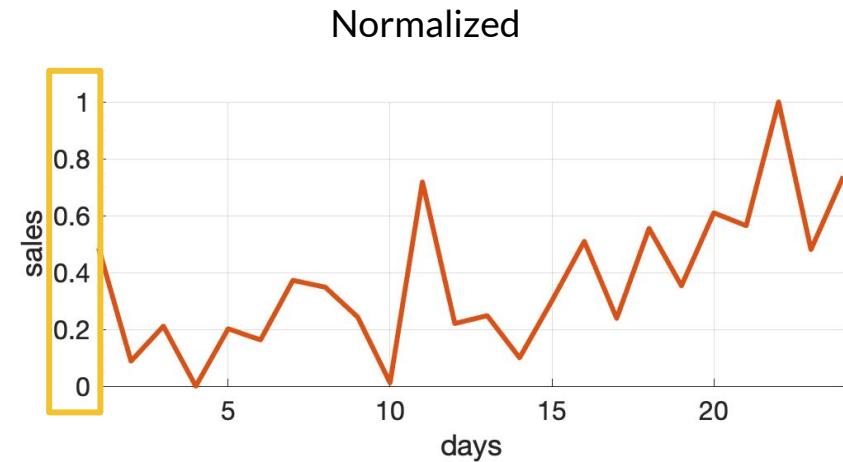
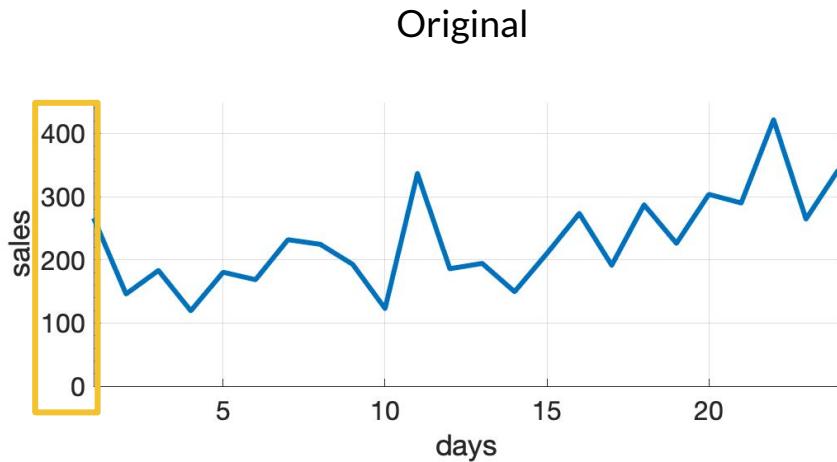
# Normalization

$$X_{\text{norm}} = \frac{X - X_{\min}}{X_{\max} - X_{\min}}$$

$$X_{\text{norm}} \in [0, 1]$$



# Normalization

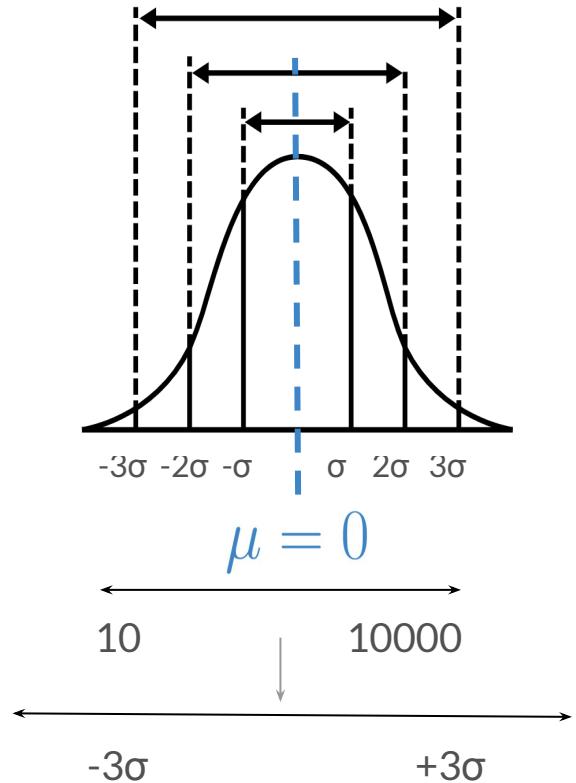


# Standardization (z-score)

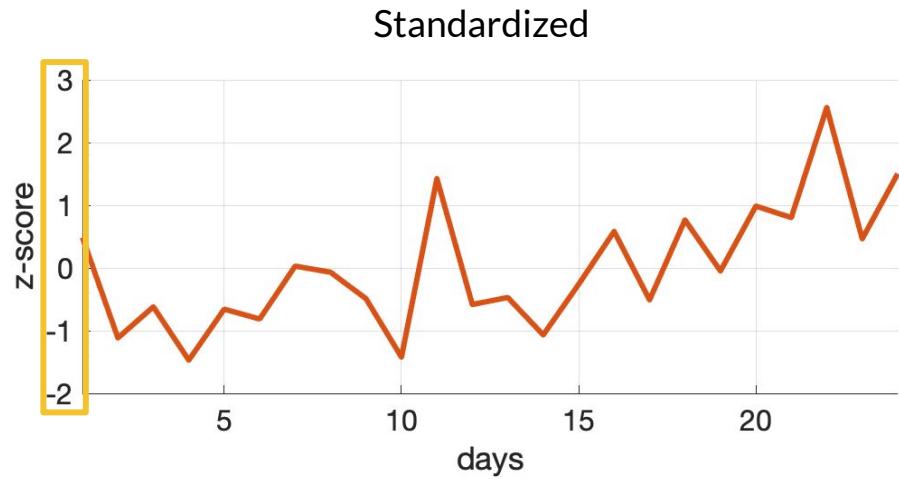
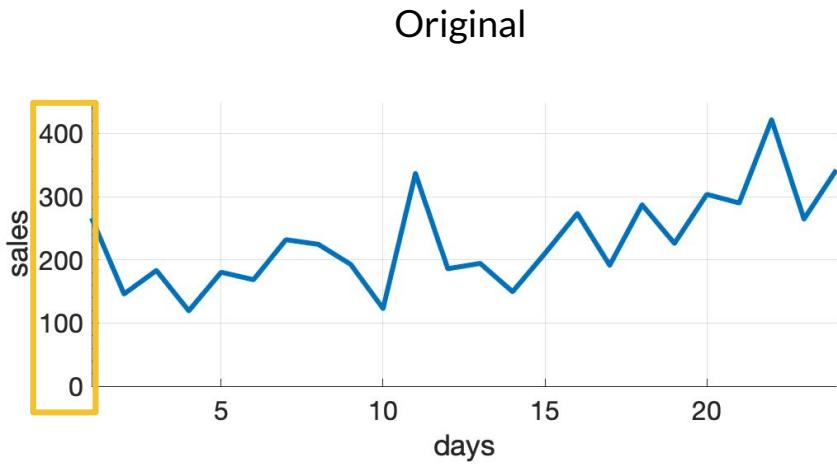
- Z-score relates the number of standard deviations away from the mean
- Example:

$$X_{\text{std}} = \frac{X - \mu}{\sigma} \quad (\text{z-score})$$

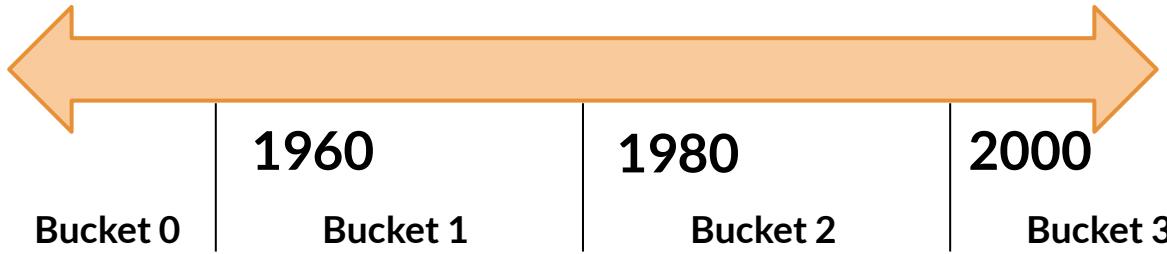
$$X_{\text{std}} \sim \mathcal{N}(0, \sigma)$$



# Standardization (z-score)

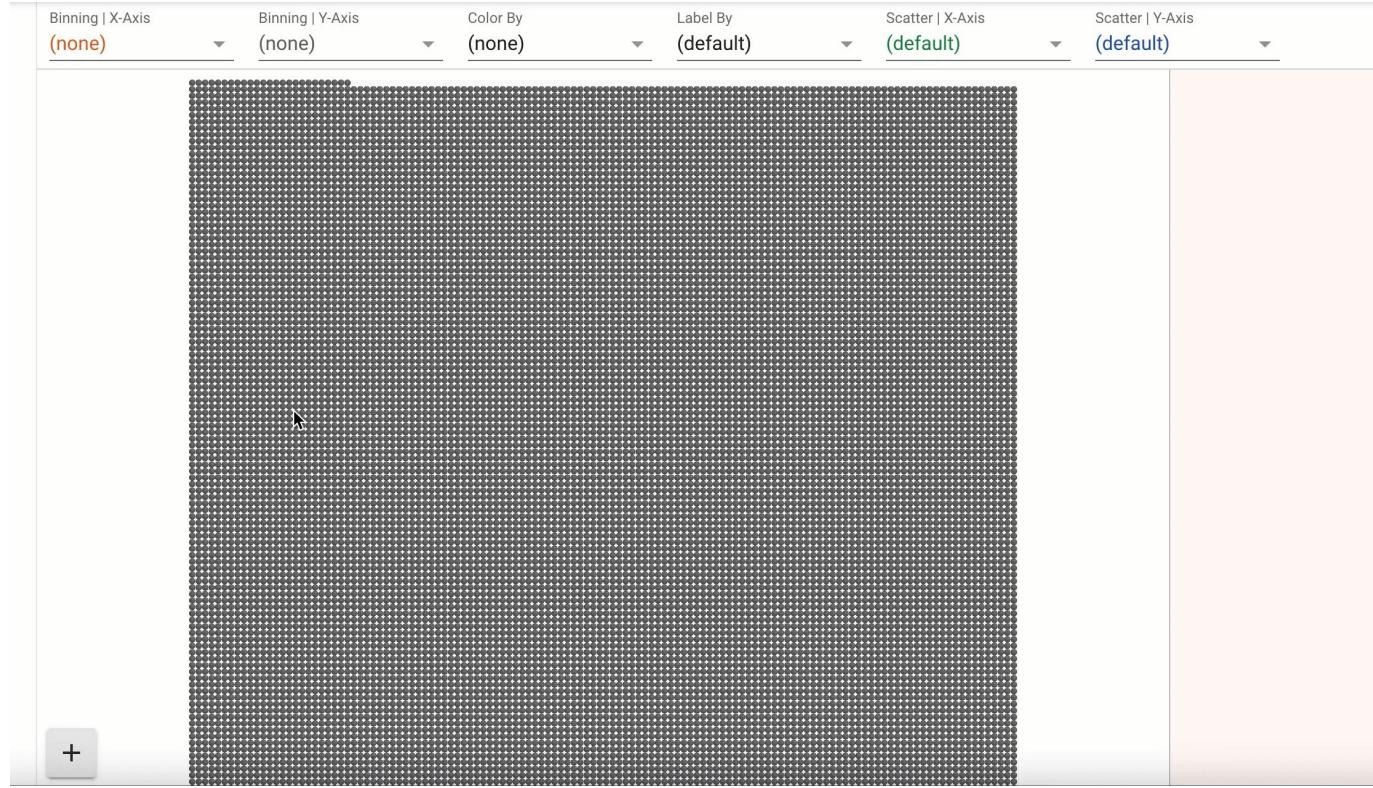


# Bucketizing / Binning



Date Range	Represented as...
< 1960	[1, 0, 0, 0]
$\geq 1960$ but $< 1980$	[0, 1, 0, 0]
$\geq 1980$ but $< 2000$	[0, 0, 1, 0]
$\geq 2000$	[0, 0, 0, 1]

# Binning with Facets



# Other techniques

Dimensionality reduction in embeddings

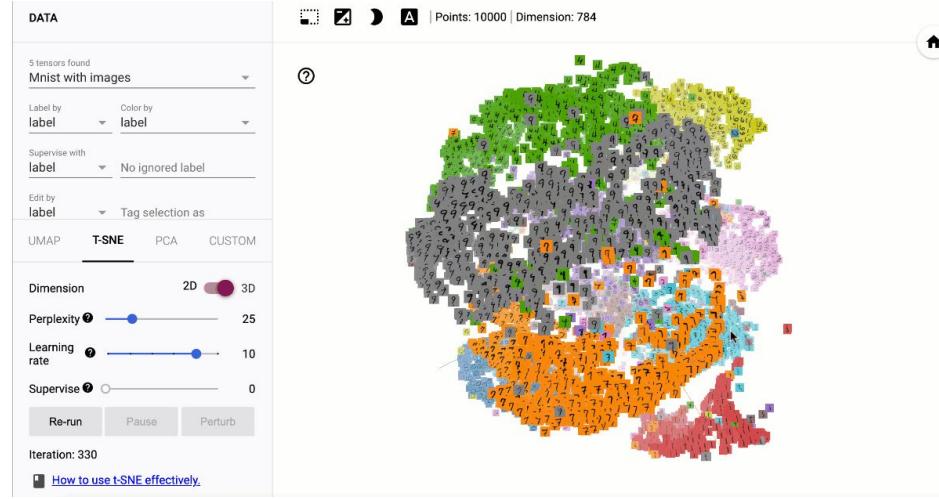
- 
- Principal component analysis (PCA)
  - t-Distributed stochastic neighbor embedding (t-SNE)
  - Uniform manifold approximation and projection (UMAP)

Feature crossing

# TensorFlow embedding projector

- Intuitive exploration of high-dimensional data
- Visualize & analyze
- Techniques
  - PCA
  - t-SNE
  - UMAP
  - Custom linear projections
- Ready to play

@projector.tensorflow.org



# Key points

- Feature engineering:
  - Prepares, tunes, transforms, extracts and constructs features.
- Feature engineering is key for model refinement
- Feature engineering helps with ML analysis

# Feature Engineering

---

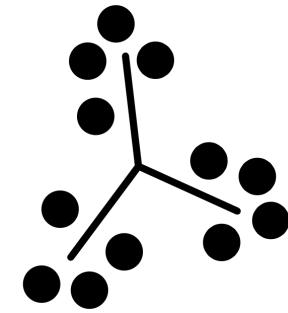


DeepLearning.AI

# Feature Crosses

# Outline

- Feature crosses
- Encoding features



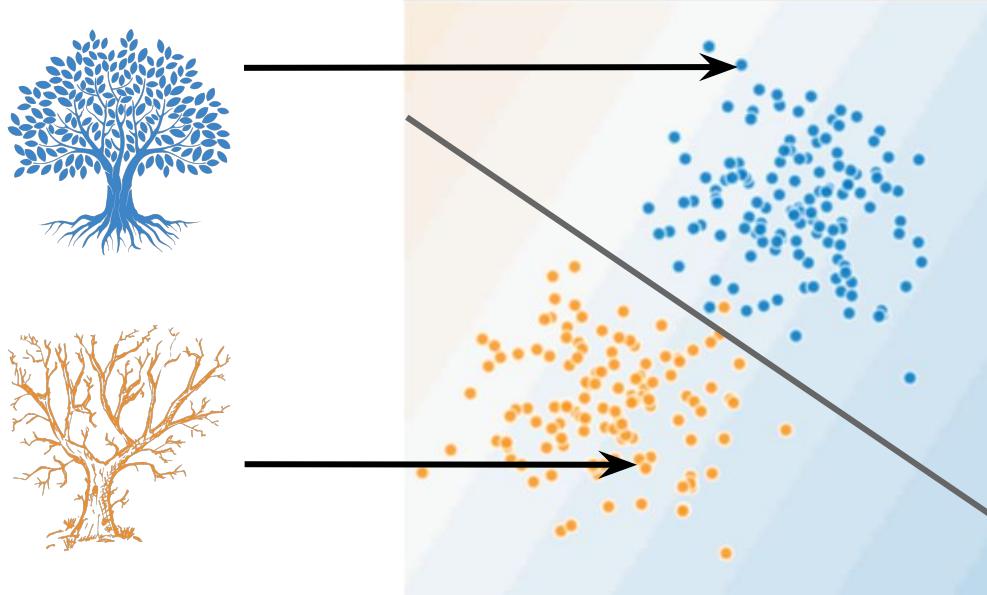
# Feature crosses

We can create many different kinds of feature crosses



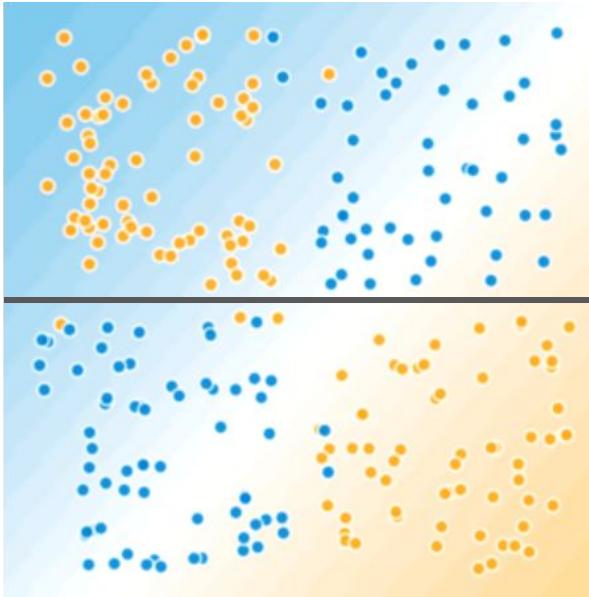
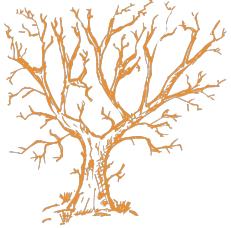
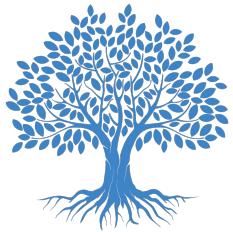
- Combines multiple features together into a new feature
- Encodes nonlinearity in the feature space, or encodes the same information in fewer features
- $[A \times B]$ : multiplying the values of two features
- $[A \times B \times C \times D \times E]$ : multiplying the values of 5 features
- [Day of week, Hour] => [Hour of week]

# Encoding features



- healthy trees
  - sick trees
- Classification boundary

# Need for encoding non-linearity



- healthy trees
  - sick trees
- Classification boundary

# Census dataset



# Key points

- Feature crossing: synthetic feature encoding nonlinearity in feature space.
- Feature coding: transforming categorical to a continuous variable.



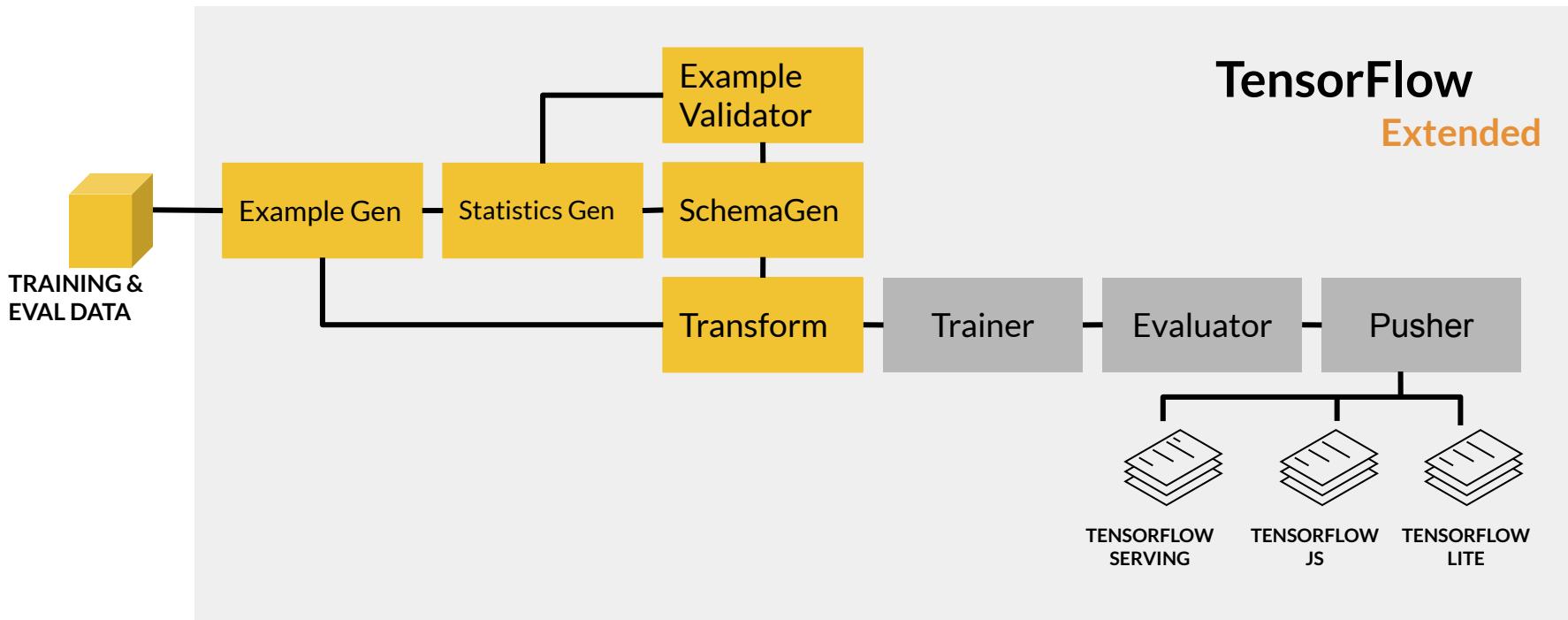
DeepLearning.AI

# Feature Transformation At Scale

---

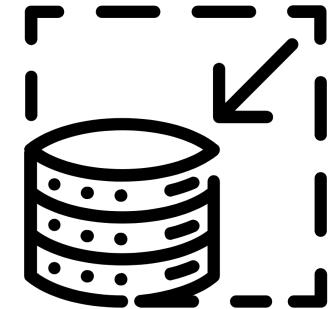
## Preprocessing Data At Scale

# ML Pipeline



# Outline

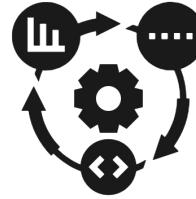
- Inconsistencies in feature engineering
- Preprocessing granularity
- Pre-processing training dataset
- Optimizing instance-level transformations
- Summarizing the challenges



# Preprocessing data at scale



Real-world models:  
terabytes of data



Large-scale data  
processing frameworks

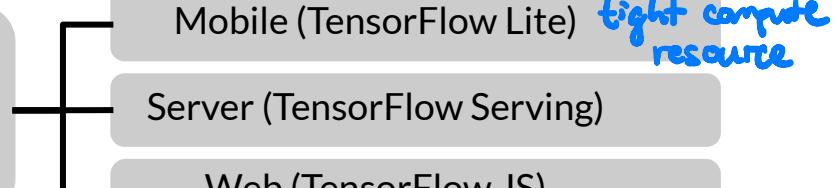


Consistent transforms  
between training &  
serving

# Inconsistencies in feature engineering

Training & serving code paths are different

Diverse deployments scenarios



Risks of introducing training-serving skews  
*different code paths*

Skews will lower the performance of your serving model

# Preprocessing granularity

Transformations	
Instance-level <i>(for serving)</i>	Full-pass <i>for training</i>
Clipping <i>Individual</i>	Minimax
Multiplying	Standard scaling
Expanding features	Bucketizing
etc.	etc.

# When do you transform?

Pre-processing training dataset

Pros	Cons
Run-once	Transformations <u>reproduced at serving</u>
Compute on entire dataset	<u>Slower iterations</u>

# How about 'within' a model?

Transforming within the model	
Pros	Cons
Easy iterations	<u>Expensive</u> transforms
Transformation guarantees	<u>Long model latency</u>
	Transformations per batch: <u>skew</u>

*compute resource  
No GPU for serving*

# Why transform per batch?

- For example, normalizing features by their average
- Access to a single batch of data, not the full dataset
- Ways to normalize per batch
  - Normalize by average within a batch
  - Precompute average and reuse it during normalization

# Optimizing instance-level transformations

- Indirectly affect training efficiency
- Typically accelerators sit idle while the CPUs transform
- Solution:
  - Prefetching transforms for better accelerator efficiency

*parallel processing*

# Summarizing the challenges

- Balancing predictive performance
- Full-pass transformations on training data
- Optimizing instance-level transformations for better training efficiency  
(GPUs, TPUs, ...) *Methods : prefetching*

# Key points

- Inconsistent data affects the accuracy of the results
- Need for scaled data processing frameworks to process large datasets in an efficient and distributed manner

# Preprocessing Data At Scale

---

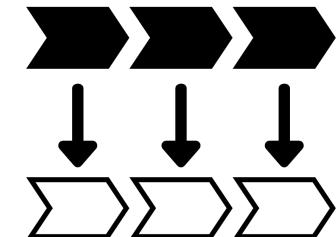


DeepLearning.AI

# TensorFlow Transform

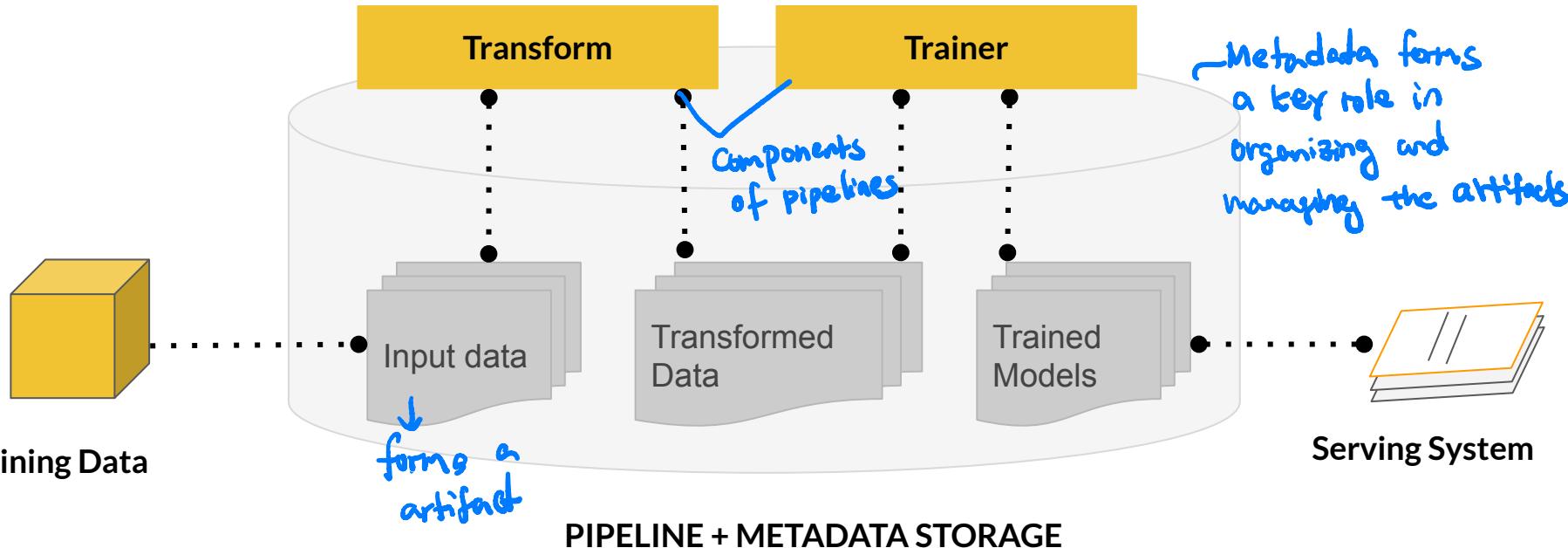
# Outline

- Going deeper
- Benefits of using TensorFlow **Transform**
- Applies feature transformations
- tf.Transform Analyzers

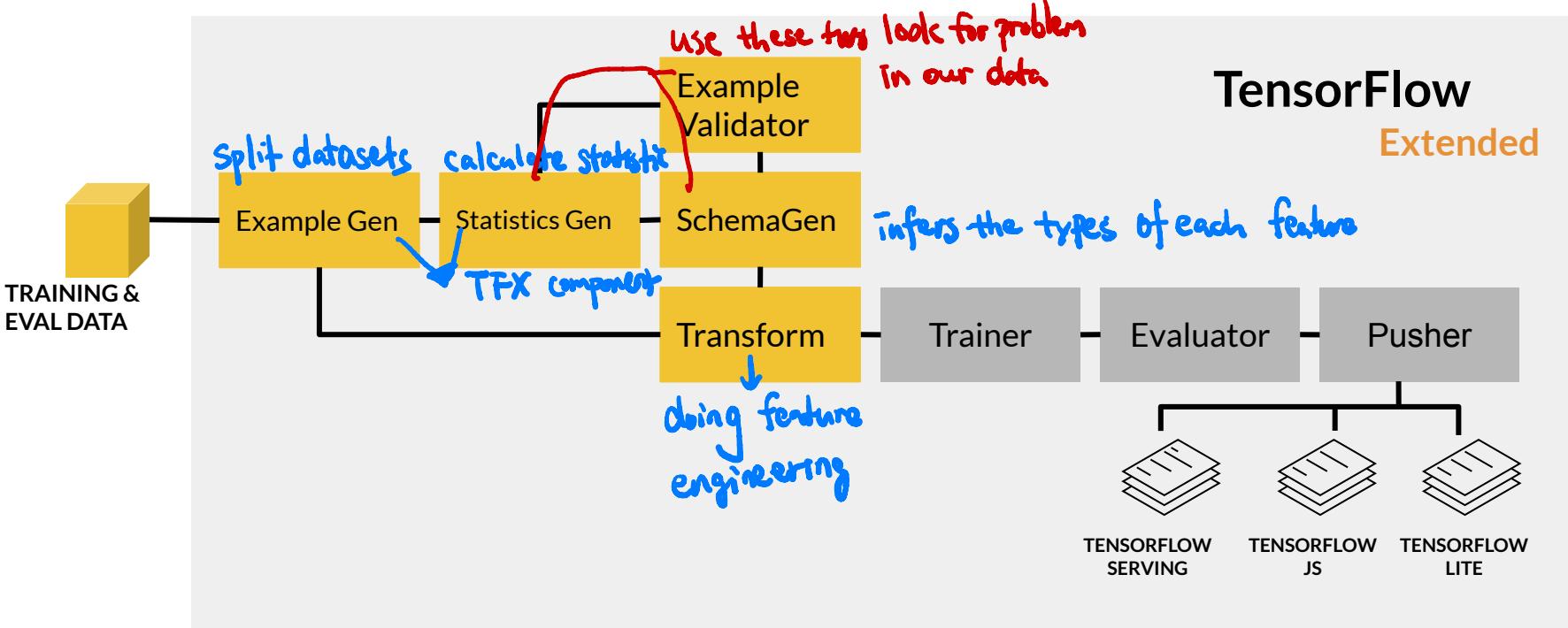


# Enter tf.Transform

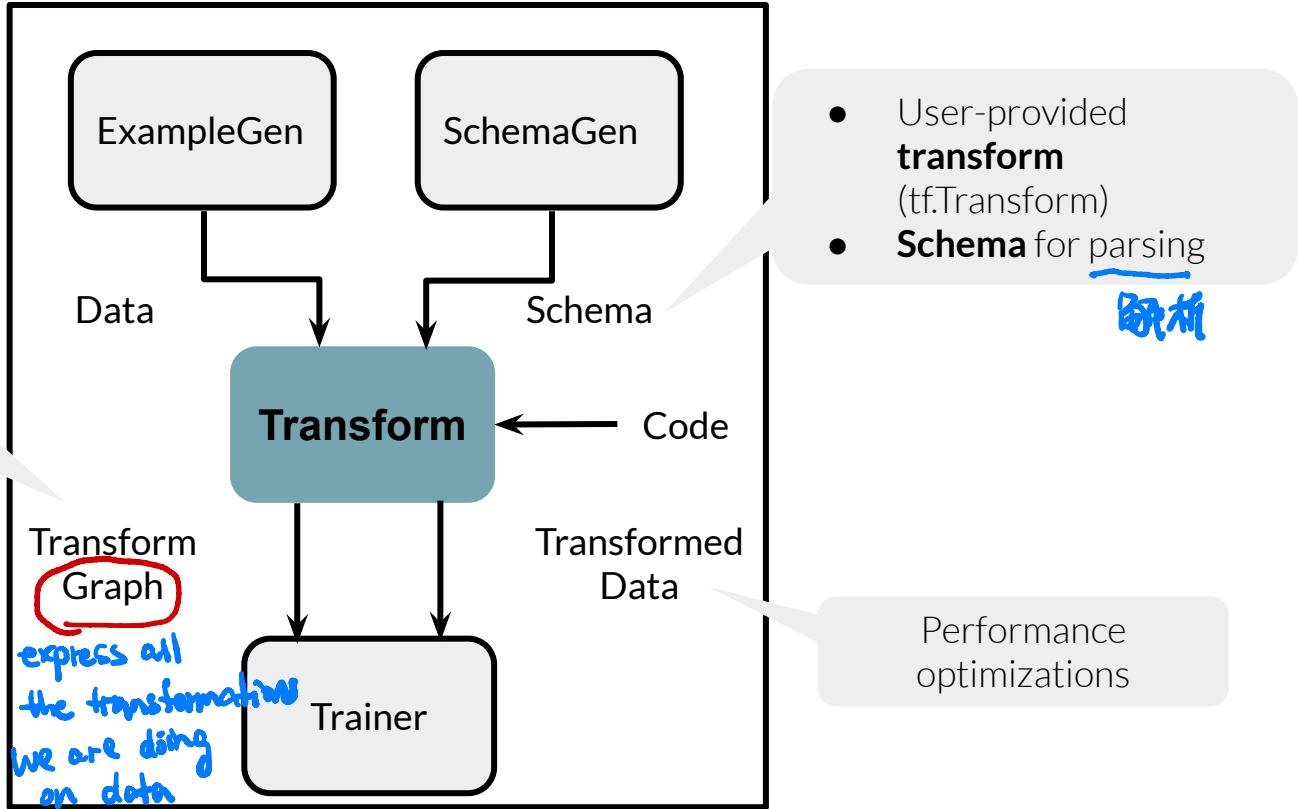
Lineage of artifacts



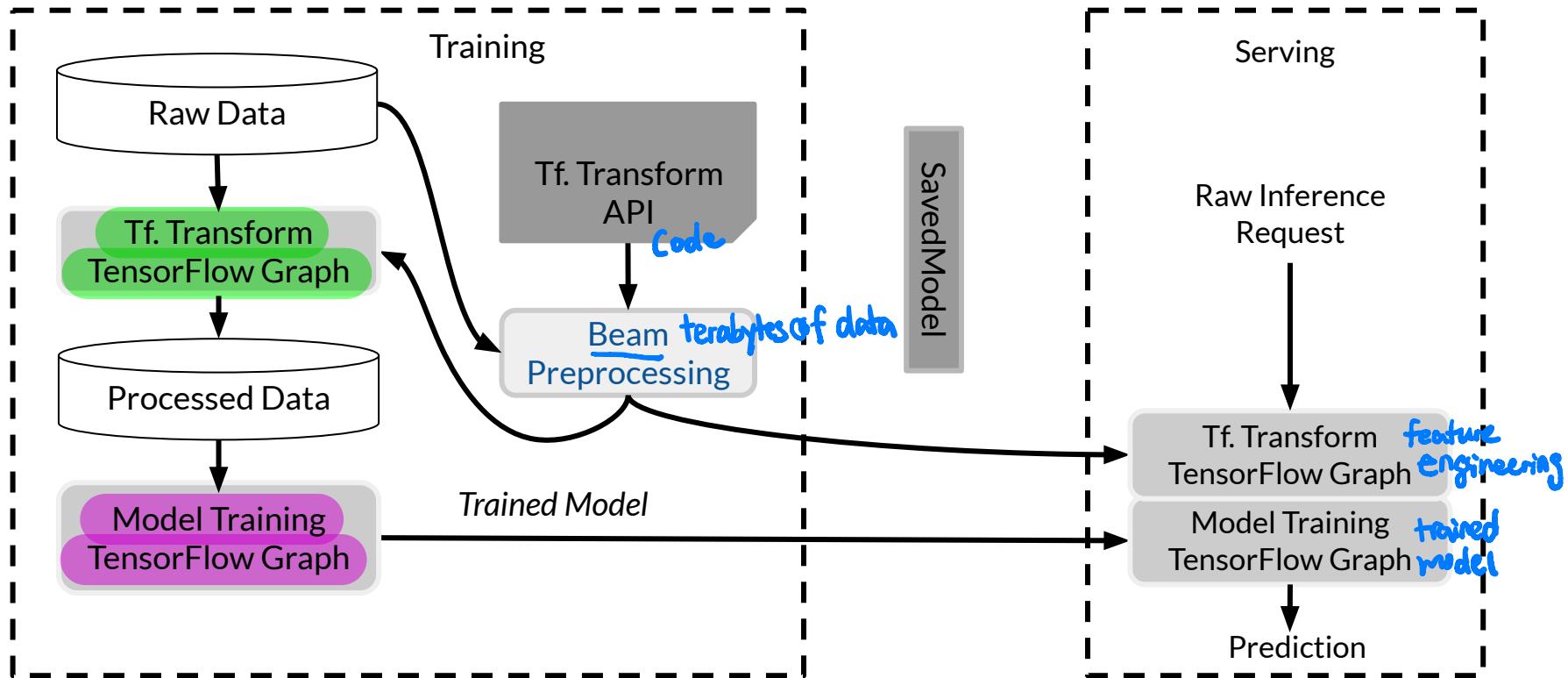
# Inside TensorFlow Extended



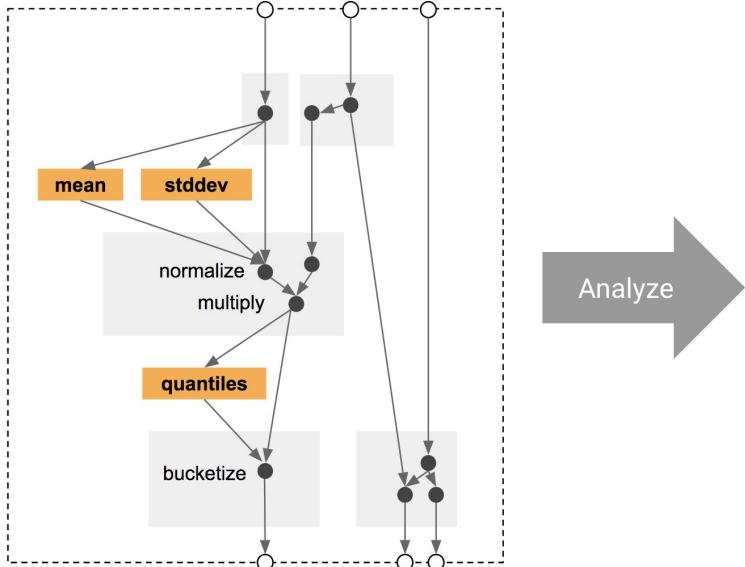
# tf.Transform layout



# tf.Transform: Going deeper



# tf.Transform Analyzers



They behave like TensorFlow Ops, but run only once during training

For example:

*tft.min computes the minimum of a tensor over the training dataset*

# How Transform **applies** feature transformations



feature Engineering + Analyse → transform individual examples  
(collect the constants) without making a pass over our entire dataset → no skew

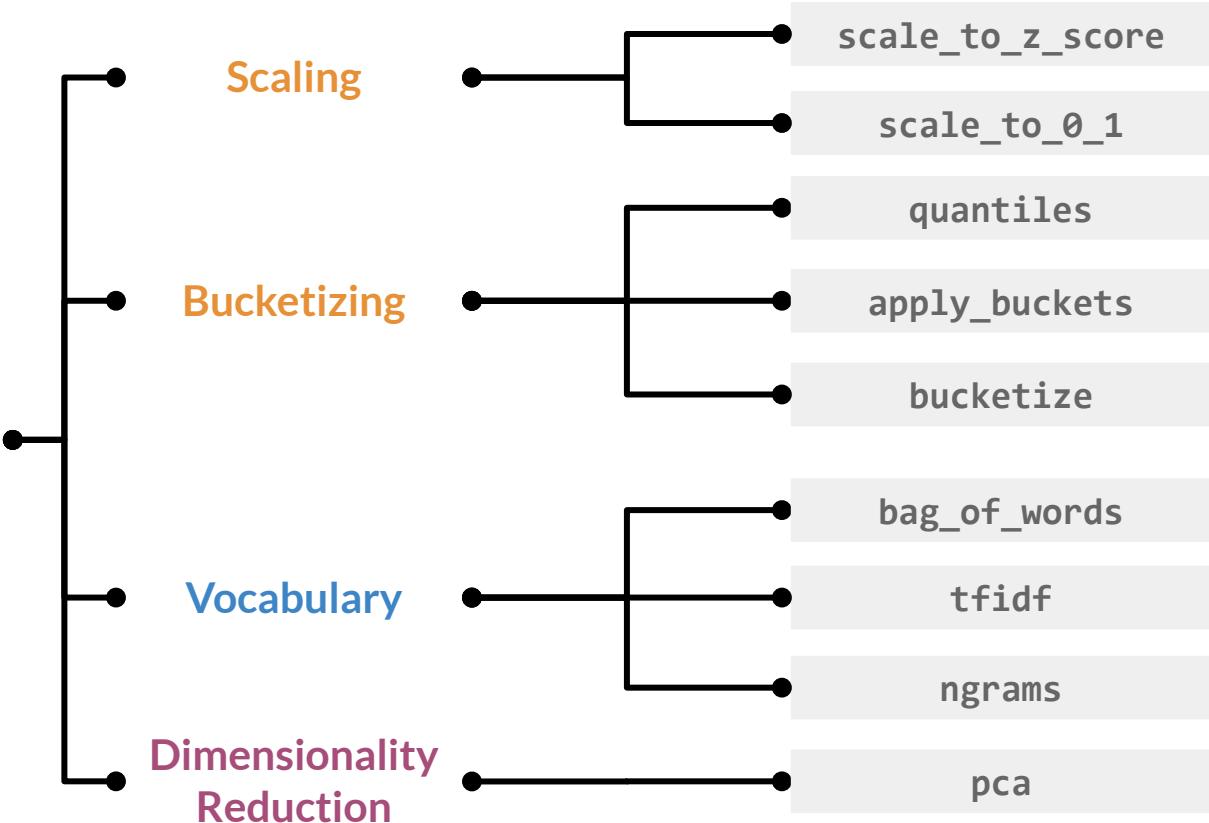


# Benefits of using tf.Transform

- Emitted tf.Graph holds all necessary constants and transformations
- Focus on data preprocessing only at training time
- Works in-line during both training and serving
- No need for preprocessing code at serving time
- Consistently applied transformations irrespective of deployment platform

# Analyzers framework

## tf.Transform Analyzers



# tf.Transform preprocessing\_fn

```
def preprocessing_fn(inputs):
    ...
    for key in DENSE_FLOAT_FEATURE_KEYS:
        outputs[key] = tft.scale_to_z_score(inputs[key])
    for key in VOCAB_FEATURE_KEYS:
        outputs[key] = tft.vocabulary(inputs[key], vocab_filename=key)
    for key in BUCKET_FEATURE_KEYS:
        outputs[key] = tft.bucketize(inputs[key], FEATURE_BUCKET_COUNT)
```

# Commonly used imports

```
import tensorflow as tf
import apache_beam as beam → distribute processing across a cluster
import apache_beam.io.iobase          with io

import tensorflow_transform as tft
import tensorflow_transform.beam as tft_beam
```



DeepLearning.AI

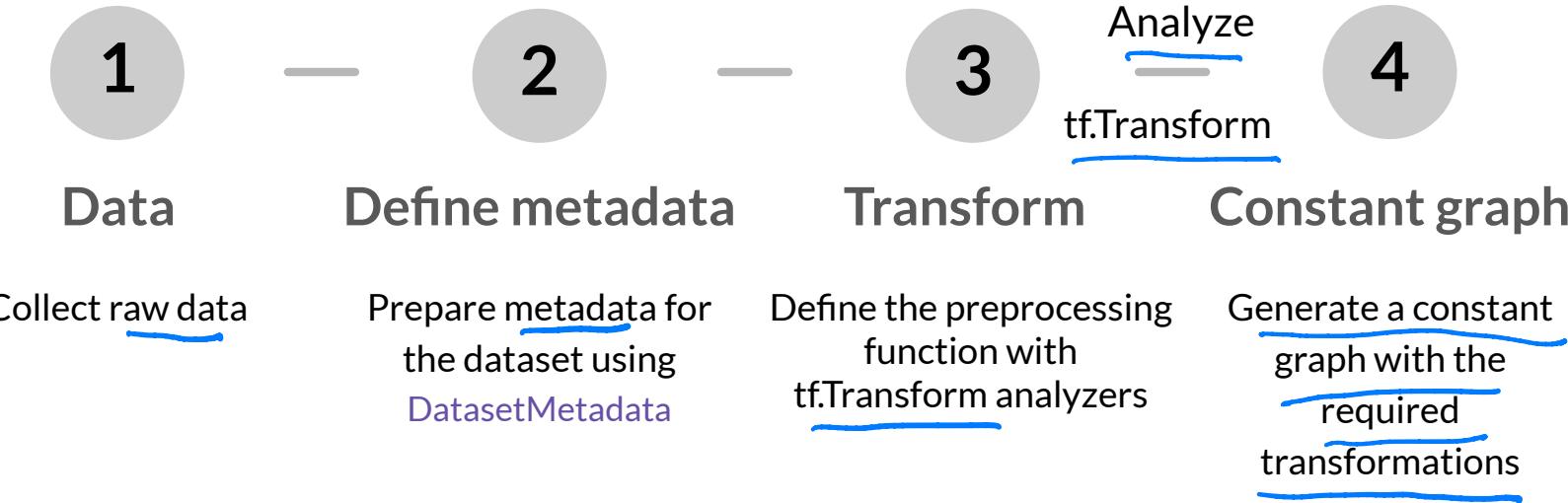
# Feature Transformation At Scale

---

## Hello World with tf.Transform

# Hello world with tf.Transform

do all of this in a reproducible way  
consistent



# Collect raw samples (Data)

```
[  
  {'x': 1, 'y': 1, 's': 'hello'},  
  {'x': 2, 'y': 2, 's': 'world'},  
  {'x': 3, 'y': 3, 's': 'hello'}]  
]
```

# Inspect data and prepare metadata (Data)

type of data

```
from tensorflow_transform.tf_metadata import (
    dataset_metadata, dataset_schema)

raw_data_metadata = dataset_metadata.DatasetMetadata(
    dataset_schema.from_feature_spec({
        'y': tf.io.FixedLenFeature([], tf.float32),
        'x': tf.io.FixedLenFeature([], tf.float32),
        's': tf.io.FixedLenFeature([], tf.string)
    }))
```

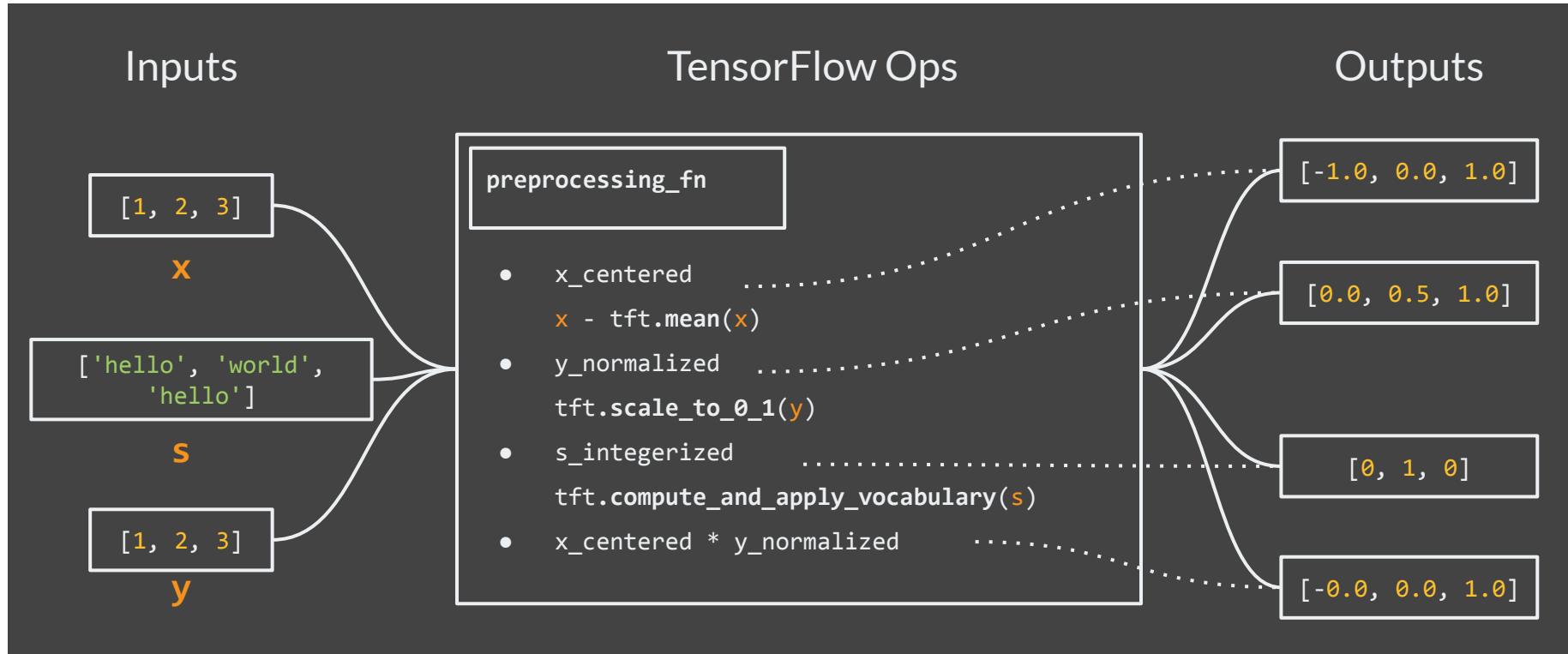
# Preprocessing data (Transform)

```
def preprocessing_fn(inputs):
    """Preprocess input columns into transformed columns."""
    x, y, s = inputs['x'], inputs['y'], inputs['s']
    x_centered = x - tft.mean(x)
    y_normalized = tft.scale_to_0_1(y)
    s_integerized = tft.compute_and_apply_vocabulary(s)
    x_centered_times_y_normalized = (x_centered * y_normalized)
```

# Preprocessing data (Transform)

```
return {  
    'x_centered': x_centered,  
    'y_normalized': y_normalized,  
    's_integerized': s_integerized,  
    'x_centered_times_y_normalized': x_centered_times_y_normalized,  
}
```

# Tensors in... tensors out



# Running the pipeline

```
def main():
    with tft_beam.Context(temp_dir=tempfile.mkdtemp()):
        transformed_dataset, transform_fn = (
            (raw_data, raw_data_metadata) | tft_beam.AnalyzeAndTransformDataset(
                preprocessing_fn))
    use this to process preprocessing_fn
```

# Running the pipeline

```
transformed_data, transformed_metadata = transformed_dataset
print('\nRaw data:\n{}\n'.format(pprint.pformat(raw_data)))
print('Transformed data:\n{}'.format(pprint.pformat(transformed_data)))

if __name__ == '__main__':
    main()
```

# Before transforming with tf.Transform

```
# Raw data:  
[{'s': 'hello', 'x': 1, 'y': 1},  
 {'s': 'world', 'x': 2, 'y': 2},  
 {'s': 'hello', 'x': 3, 'y': 3}]
```

# After transforming with tf.Transform

```
# After transform
[{'s_integerized': 0,
 'x_centered': -1.0,
 'x_centered_times_y_normalized': -0.0,
 'y_normalized': 0.0},
 {'s_integerized': 1,
 'x_centered': 0.0,
 'x_centered_times_y_normalized': 0.0,
 'y_normalized': 0.5},
 {'s_integerized': 0,
 'x_centered': 1.0,
 'x_centered_times_y_normalized': 1.0,
 'y_normalized': 1.0}]
```

# Key points

- `tf.Transform` allows the pre-processing of input data and creating features
- `tf.Transform` allows defining pre-processing pipelines and their execution using large-scale data processing frameworks
- In a TFX pipeline, the Transform component implements feature engineering using TensorFlow Transform



DeepLearning.AI

# Feature Selection

---

# Feature Spaces

# Outline

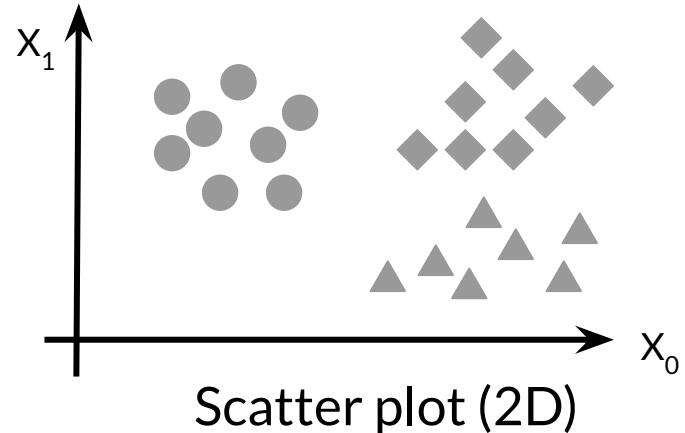
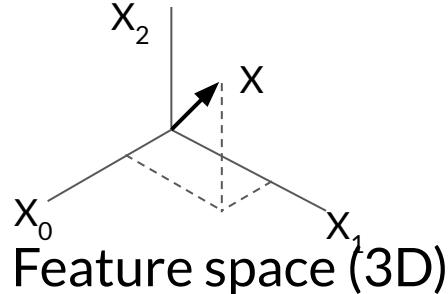
- Introduction to Feature Spaces
- Introduction to Feature Selection
- Filter Methods
- Wrapper Methods
- Embedded Methods

# Feature space

- N dimensional space defined by your N features
- Not including the target label

$$\mathbf{X} = \begin{bmatrix} X_0 \\ X_1 \\ \vdots \\ X_d \end{bmatrix}$$

Feature vector



# Feature space

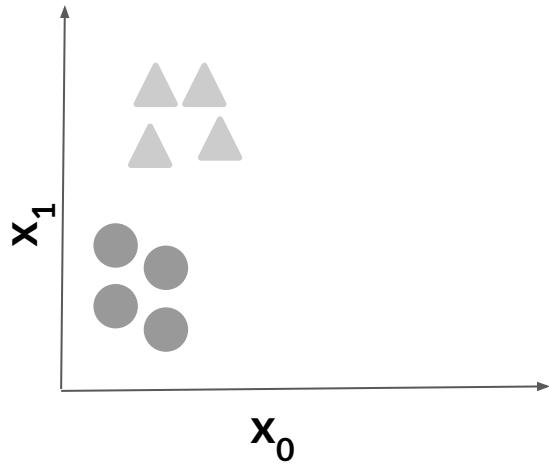


No. of Rooms $X_0$	Area $X_1$	Locality $X_2$	Price $Y$
5	1200 sq. ft	New York	\$40,000
6	1800 sq. ft	Texas	\$30,000

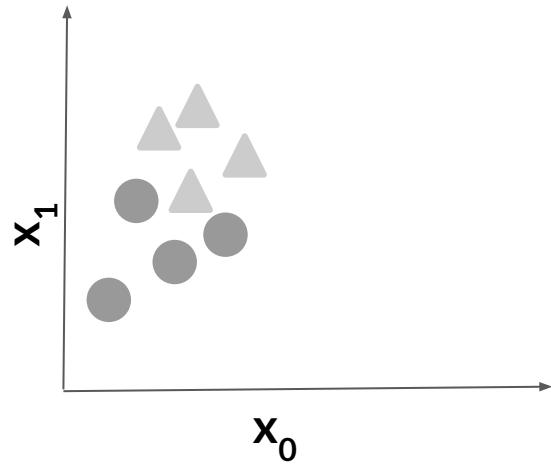
$$Y = f(X_0, X_1, X_2)$$

$f$  is your ML model acting on feature space  $X_0, X_1, X_2$

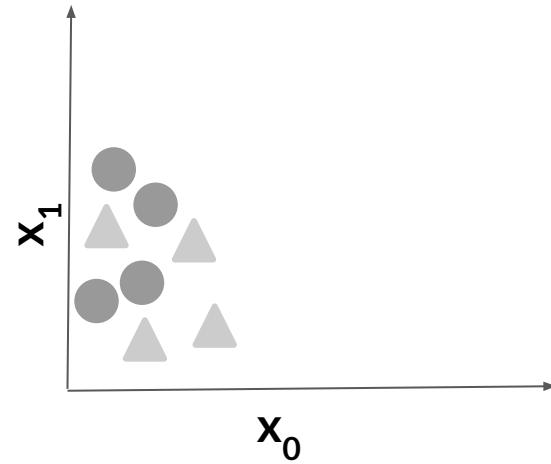
# 2D Feature space - Classification



Ideal

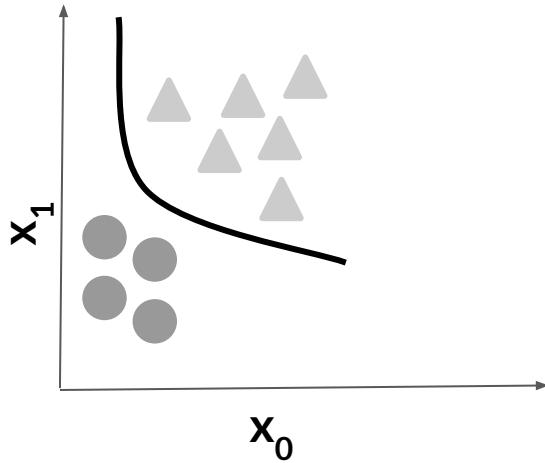


Realistic



Poor

# Drawing decision boundary



Model learns decision boundary

Boundary used to classify data points

# Feature space coverage

- Train/Eval datasets representative of the serving dataset
  - Same numerical ranges
  - Same classes
  - Similar characteristics for image data
  - Similar vocabulary, syntax, and semantics for NLP data

# Ensure feature space coverage

- Data affected by: seasonality, trend, drift.
- Serving data: new values in features and labels.
- Continuous monitoring, key for success!



DeepLearning.AI

# Feature Selection

---

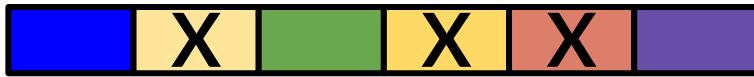
# Feature Selection

# Feature selection

All Features



Feature selection

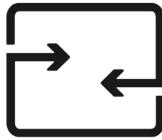


Useful features

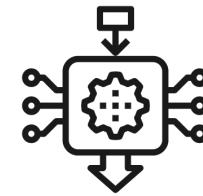


- Identify features that best represent the relationship
- Remove features that don't influence the outcome
- Reduce the size of the feature space
- Reduce the resource requirements and model complexity

# Why is feature selection needed?

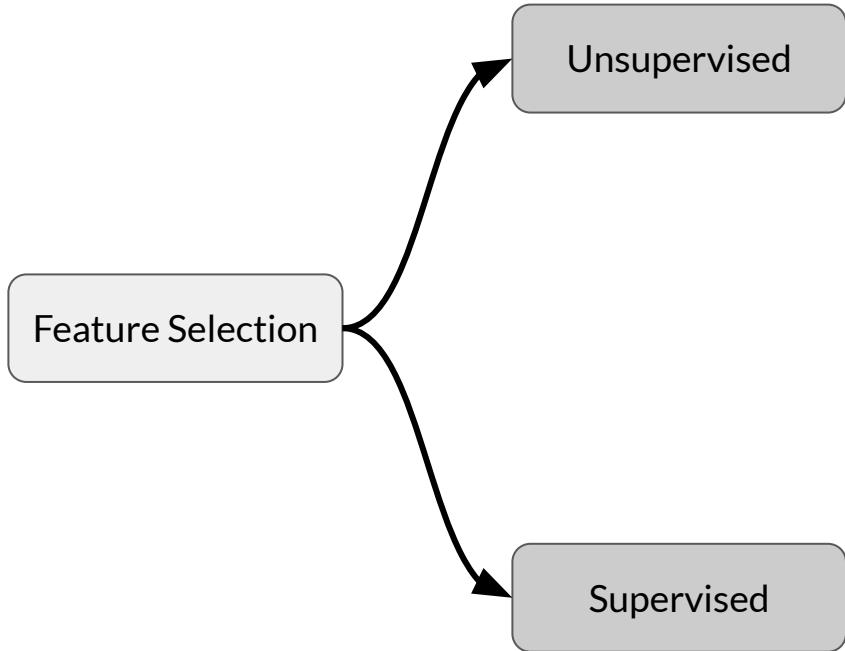


Reduce storage and I/O requirements



Minimize training and inference costs

# Feature selection methods



# Unsupervised feature selection

## 1. Unsupervised

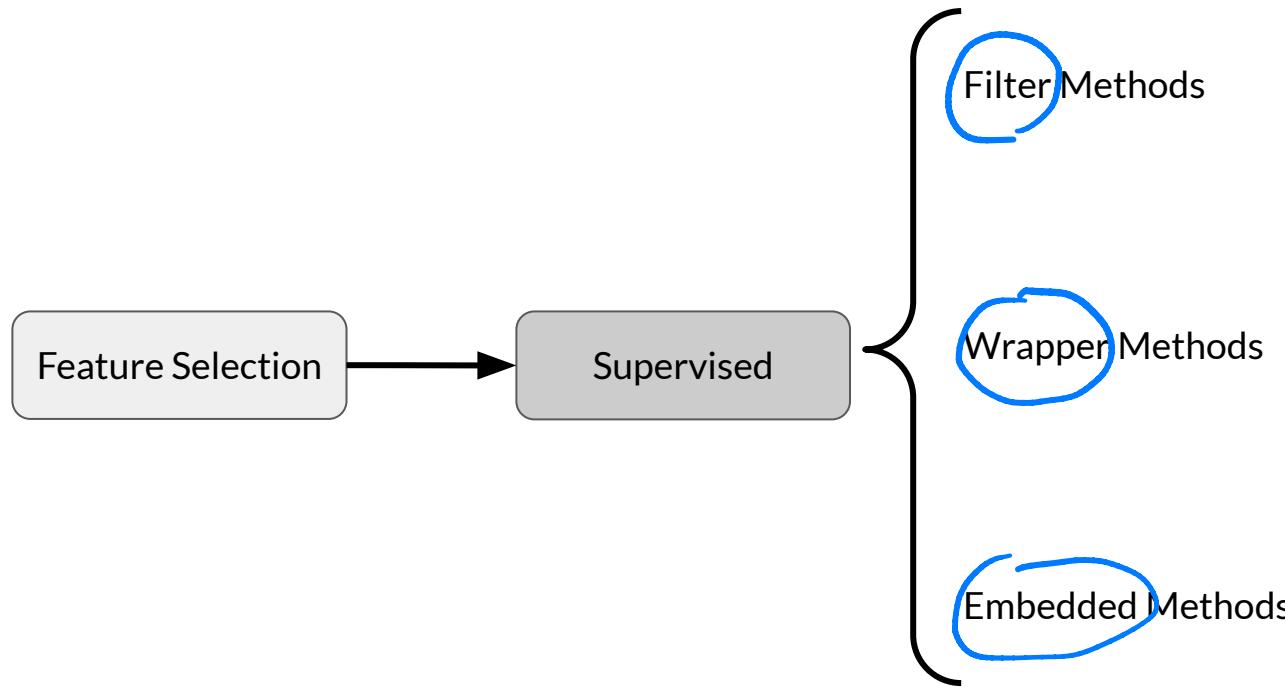
- Features-target variable relationship not considered
- Removes redundant features (correlation)

# Supervised feature selection

## 2. Supervised

- Uses features-target variable relationship
- Selects those contributing the most

# Supervised methods



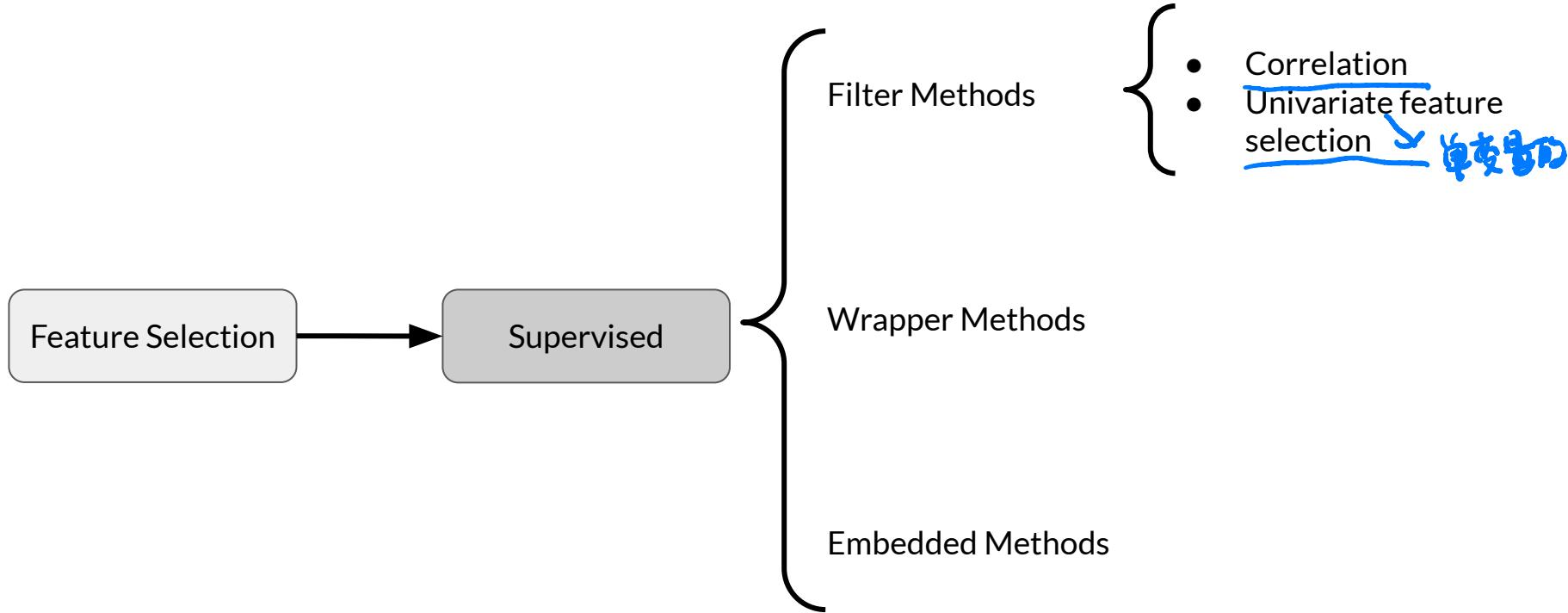
# Feature Selection



DeepLearning.AI

## Filter Methods

# Filter methods



# Filter methods

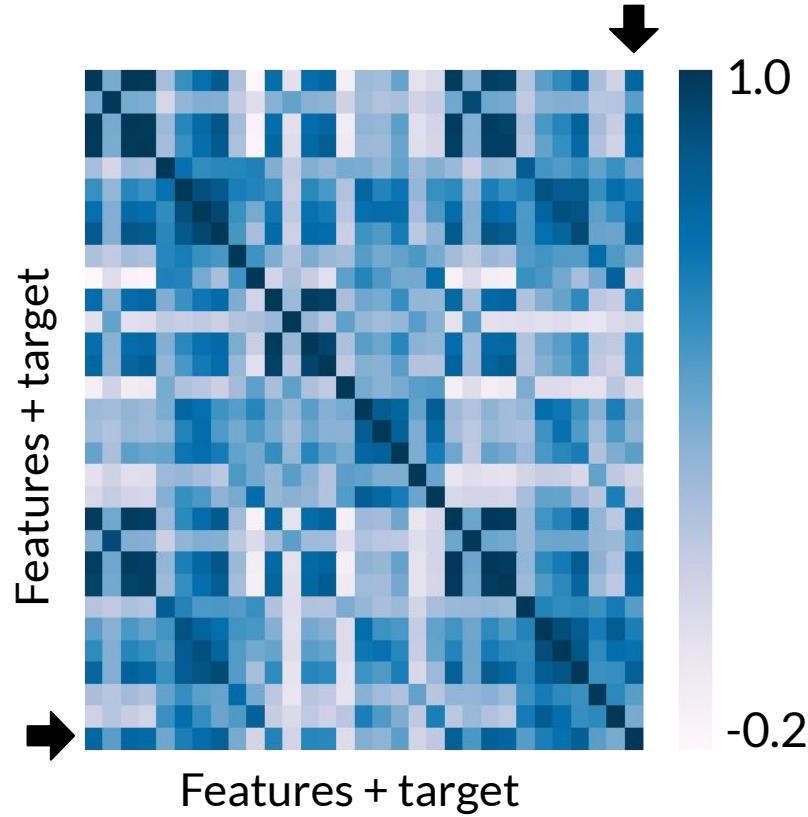
- Correlated features are usually redundant
  - Remove them!

Popular filter methods:

- Pearson Correlation
  - Between features, and between the features and the label
- Univariate Feature Selection

# Correlation matrix

- Shows how features are related:
  - To each other (Bad)
  - And with target variable (Good)
- Falls in the range  $[-1, 1]$ 
  - 1 High positive correlation
  - -1 High negative correlation



# Feature comparison statistical tests

- Pearson's correlation: Linear relationships
- Kendall Tau Rank Correlation Coefficient: Monotonic relationships & small sample size
- Spearman's Rank Correlation Coefficient: Monotonic relationships

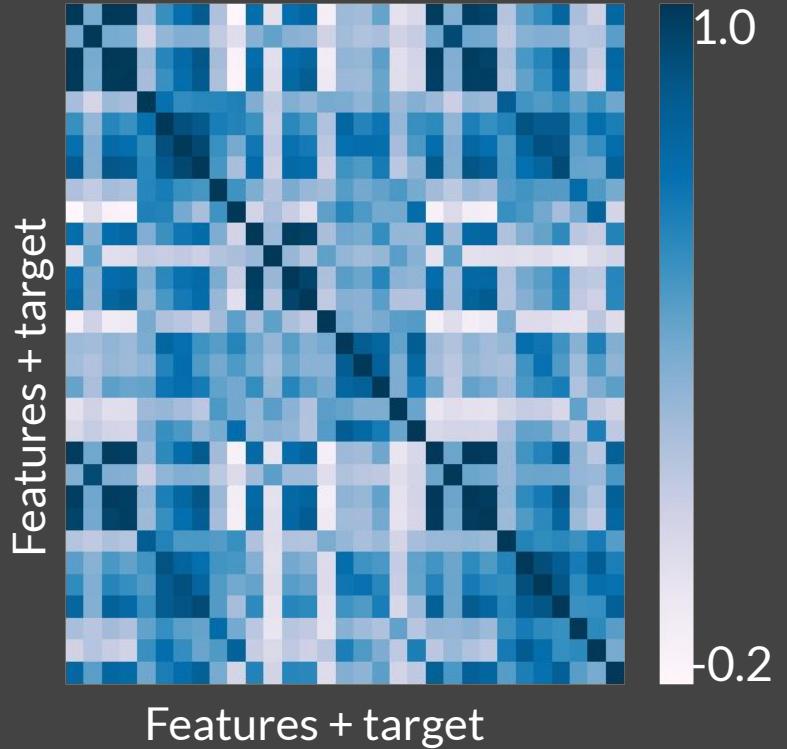
Other methods:

- Mutual information
- F-Test
- Chi-Squared test

# Determine correlation

```
# Pearson's correlation by default  
cor = df.corr()
```

```
plt.figure(figsize=(20,20))  
# Seaborn  
sns.heatmap(cor, annot=True, cmap=plt.cm.PuBu)  
plt.show()
```



# Selecting features

```
cor_target = abs(cor["diagnosis_int"])

# Selecting highly correlated features as potential features to eliminate
relevant_features = cor_target[cor_target>0.2]
```



# Performance table

Method	Feature Count	Accuracy	AUROC	Precision	Recall	F1 Score
All Features	30	0.967262	0.964912	0.931818	0.97619	0.953488
Correlation	21	0.974206	0.973684	0.953488	0.97619	0.964706

**Best Result**

# Univariate feature selection in SKLearn

SKLearn Univariate feature selection routines:

1. **SelectKBest**
2. SelectPercentile
3. GenericUnivariateSelect

*SelectKBest(chi2, k=20)*  
Best 20 features

*used in the methods*

Statistical tests available:

- Regression: f\_regression, mutual\_info\_regression
- Classification: chi2, f\_classif, mutual\_info\_classif

# SelectKBest implementation

```
def univariate_selection():

    X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
                                                       test_size = 0.2, stratify=Y, random_state = 123)

    X_train_scaled = StandardScaler().fit_transform(X_train)
    X_test_scaled = StandardScaler().fit_transform(X_test)

    min_max_scaler = MinMaxScaler()
    Scaled_X = min_max_scaler.fit_transform(X_train_scaled)

    selector = SelectKBest(chi2, k=20) # Use Chi-Squared test
    X_new = selector.fit_transform(Scaled_X, Y_train)

    feature_idx = selector.get_support()
    feature_names = df.drop("diagnosis_int", axis = 1 ).columns[feature_idx]

    return feature_names
```

# Performance table

Method	Feature Count	Accuracy	AUROC	Precision	Recall	F1 Score
All Features	30	0.967262	0.964912	0.931818	0.97619	0.953488
Correlation	21	0.974206	0.973684	0.953488	0.97619	0.964706
Univariate ( $\text{Chi}^2$ )	20	0.960317	0.95614	0.91111	0.97619	0.94252

**Best Result**

# Feature Selection

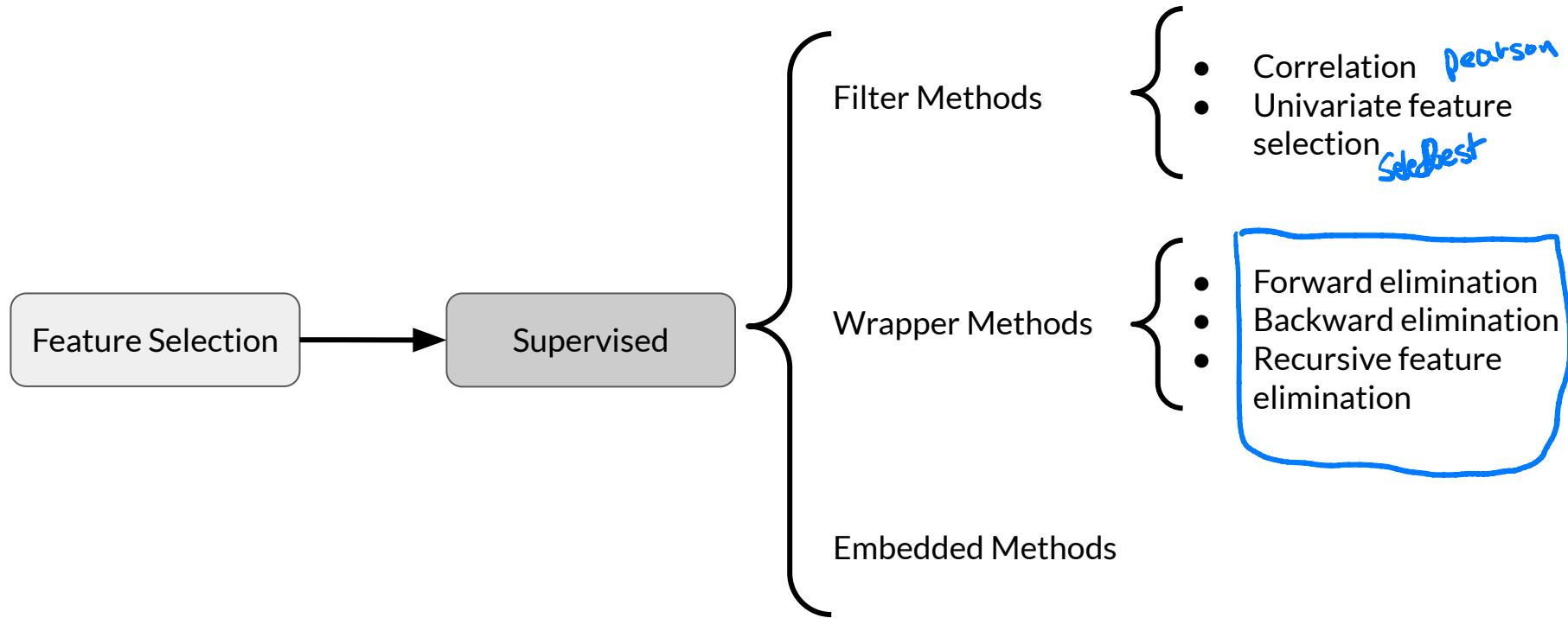
---



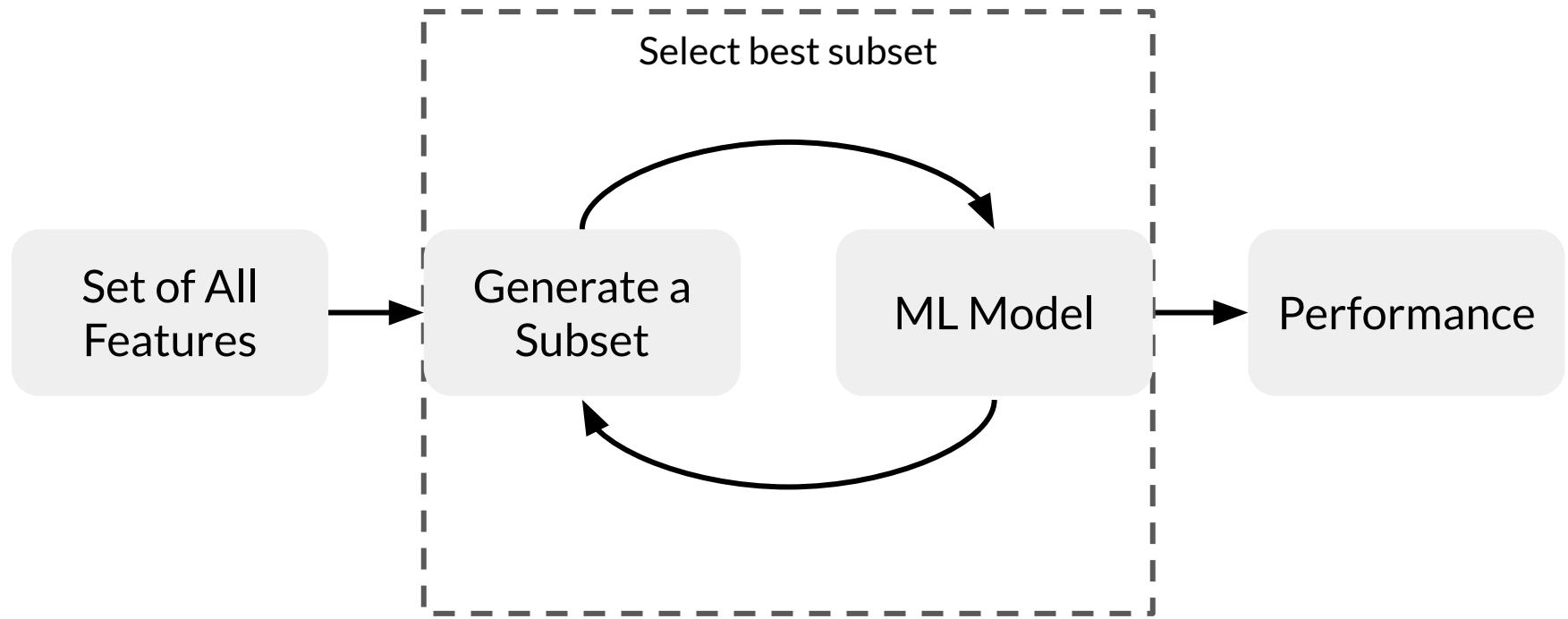
DeepLearning.AI

# Wrapper Methods

# Wrapper methods



# Wrapper methods



# Wrapper methods

Popular wrapper methods

1. Forward Selection
2. Backward Selection
3. Recursive Feature Elimination

# Forward selection

1. Iterative, greedy method
2. Starts with 1 feature
3. Evaluate model performance when **adding** each of the additional features, one at a time
4. Add next feature that gives the best performance
5. Repeat until there is no improvement

# Backward elimination

1. Start with all features
2. Evaluate model performance when **removing** each of the included features, one at a time
3. Remove next feature that gives the best performance
4. Repeat until there is no improvement

# Recursive feature elimination (RFE)

1. Select a model to use for evaluating feature importance
2. Select the desired number of features
3. Fit the model
4. Rank features by importance
5. Discard least important features
6. Repeat until the desired number of features remains

# Recursive feature elimination

```
def run_rfe():

    X_train, X_test, y_train, y_test = train_test_split(X,Y, test_size = 0.2, random_state = 0)

    X_train_scaled = StandardScaler().fit_transform(X_train)
    X_test_scaled = StandardScaler().fit_transform(X_test)

    model = RandomForestClassifier(criterion='entropy', random_state=47)
    rfe = RFE(model, 20)
    rfe = rfe.fit(X_train_scaled, y_train)

    feature_names = df.drop("diagnosis_int",axis = 1 ).columns[rfe.get_support()]
    return feature_names

rfe_feature_names = run_rfe()

rfe_eval_df = evaluate_model_on_features(df[rfe_feature_names], Y)
rfe_eval_df.head()
```

# Performance table

Method	Feature Count	Accuracy	AUROC	Precision	Recall	F1 Score
All Features	30	0.96726	0.96491	0.931818	0.97619	0.953488
Correlation	21	0.97420	0.97368	0.9534883	0.97619	0.964705
Univariate ( $\text{Chi}^2$ )	20	0.96031	0.95614	0.91111	0.97619	0.94252
Recursive Feature Elimination	20	0.97420	0.97368	0.953488	0.97619	0.964706

**Best Result**

# Feature Selection

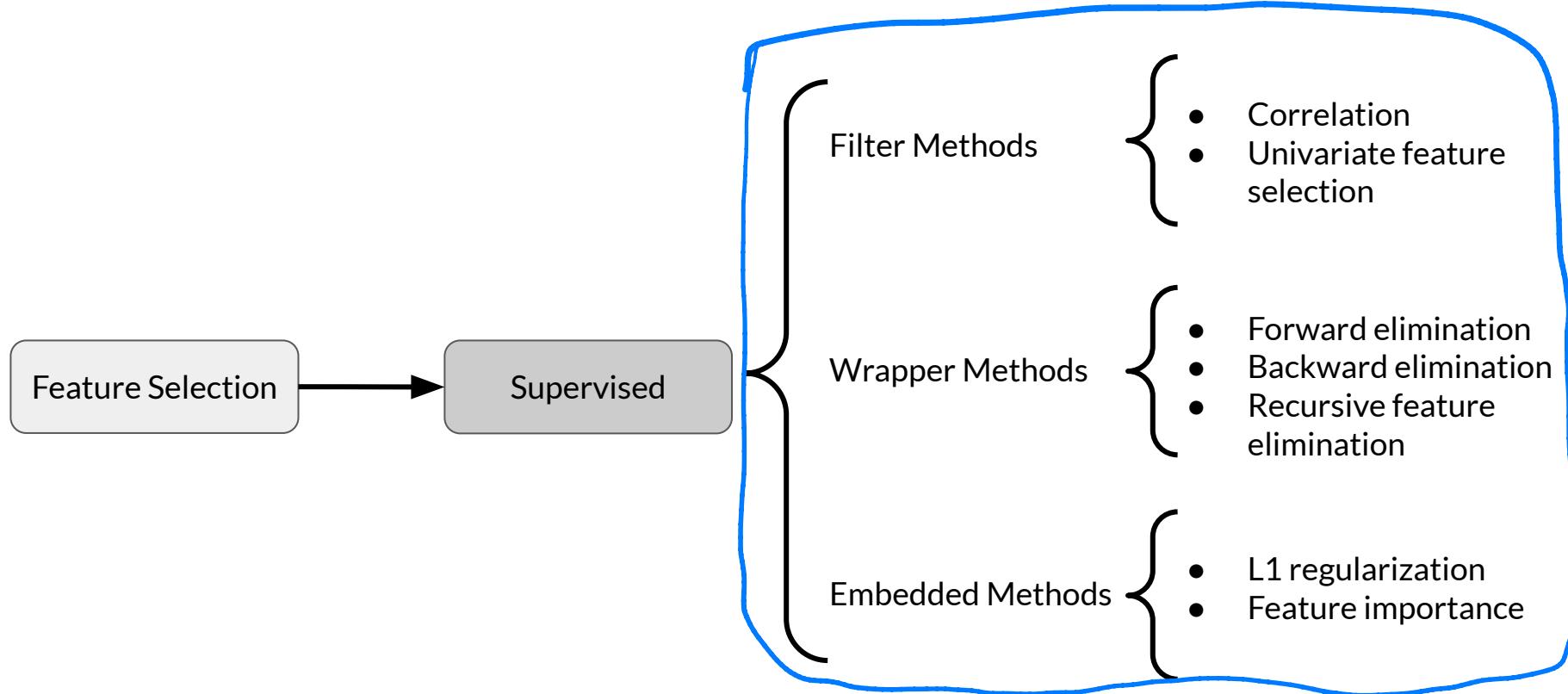
---



DeepLearning.AI

## Embedded Methods

# Embedded methods



# Feature importance

- Assigns scores for each feature in data
- Discard features scored lower by feature importance

# Feature importance with SKLearn

- Feature Importance class is in-built in Tree Based Models (eg., `RandomForestClassifier`)
- Feature importance is available as a property `feature_importances_`
- *We can then use `SelectFromModel` to select features from the trained model based on assigned feature importances.*

# Extracting feature importance

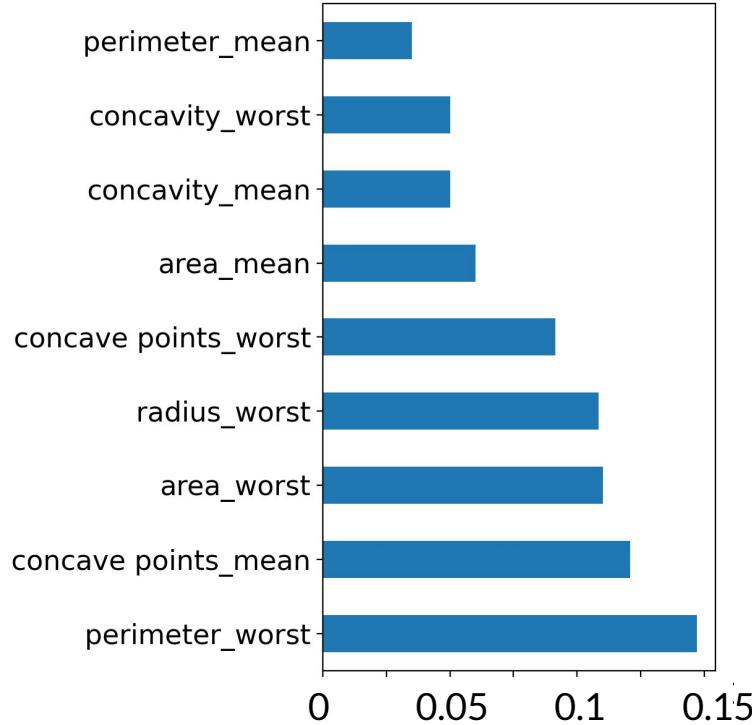
```
def feature_importances_from_tree_based_model_():

    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2,
                                                       stratify=Y, random_state = 123)
    model = RandomForestClassifier()
    model = model.fit(X_train,Y_train)

    feat_importances = pd.Series(model.feature_importances_, index=X.columns)
    feat_importances.nlargest(10).plot(kind='barh')
    plt.show()

    return model
```

# Feature importance plot



# Select features based on importance

```
def select_features_from_model(model):  
  
    model = SelectFromModel(model, prefit=True, threshold=0.012)  
  
    feature_idx = model.get_support()  
    feature_names = df.drop("diagnosis_int", 1).columns[feature_idx]  
    return feature_names
```

# Tying together and evaluation

```
# Calculate and plot feature importances
model = feature_importances_from_tree_based_model_()

# Select features based on feature importances
feature_imp_feature_names = select_features_from_model(model)
```

# Performance table

Method	Feature Count	Accuracy	ROC	Precision	Recall	F1 Score
All Features	30	0.96726	0.964912	0.931818	0.9761900	0.953488
Correlation	21	0.97420	0.973684	0.953488	0.9761904	0.964705
Univariate Feature Selection	20	0.96031	0.95614	0.91111	0.97619	0.94252
Recursive Feature Elimination	20	0.9742	0.973684	0.953488	0.97619	0.964706
Feature Importance	14	0.96726	0.96491	0.931818	0.97619	0.953488

**Best Result**

# Review

- Intro to Preprocessing
- Feature Engineering
- Preprocessing Data at Scale
  - TensorFlow Transform
- Feature Spaces
- Feature Selection
  - Filter Methods
  - Wrapper Methods
  - Embedded Methods

# Data Journey and Data Storage



DeepLearning.AI

---

# Welcome

# Data Journey and Data Storage



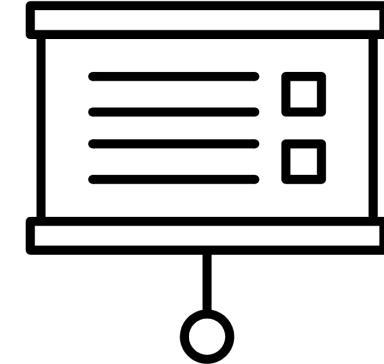
DeepLearning.AI

---

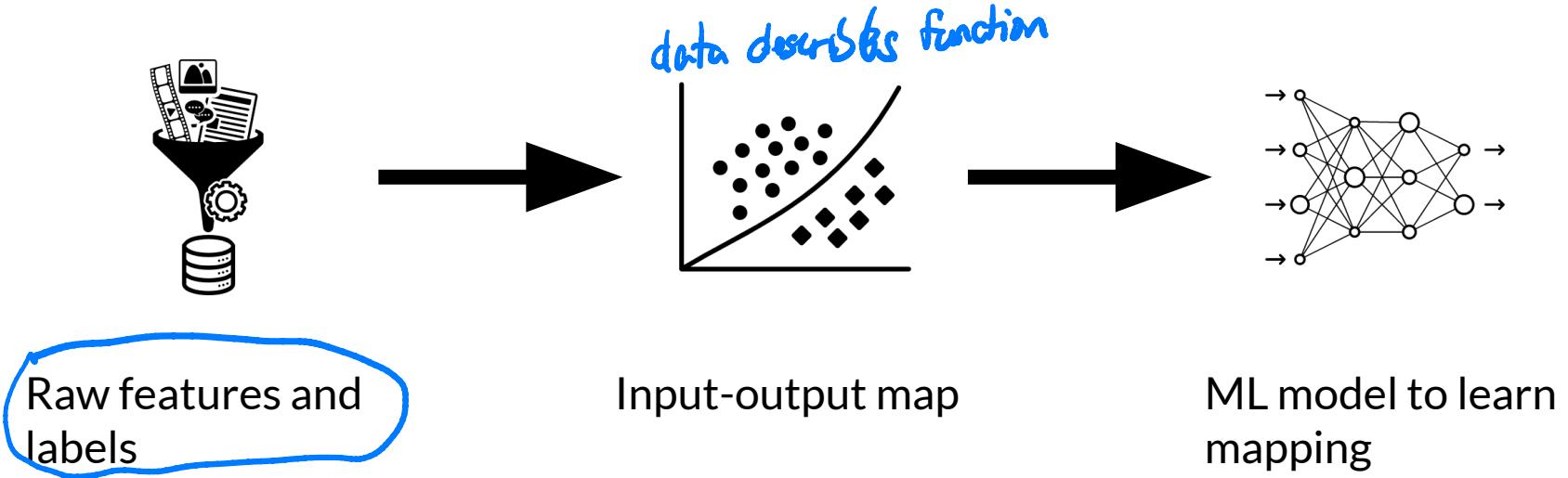
# Data Journey

# Outline

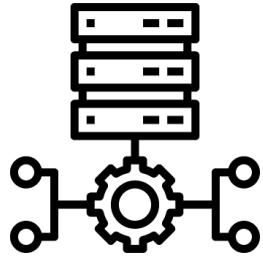
- Metadata
  - Schema = describe data
  - Storage
- The data journey
- Accounting for data and model evolution
- Intro to ML metadata
  - Data provenance → Pipeline →
  - debugging and reproducibility
- Using ML metadata to track changes



# The data journey



# Data transformation



- Data transforms as it flows through the process
  - Interpreting model results requires understanding data transformation
- data formats  
• apply feature engineering  
• train the model*
- ↑

# Artifacts and the ML pipeline

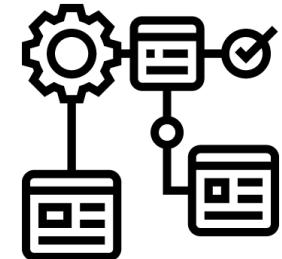
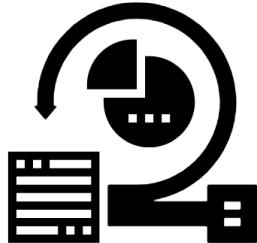


- Artifacts are created as the components of the ML pipeline execute
- Artifacts include all of the data and objects which are produced by the pipeline components *(data, feature engineering, schema, metrics, basically, everything is produced)*
- This includes the data, in different stages of transformation, the schema, the model itself, metrics, etc.

# Data provenance and lineage

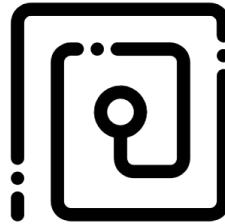
*Synonyms interchangeably  
is a sequence of artifacts*

- The chain of transformations that led to the creation of a particular artifact.
- Important for debugging and reproducibility.



# Data provenance: Why it matters

Helps with debugging and understanding the ML pipeline:



Inspect artifacts at each point in the training process

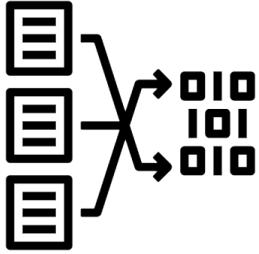
Trace back through a training run

Compare training runs

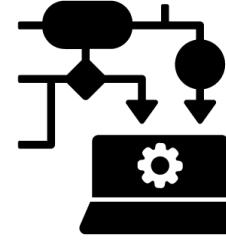
# Data lineage: data protection regulation

- Organizations must closely track and organize personal data
- Data lineage is extremely important for regulatory compliance  
法规遵从

# Data provenance: Interpreting results



Data transformations sequence  
leading to predictions



Understanding the model as it  
evolves through runs

*trained , optimized*

# Data versioning

- Data pipeline management is a major challenge
- Machine learning requires reproducibility *fairly consistently*
- Code versioning: GitHub and similar code repositories
- Environment versioning: Docker, Terraform, and similar
- Data versioning:
  - Version control of datasets
  - Examples: DVC, Git-LFS *large file storage*

# Data Journey and Data Storage

---



DeepLearning.AI

## Intro to ML Metadata

# Metadata: Tracking artifacts and pipeline changes

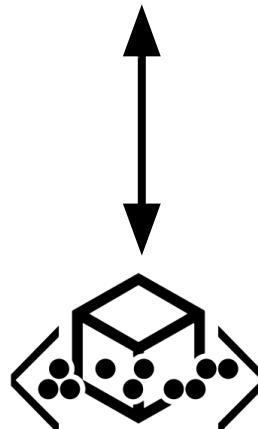
legal liability 法律責任



ML pipeline

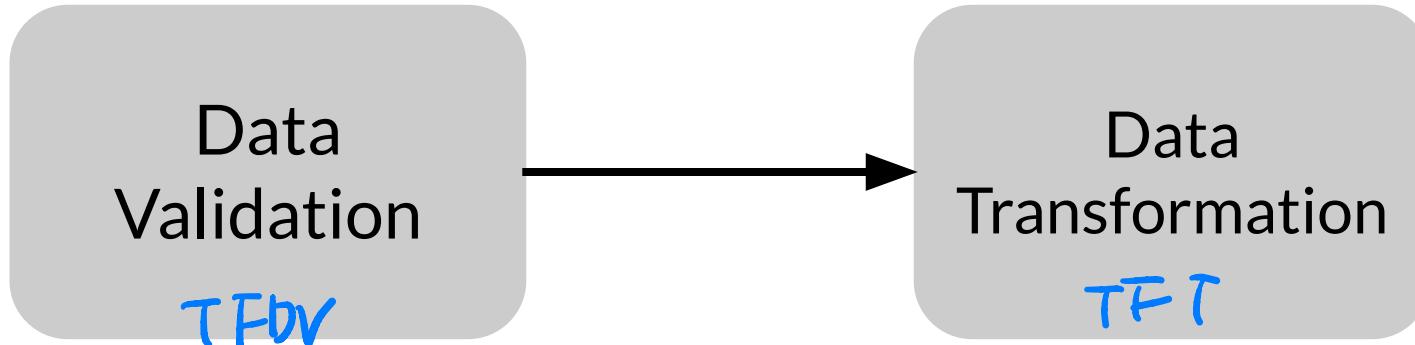
Metadata

helps understand  
and analyze interconnected parts

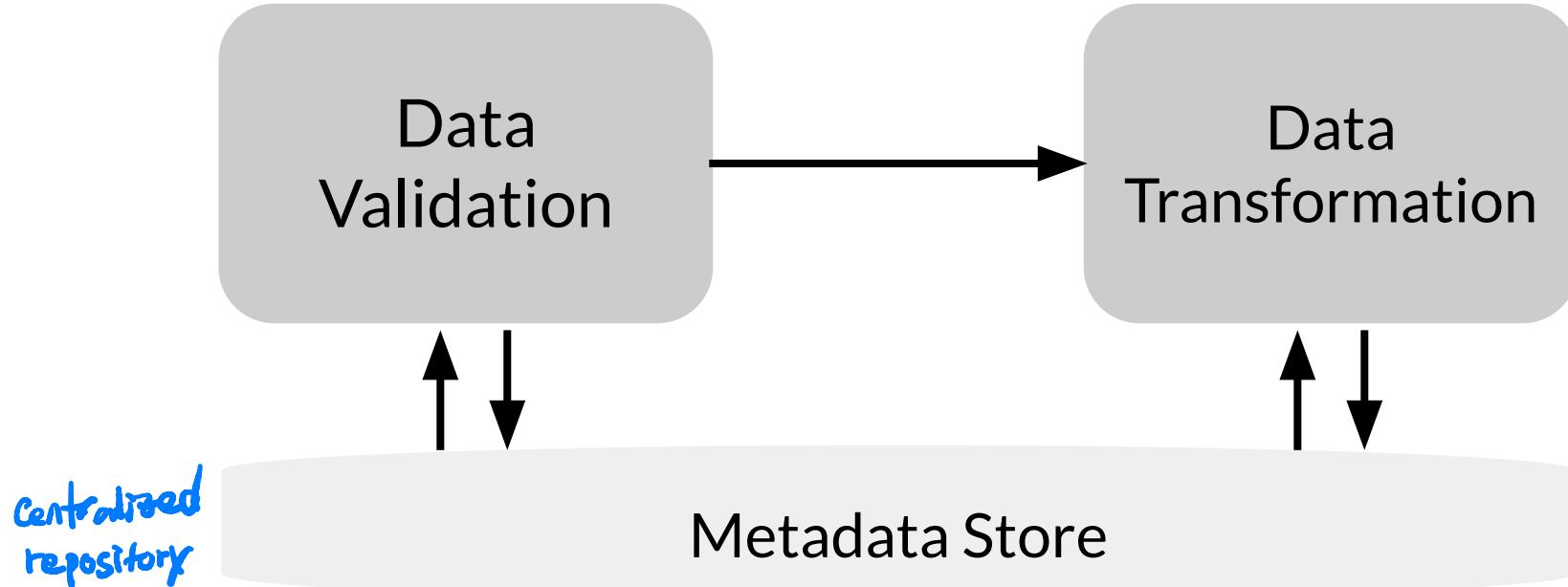


Metadata includes =  
— pipeline components  
— execution  
— training runs  
— results artifacts

# Ordinary ML data pipeline

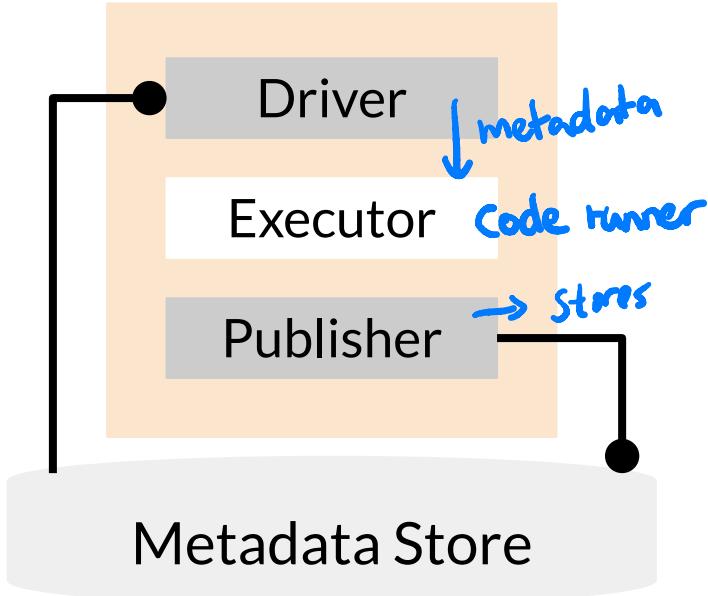


# Metadata: Tracking progress



centralised  
repository

# Metadata: TFX component architecture



- **Driver:**
  - Supplies required metadata to executor
- **Executor:**
  - Place to code the functionality of component
- **Publisher:**
  - Stores result into metadata

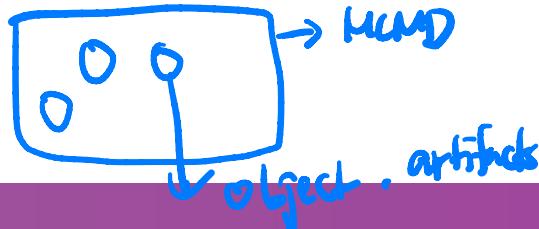
# ML Metadata library (MLMD)

- Tracks metadata flowing between components in pipeline
- Supports multiple storage backends

MLMD — used independently  
— as integral part of an ML pipeline

disc  
file  
block

artifacts: objects which are stored in MLMD are referred to as artifacts

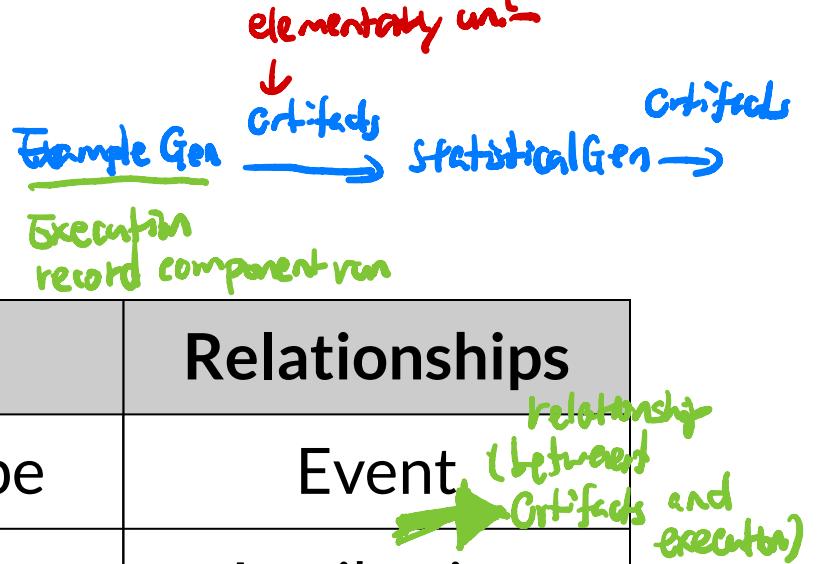


MLMD = need to know data flows  
between components

# ML Metadata terminology

Units	Types	Relationships
Artifact	ArtifactType	Event
Execution	ExecutionType	Attribution
Context	ContextType	Association

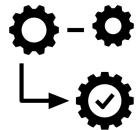
ExampleGen  
Artifact 1  
Execution 1  
together or  
separately  
Group → context



# Metadata stored



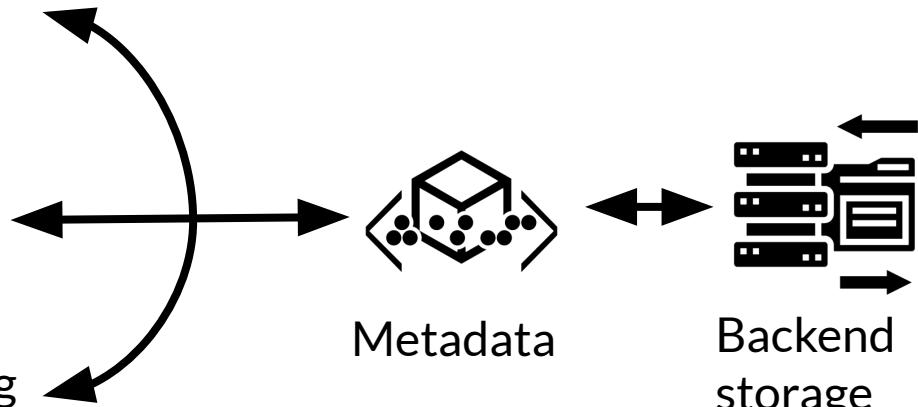
Artifacts: Data going as input or generated as output by a component



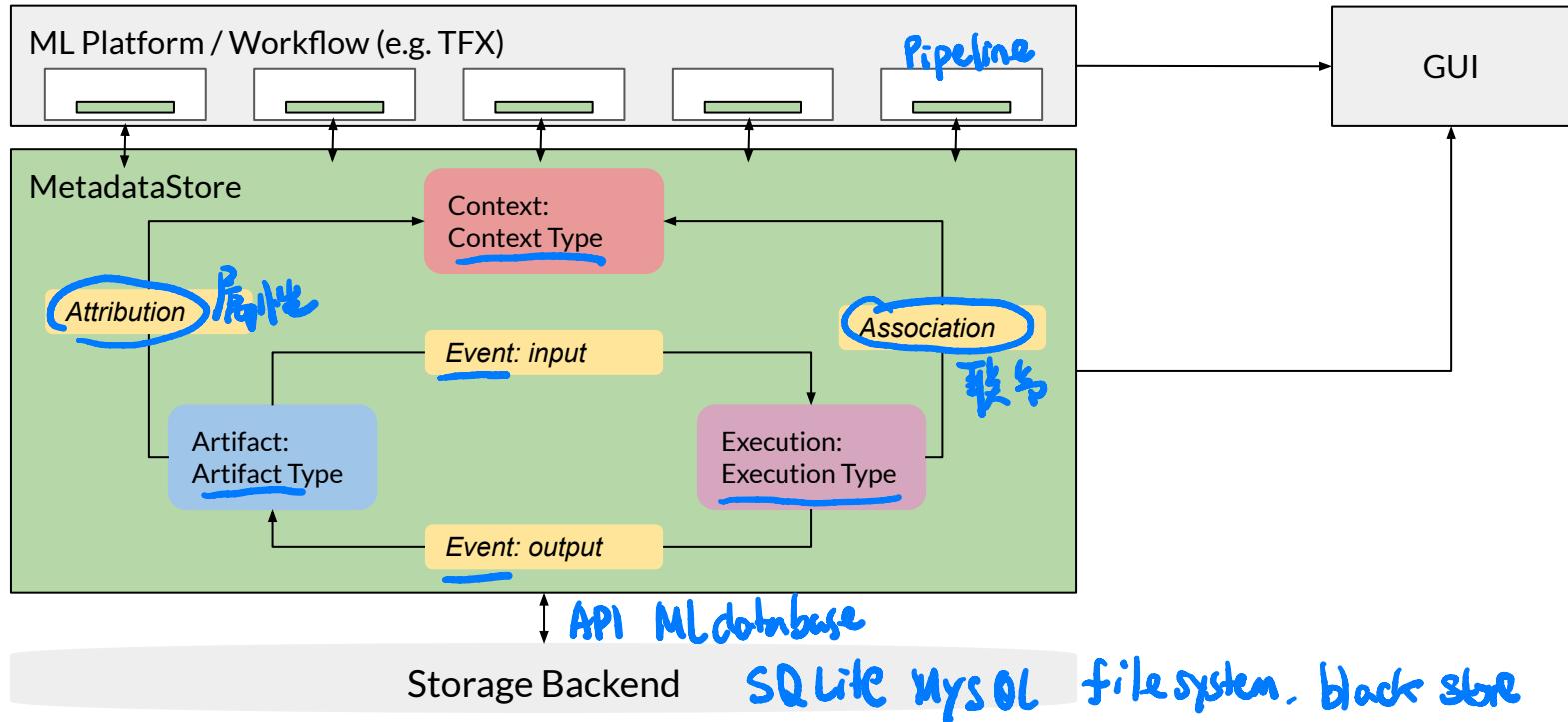
Execution: Record of component in pipeline.



Context: Conceptual grouping of executions and artifacts.



# Inside MetadataStore



# Key points

ML metadata:

- Architecture and nomenclature
- Tracking metadata flowing between components in pipeline

# Data Journey and Data Storage

---

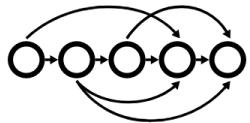


DeepLearning.AI

# ML Metadata in action

# Other benefits of ML Metadata

debug



Produce DAG of pipelines

Directed acyclic graph



Verify the inputs used in an execution



List all artifacts  
models



Compare artifacts

# Import ML Metadata

```
!pip install ml-metadata
```

```
from ml_metadata import metadata_store  
from ml_metadata.proto import metadata_store_pb2
```

protocolbuffer  
= Datiformat zur Serialisierung  
mit einer Schnittstellen-  
Beschreibungssprache

# ML Metadata storage backend

- ML metadata registers metadata in a database called Metadata Store
- APIs to record and retrieve metadata to and from the storage backend:
  - Fake database: in-memory for fast experimentation/prototyping
  - SQLite: in-memory and disk
  - MySQL: server based
  - Block storage: File system, storage area network, or cloud based

# Fake database

↓ Connection config object 運行对象面盤。

```
connection_config = metadata_store_pb2.ConnectionConfig()  
  
# Set an empty fake database proto  
  
connection_config.fake_database.SetInParent() primary memory of system  
  
store = metadata_store.MetadataStore(connection_config)  
Store object
```

# SQLite

```
connection_config = metadata_store_pb2.ConnectionConfig()  
  
connection_config.sqlite.filename_uri = '...'  
connection_config.sqlite.connection_mode = 3 # READWRITE_OPENCREATE
```

store = metadata\_store.MetadataStore(connection\_config)

A key part of  
how you interact with ML Metadata

# MySQL

```
connection_config = metadata_store_pb2.ConnectionConfig()

connection_config.mysql.host = '...'
connection_config.mysql.port = '...'
connection_config.mysql.database = '...'
connection_config.mysql.user = '...'
connection_config.mysql.password = '...'

store = metadata_store.MetadataStore(connection_config)
```

# ML metadata practice: ungraded lab

- Using a tabular data set, you will explore:
  - Explicit programming in ML Metadata
  - Integration with TFDV
  - Store progress and create provisions to backtrack the experiment

# Key points

- Walk through over the data journey addressing lineage and provenance
- The importance of metadata for tracking data evolution
- ML Metadata library and its usefulness to track data changes
- Running an example to register artifacts, executions, and contexts



DeepLearning.AI

## Evolving Data

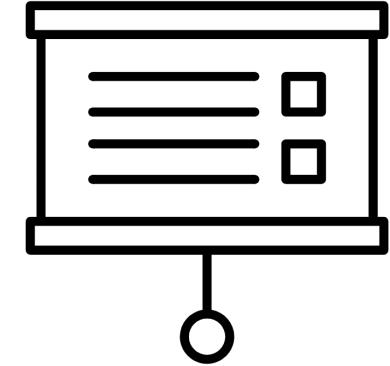
---

# Schema Development

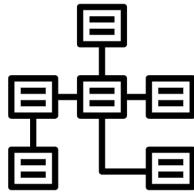
# Outline

企业模式环境

- Develop enterprise schema environments
- Iteratively generate and maintain enterprise data schemas



# Review: Recall Schema



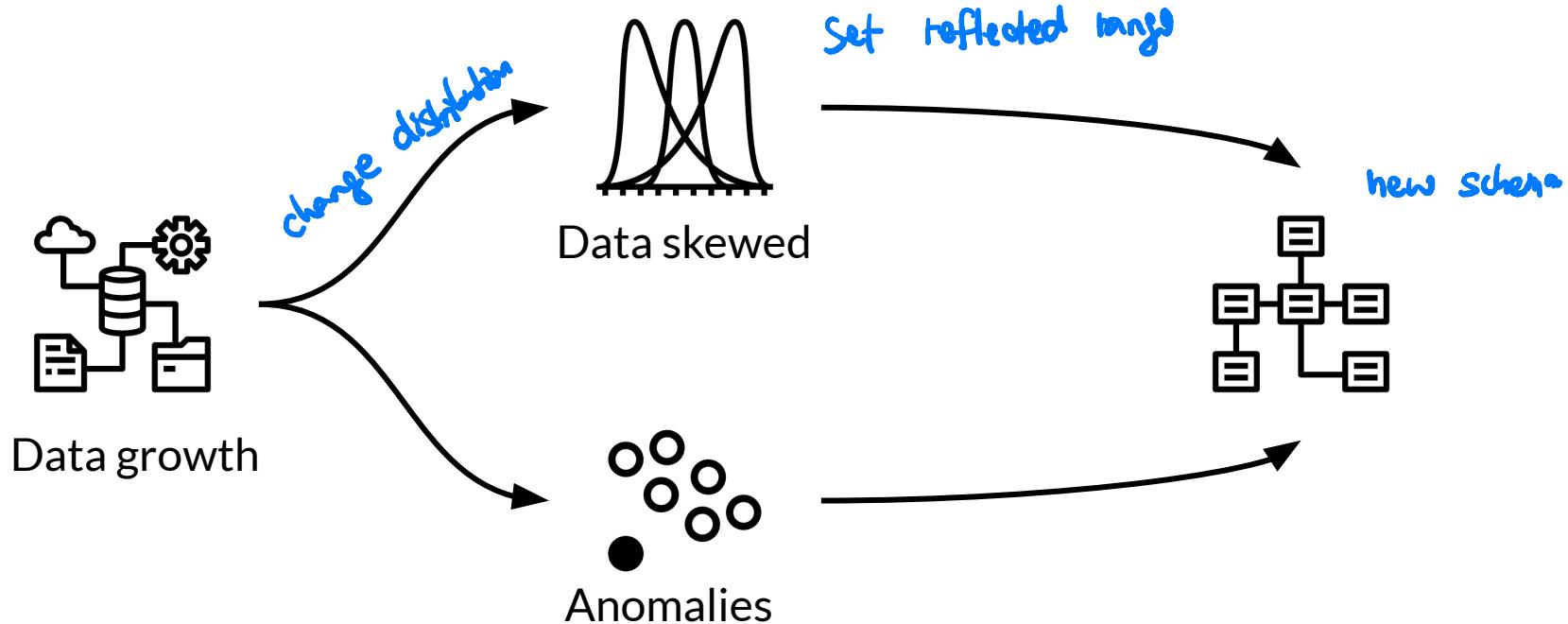
Schema

*relational objects  
summarizing features  
in a given dataset or  
projects*

- Feature name
- Type: float, int, string, etc
- Required or optional
- Valency (features with multiple values)  
*ACBAA*
- Domain: range, categories
- Default values

- lists
- array features
- min Max values

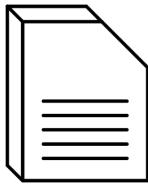
# Iterative schema development & evolution



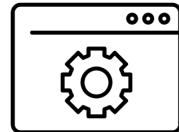
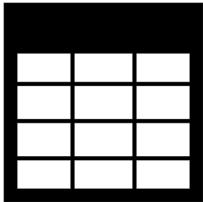
# Reliability during data evolution

扰乱中断

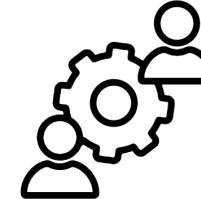
Platform needs to be resilient to disruptions from:



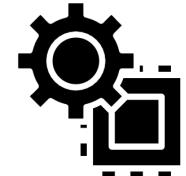
Inconsistent data



Software



User configurations



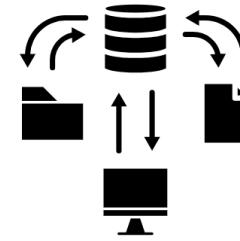
Execution environments

# Scalability during data evolution

Platform must scale during:



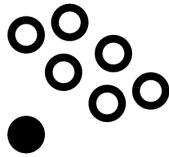
High data volume during training



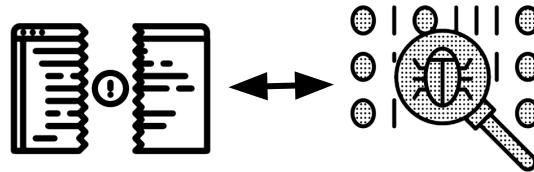
Variable request traffic  
during serving

# Anomaly detection during data evolution

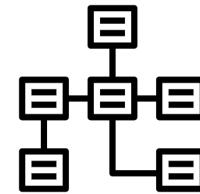
Platform designed with these principles:



Easy to detect anomalies

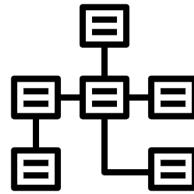


Data errors treated  
same as code bugs

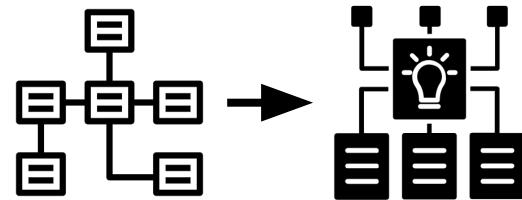


Update data schema

# Schema inspection during data evolution



Looking at schema **versions** to track data evolution



Schema can drive other automated processes



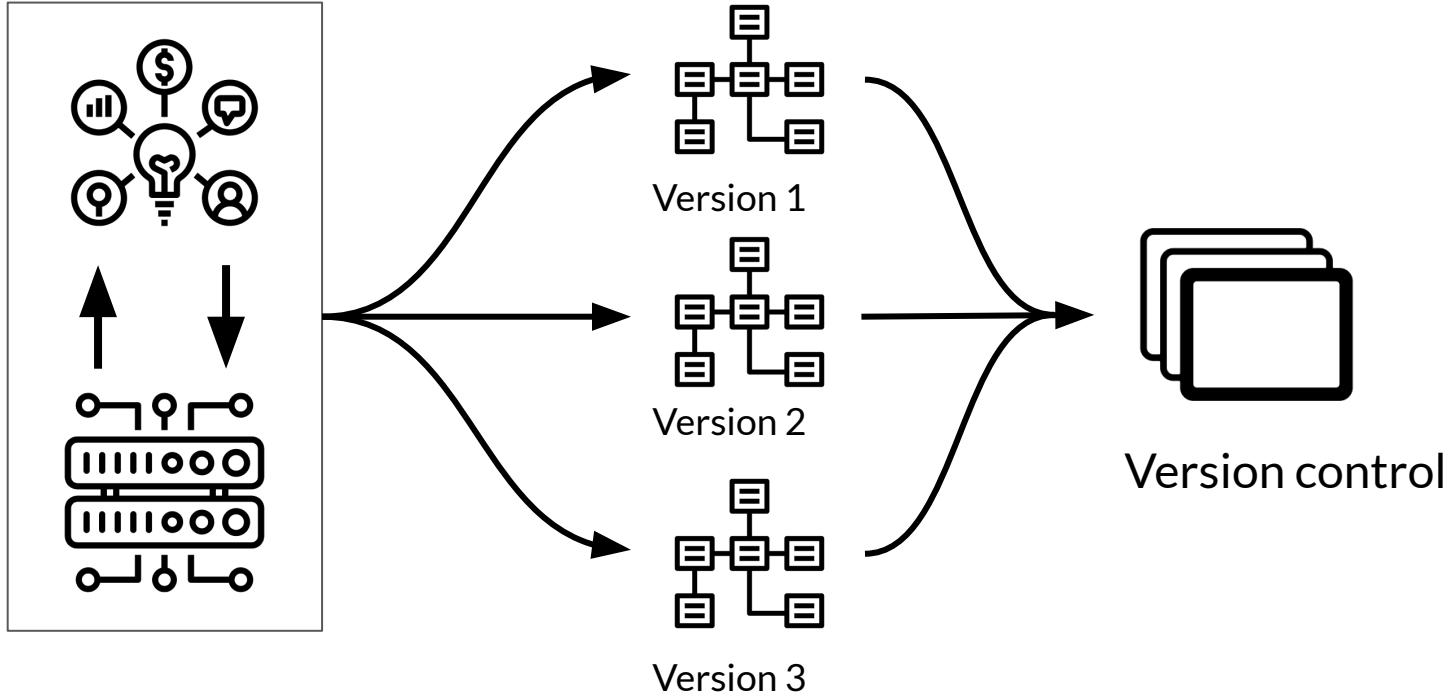
DeepLearning.AI

## Evolving Data

---

# Schema Environments

# Multiple schema versions

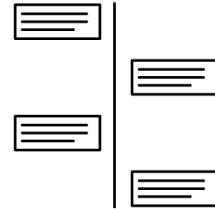


# Maintaining varieties of schema

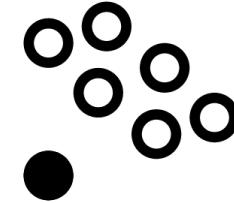


Business use-case needs  
to support data from  
different sources.

*different schemas to reflect difference in data*



Data evolves rapidly



Is anomaly part of  
accepted type of  
data?

# Inspect anomalies in serving dataset

```
stats_options = tfdv.StatsOptions(schema=schema,  
                                  infer_type_from_schema=True)  
  
eval_stats = tfdv.generate_statistics_from_csv(  
    data_location=SERVING_DATASET,  
    stats_options=stats_options  
)  
  
serving_anomalies = tfdv.validate_statistics(eval_stats, schema)  
tfdv.display_anomalies(serving_anomalies)
```

① Create Statistics using Schema

② Compare

# Anomaly: No labels in serving dataset

	Anomaly short description	Anomaly long description
Feature name		
'Cover_Type'	Out-of-range values	Unexpectedly small value: 0.

# Schema environments

- Customize the schema for each environment
- Ex: Add or remove label in schema based on type of dataset

Training      Serving

# Create environments for each schema

```
schema.default_environment.append('TRAINING')
schema.default_environment.append('SERVING')

tfdv.get_feature(schema, 'Cover_Type'
    .not_in_environment.append('SERVING')
```

# Inspect anomalies in serving dataset

```
serving_anomalies = tfdv.validate_statistics(eval_stats,  
                                              schema,  
                                              environment='SERVING')  
  
tfdv.display_anomalies(serving_anomalies)  
# No anomalies found
```

# Key points

- Iteratively update and fine-tune schema to adapt to evolving data
- How to deal with scalability and anomalies
- Set schema environments to detect anomalies in serving requests

# Enterprise Data Storage



DeepLearning.AI

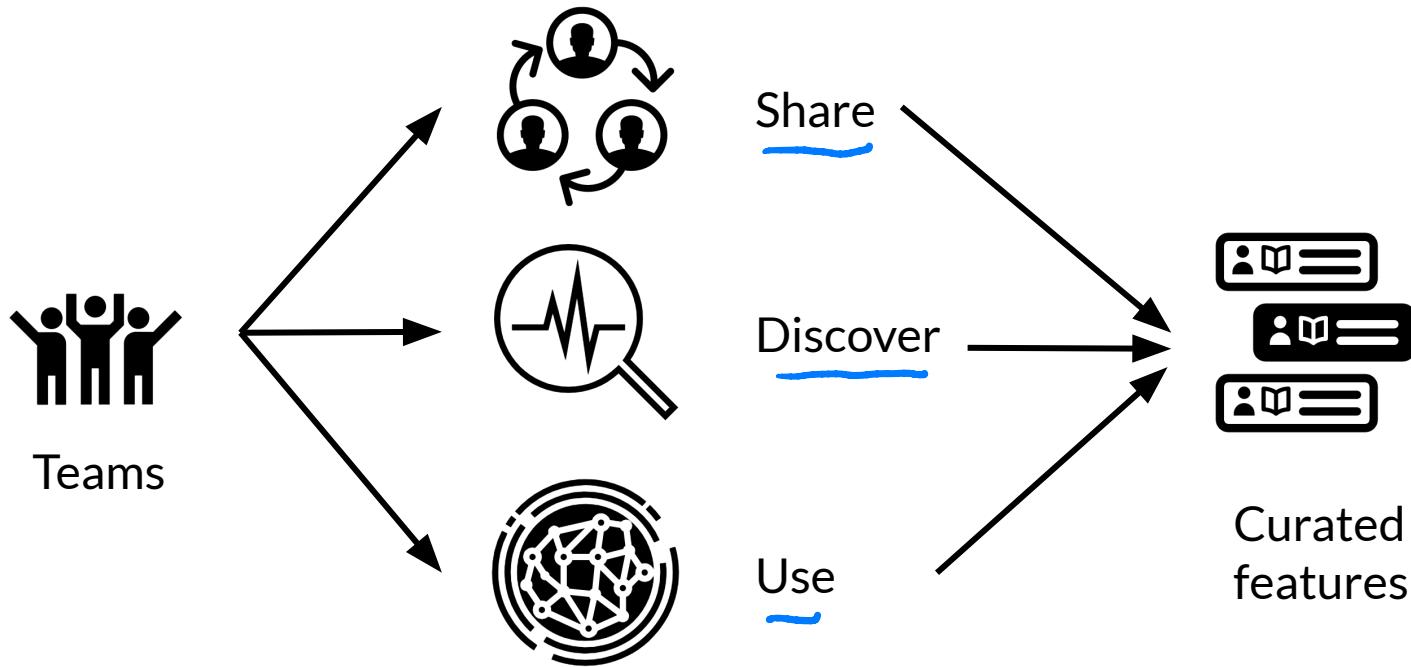
## Feature Stores

# Feature stores

repo

documented  
curated  
access controlled features

Online  
offline



# Feature stores

Many modeling problems use identical or similar features



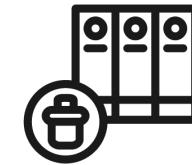
# Feature stores



Avoid duplication

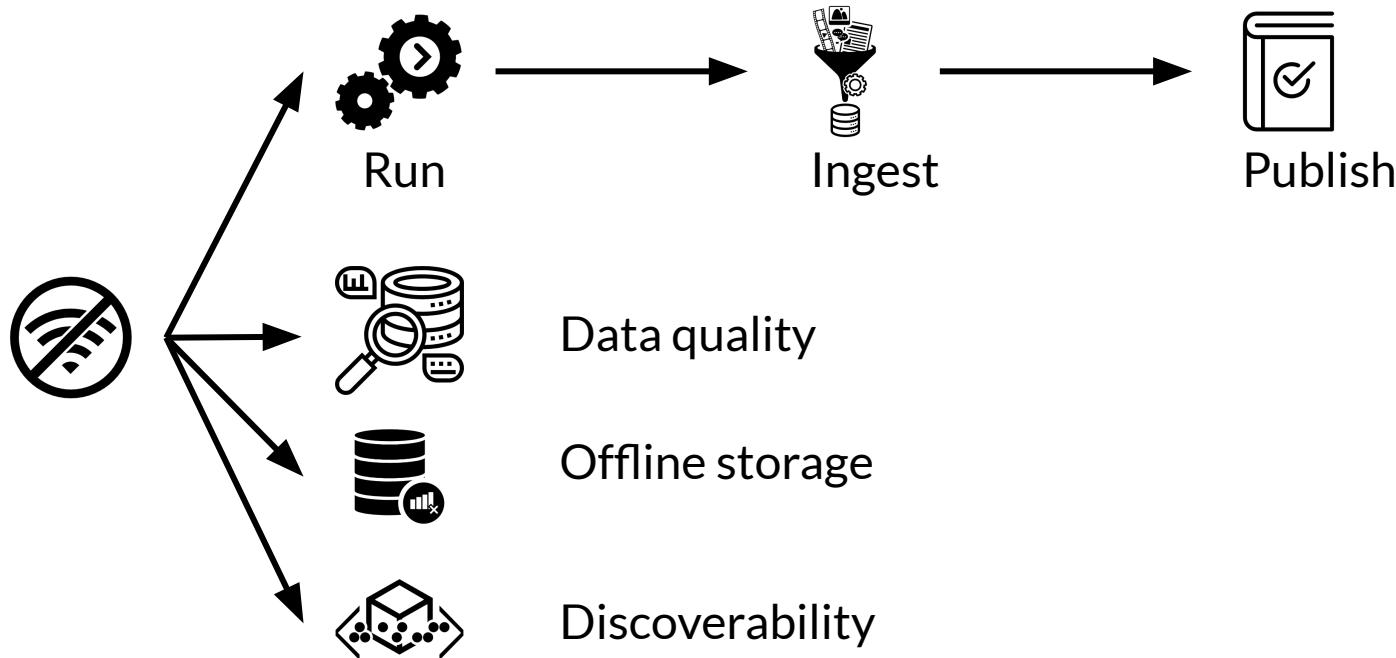


Control access

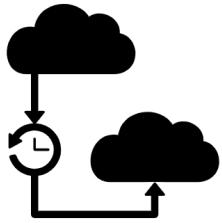


Purge *清理*

# Offline feature processing



# Online feature usage



Low latency access  
to features



Features difficult  
to compute online



Precompute and  
store for low  
latency access

# Features for online serving - Batch



Batch  
precomputing



Loading  
history

- Simple and efficient
- Works well for features to only be updated every few hours or once a day
- Same data is used for training and serving

# Feature store: key aspects

- Managing feature data from a single person to large enterprises.
- Scalable and performant access to feature data in training and serving.
- Provide consistent and point-in-time correct access to feature data.
- Enable discovery, documentation, and insights into your features.

# Enterprise Data Storage



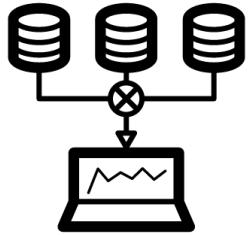
DeepLearning.AI

---

# Data Warehouse

# Data warehouse

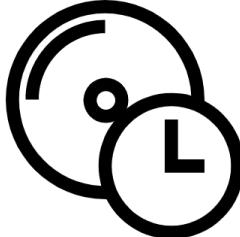
Store data in a data warehouse  
follow a consistent schema



Aggregates  
data sources



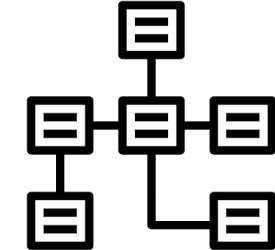
Processed  
and analyzed



Read  
optimized



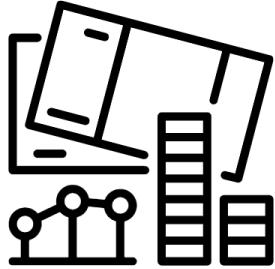
Not  
real time



Follows  
schema

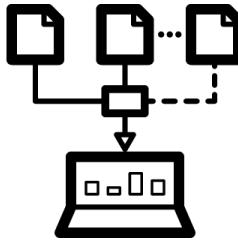


# Key features of data warehouse

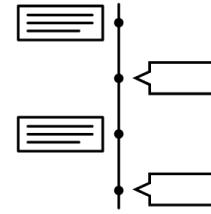


Subject oriented

*Stored in it  
revolves around  
a topic*

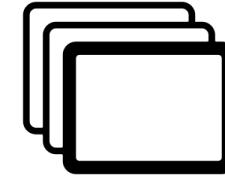


Integrated



Non volatile

*previous version  
will not be erased*



Time variant



# Advantages of data warehouse



Enhanced  
ability to  
analyze data



Timely access  
to data



Enhanced  
data quality  
and  
consistency



High return on  
investment



Increased query  
and system  
performance



# Comparison with databases

Data warehouse	Database
Online analytical processing (OLAP)	Online transactional processing (OLTP)
Data is refreshed from source systems	Data is available real-time
Stores historical and current data	Stores only current data
Data size can scale to $\geq$ terabytes	Data size can scale to gigabytes
Queries are complex, used for analysis	Queries are simple, used for transactions
Queries are long running jobs	Queries executed almost in real-time
Tables need not be normalized	Tables normalized for efficiency

# Enterprise Data Storage

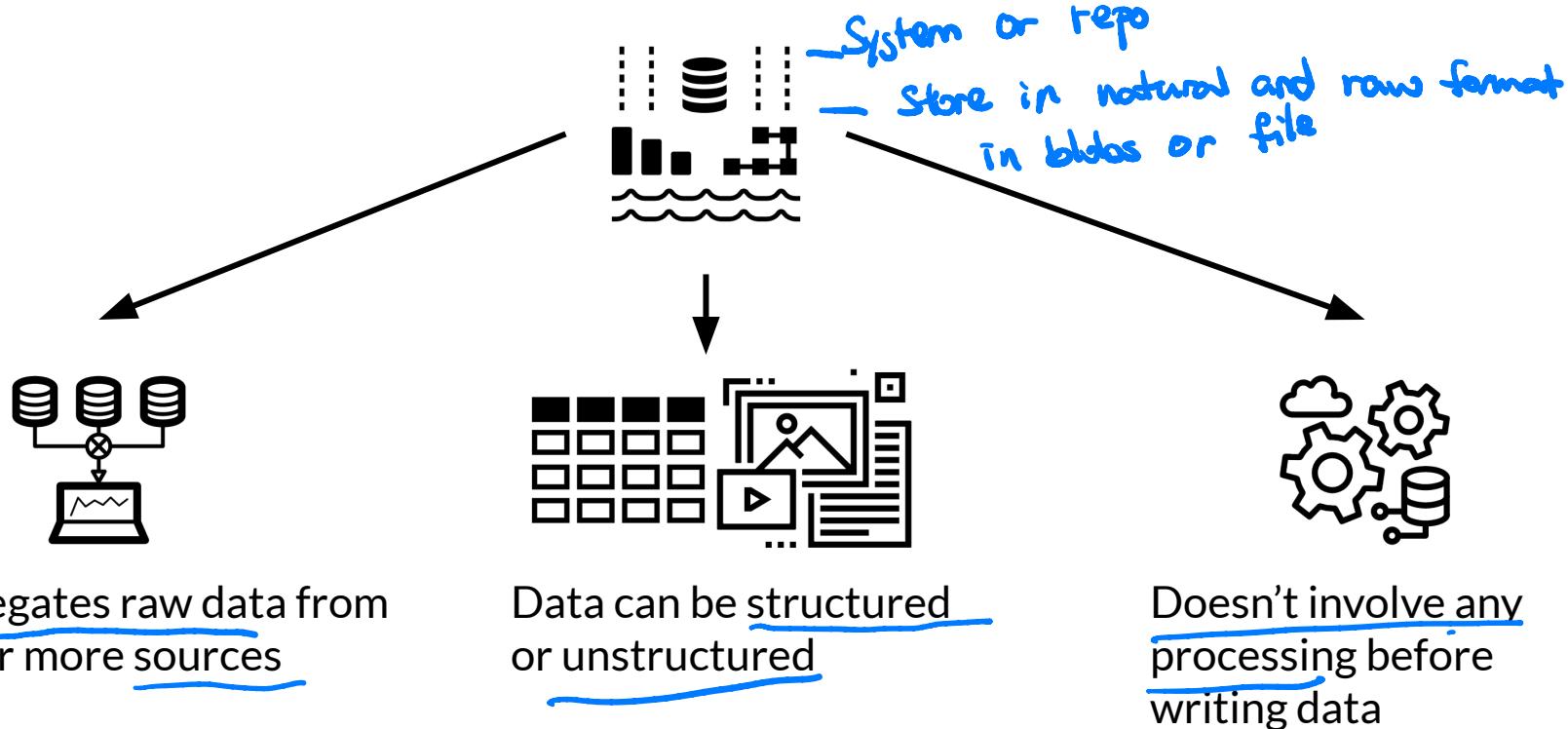


DeepLearning.AI

---

# Data Lakes

# Data lakes



# Comparison with data warehouse

	Data warehouses	Data lakes
<b>Data Structure</b>	Processed	Raw
<b>Purpose of data</b>	Currently in use	Not yet determined
<b>Users</b>	Business professionals	Data scientists
<b>Accessibility</b>	More complicated and costly to make changes	Highly accessible and quick to update

# Key points

- **Feature store:** central repository for storing documented, curated, and access-controlled features, specifically for ML.
- **Data warehouse:** subject-oriented repository of structured data optimized for fast read.
- **Data lakes:** repository of data stored in its natural and raw format.

# Advanced Labeling, Augmentation and Data Preprocessing



DeepLearning.AI

---

# Welcome



DeepLearning.AI

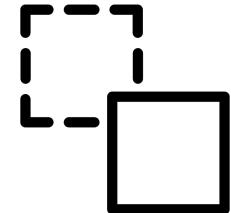
## Advanced Labeling

## Semi-supervised Labeling

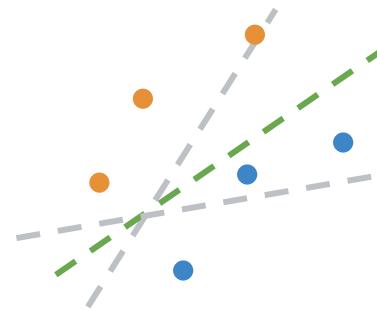
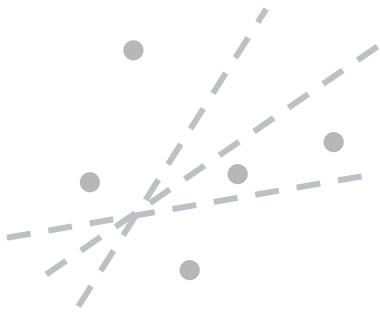
# Outline

- Overview of advanced labeling techniques:

- Semi-supervised learning    improve model performance by expanding labeled database
- Active learning    intelligence sampling to assign labels based on existing data to unlabeled data
- Weak supervision with Snorkel    → framework  
    programmatically labeled data



# Why is Advanced Labeling Important?



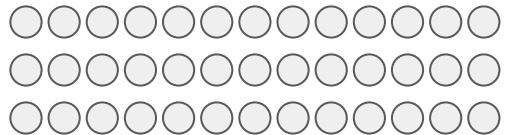
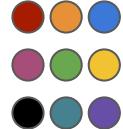
ML use is growing everywhere

...as is the need for labeled training sets

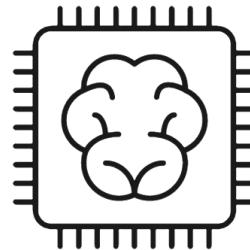
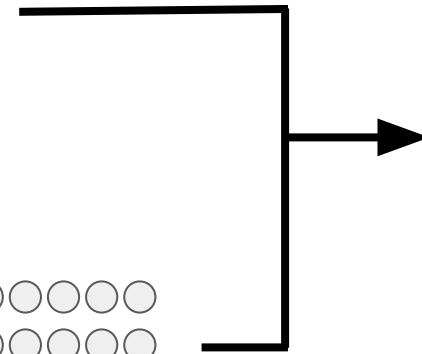
- Manually labeling of data is expensive *and difficult*
- Unlabeled data is usually cheap and easy to get
- Unlabeled data contains a lot of information that can improve our model

# Human labeling, Semi-supervised

Small pool human  
labeled data



Large pool unlabeled data

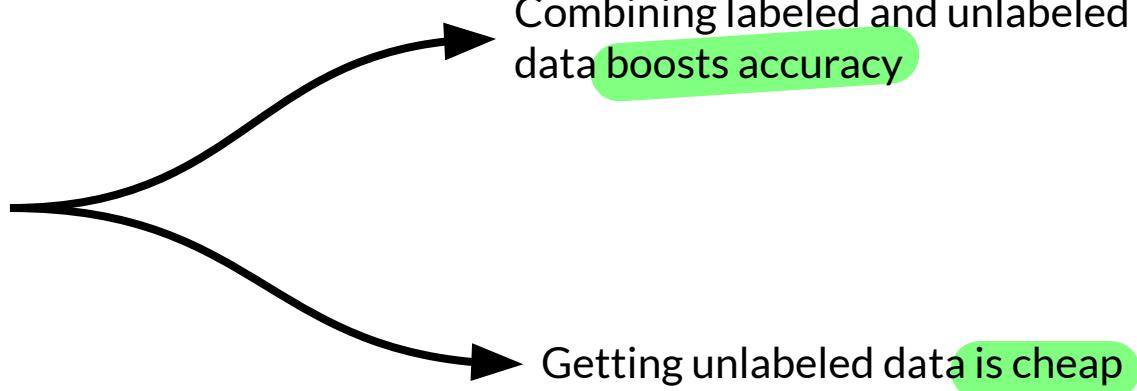


Train your model

Relies on some degree of  
uniformity or clustering  
within feature space

# Human labeling, Semi-supervised

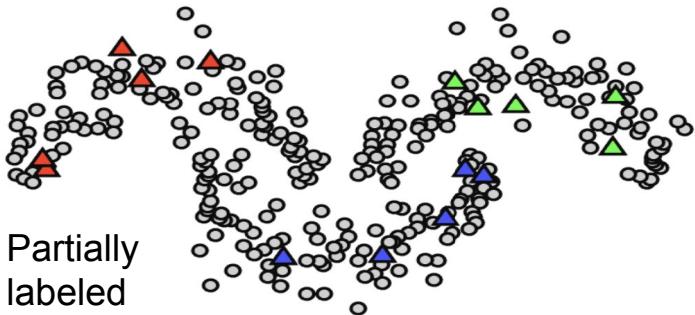
Advantages



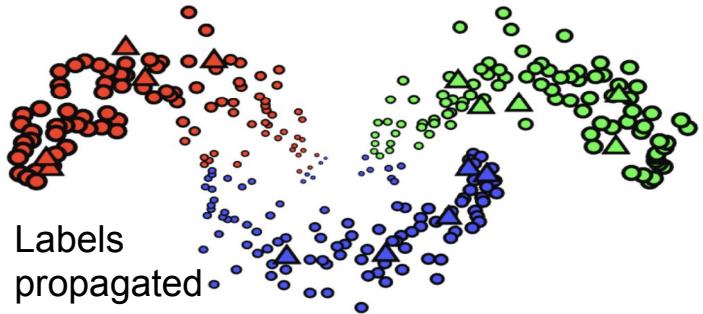
Label propagation is an algorithm that assigns label to previously unlabeled examples

- Semi-supervised ML algorithm
- A subset of the examples have labels
- Labels are propagated to the unlabeled points:
  - Based on similarity or “community structure”

# Label propagation - Graph based



Unlabeled examples can be assigned labels based on their neighbors





DeepLearning.AI

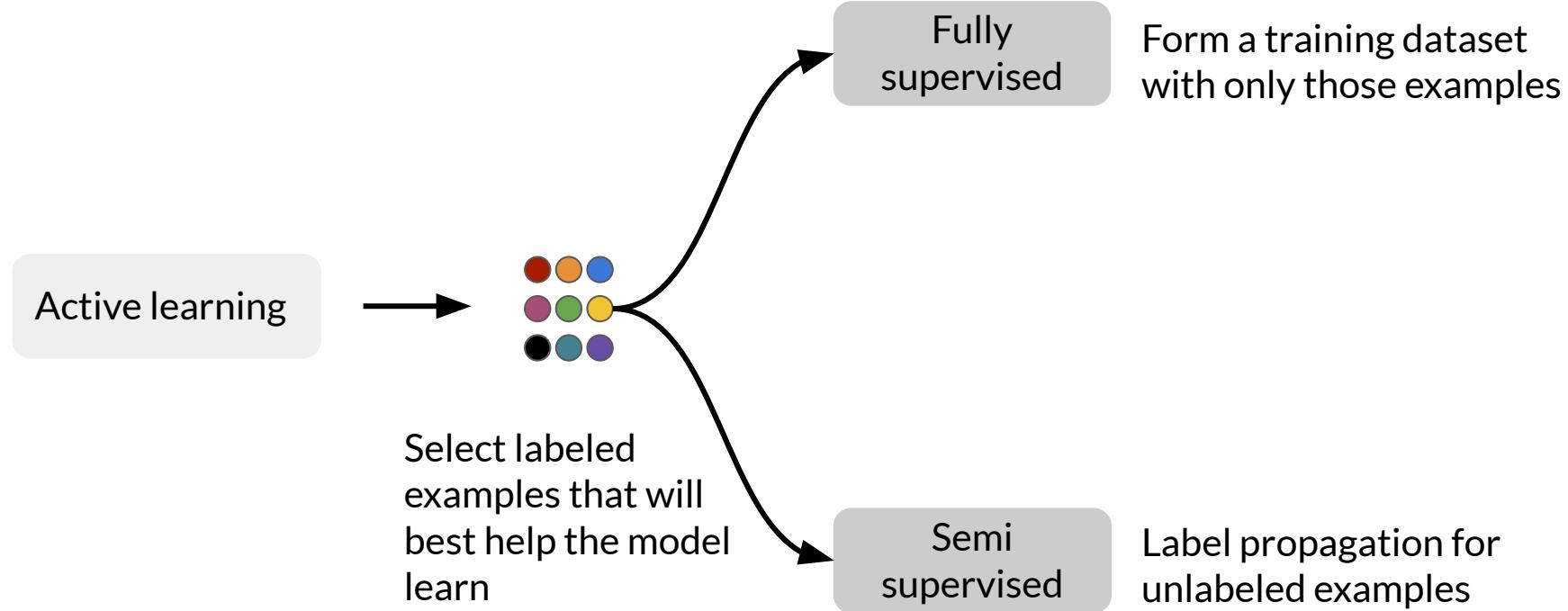
## Advanced Labeling

## Active Learning

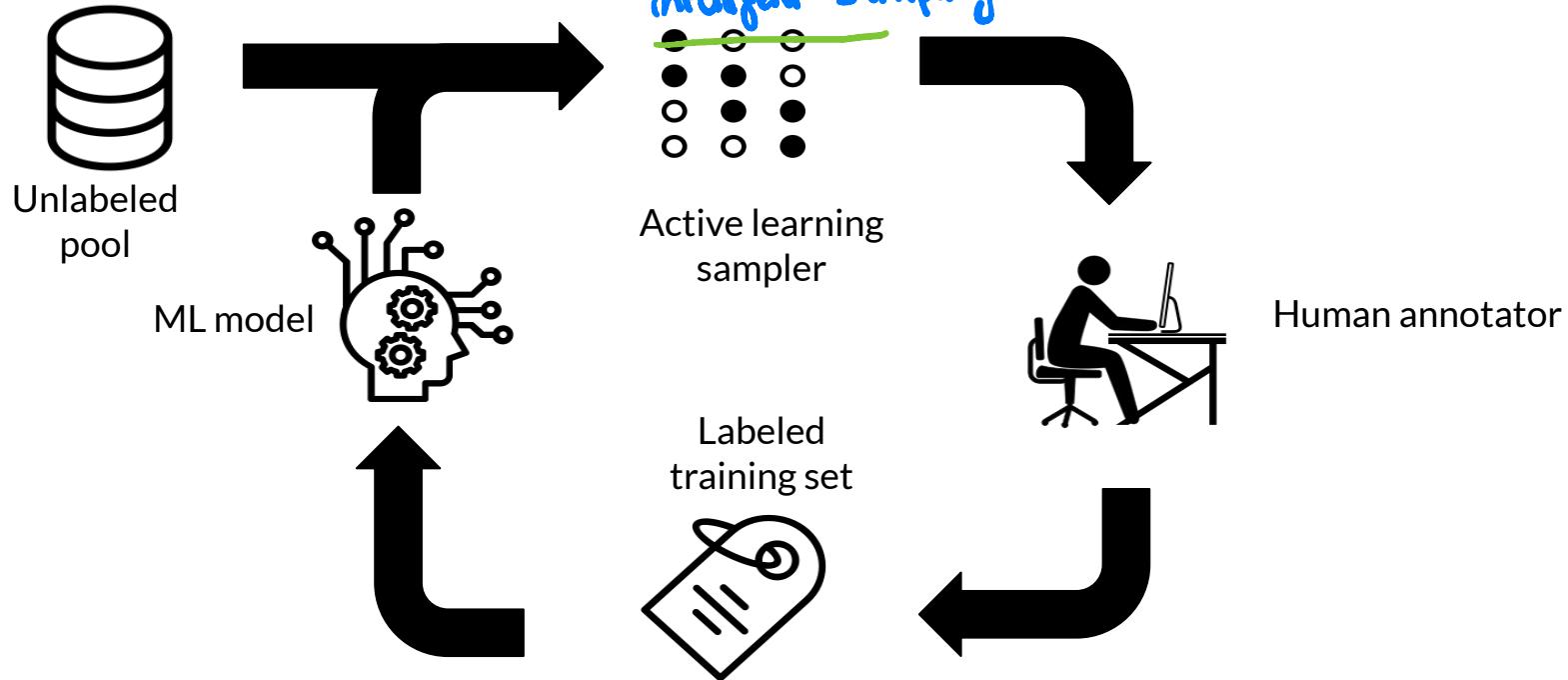
# Active learning

- A family of algorithms for intelligently sampling data
- Select the points to be labeled that would be most informative for model training
- Very helpful in the following situations:
  -  Constrained data budgets: you can only afford labeling a few points
  -  Imbalanced dataset: helps selecting rare classes for training
  -  Target metrics: when baseline sampling strategy does not improve selected metrics

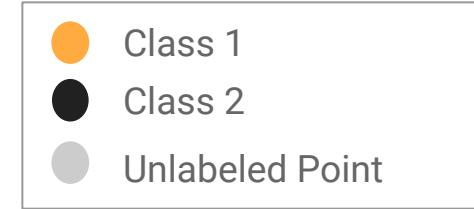
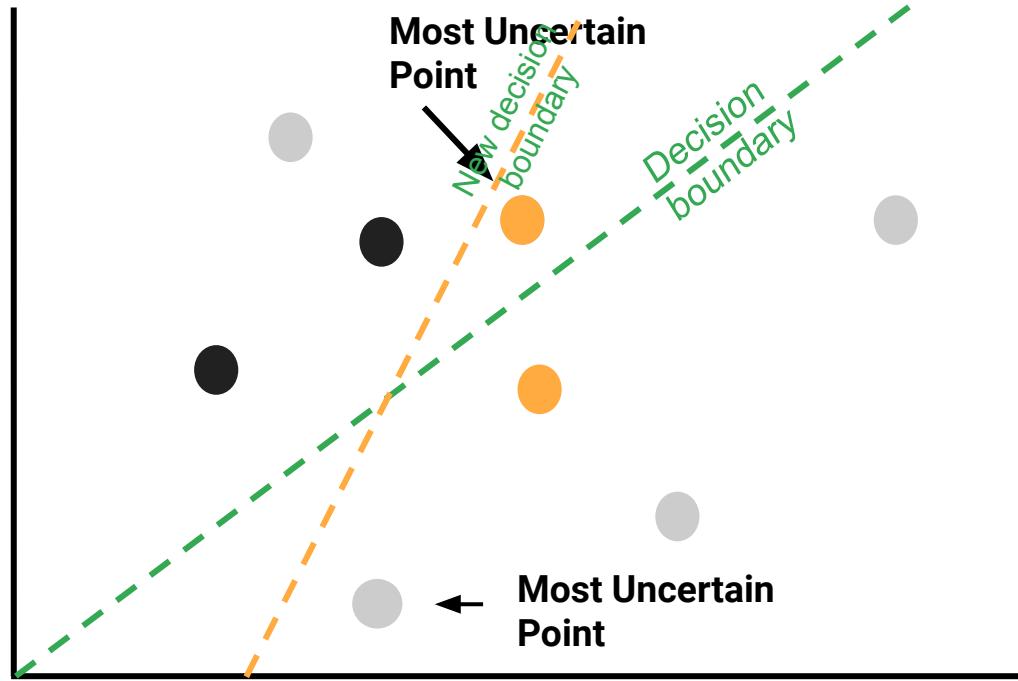
# Active learning strategies



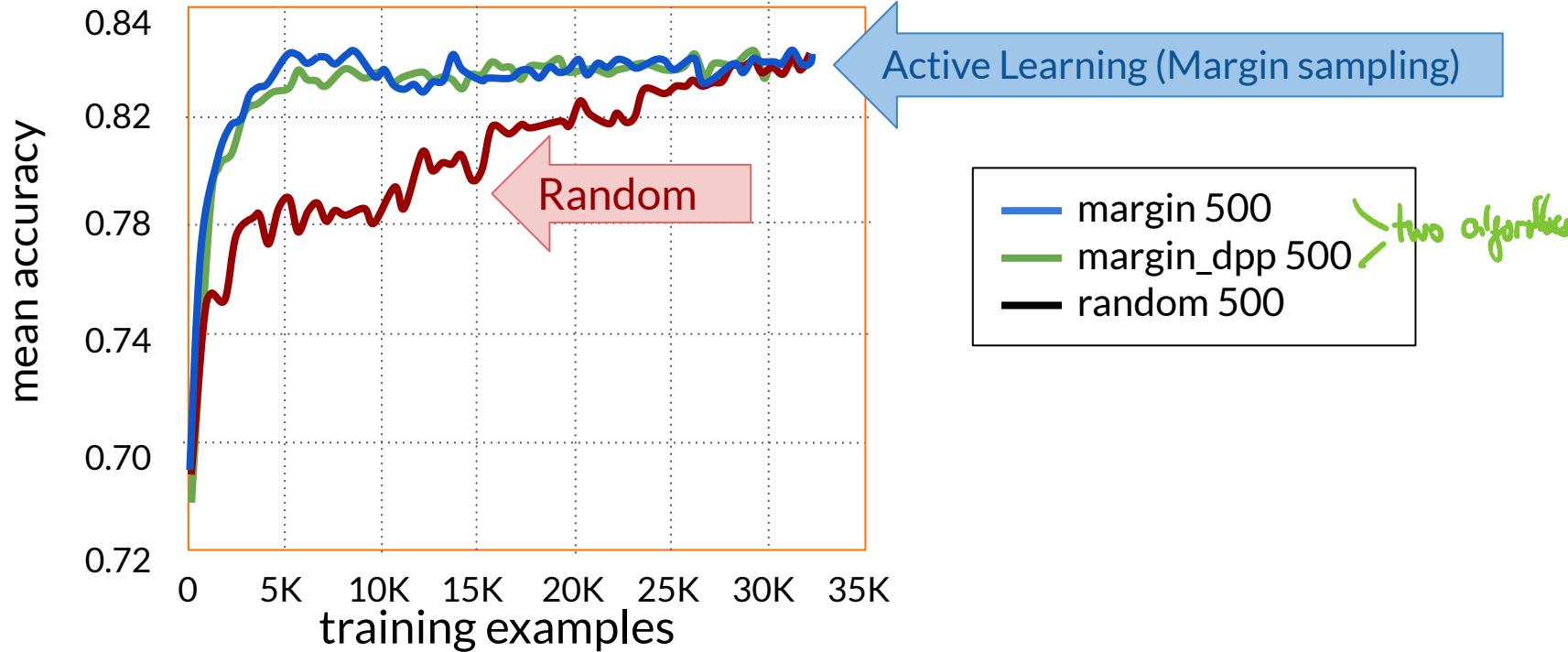
# Active learning cycle



# Margin sampling



# Example results - Different Sampling Techniques



# Active learning sampling techniques

**Margin sampling:** Label points the current model is least confident in.

**Cluster-based sampling:** sample from well-formed clusters to "cover" the entire space.

**Query-by-committee:** train an ensemble of models and sample points that generate disagreement.

**Region-based sampling:** Runs several active learning algorithms in different partitions of the space.

# Advanced Labeling



DeepLearning.AI

# Weak Supervision

# Hand labeling: intensive labor

“Hand-labeling training data for machine learning problems is effective, but very labor and time intensive. This work explores how to use algorithmic labeling systems relying on other sources of knowledge that can provide many more labels but which are noisy.”

Jeff Dean, March 14, 2019

# Weak supervision

main objective is to learn a generative model  
that determine the relevance of each of these  
noisy source

“Weak supervision is about leveraging higher-level and/or noisier input from subject matter experts (SMEs).”

-Weak Supervision: The New Programming Paradigm for Machine Learning  
Blog post by Ratner, Varma, Hancock, Re, and Hazy Lab

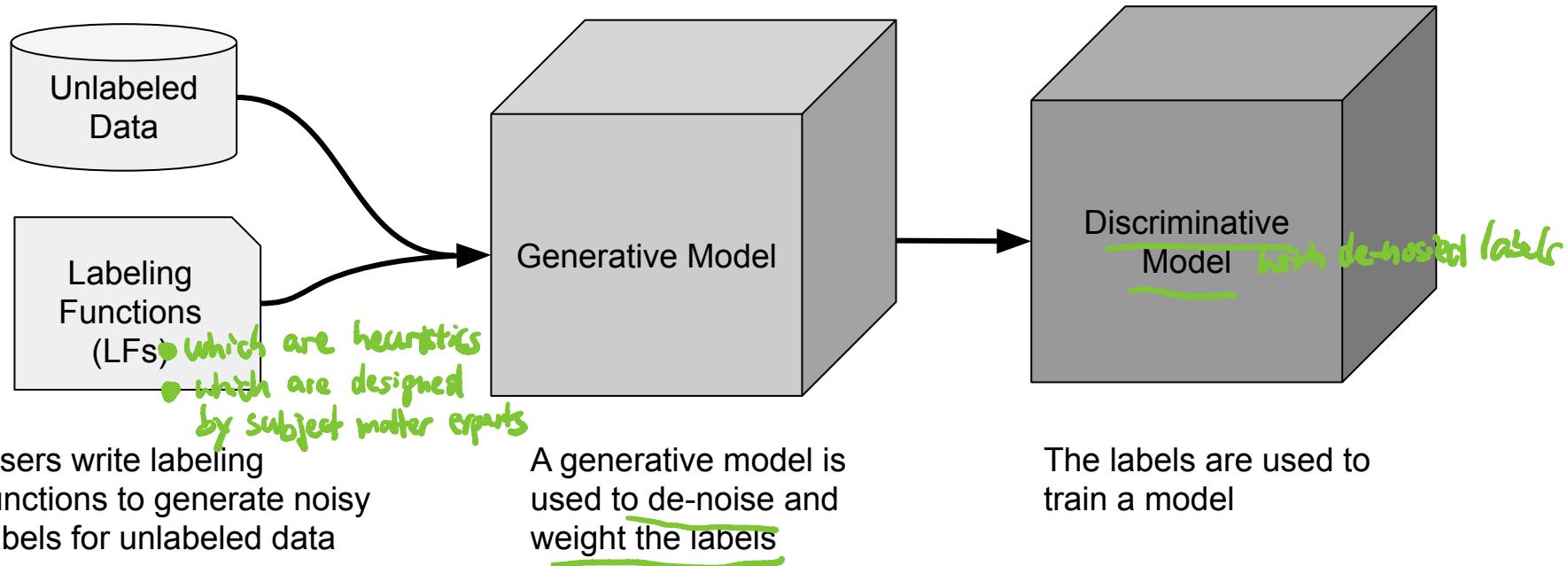
# Weak supervision

- Unlabeled data, without ground-truth labels
- One or more weak supervision sources
  - A list of heuristics that can automate labeling
  - Typically provided by subject matter experts
- Noisy labels have a certain probability of being correct, not 100%
- Objective: learn a generative model to determine weights for weak supervision sources

# Snorkel

- Project started at Stanford in 2016
- Programmatically building and managing training datasets without manual labeling
- Automatically: models, cleans, and integrates the resulting training data
- Applies novel, theoretically-grounded techniques
- Also offers data augmentation and slicing

# Data programming pipeline in Snorkel



# Snorkel labeling functions

```
from snorkel.labeling import labeling_function

@labeling_function()
def lf_keyword_my(x):
    """Many spam comments talk about 'my channel', 'my video', etc."""
    return SPAM if "my" in x.text.lower() else ABSTAIN

@labeling_function()
def lf_short_comment(x):
    """Non-spam comments are often short, such as 'cool video!'."""
    return NOT_SPAM if len(x.text.split()) < 5 else ABSTAIN
```

# Key points

- Semi-supervised learning:
  - Applies the best of supervised and unsupervised approaches
  - Using a small amount of labeled data boosts model accuracy
- Active learning: *intelligent sampling*
  - Selects the most important examples to label
  - Improves predictive accuracy
- Weak supervision:
  - Uses heuristics to apply noisy labels to unlabeled examples
  - Snorkel is handy framework for weak supervision



DeepLearning.AI

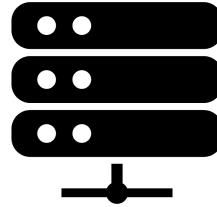
# Data Augmentation

---

# Data Augmentation

# Outline

- Generating synthetic data
- Augmenting an image dataset: CIFAR-10 example
- Other advanced techniques



# How do you get more data?

- Augmentation as a way to expand datasets
- One way is introducing minor alterations
- For images: flips, rotations, etc.



# How does augmentation data help?

- Adds examples that are similar to real examples
- Improves coverage of feature space
- Beware of invalid augmentations!



# An example: CIFAR-10 data set

- 60,000 32x32 color images
- 10 classes of objects  
(6,000 images per class)



horse (7)



ship (8)



deer (4)



deer (4)



frog (6)



dog (5)



bird (2)



truck (9)



frog (6)

# Data augmentation on CIFAR-10

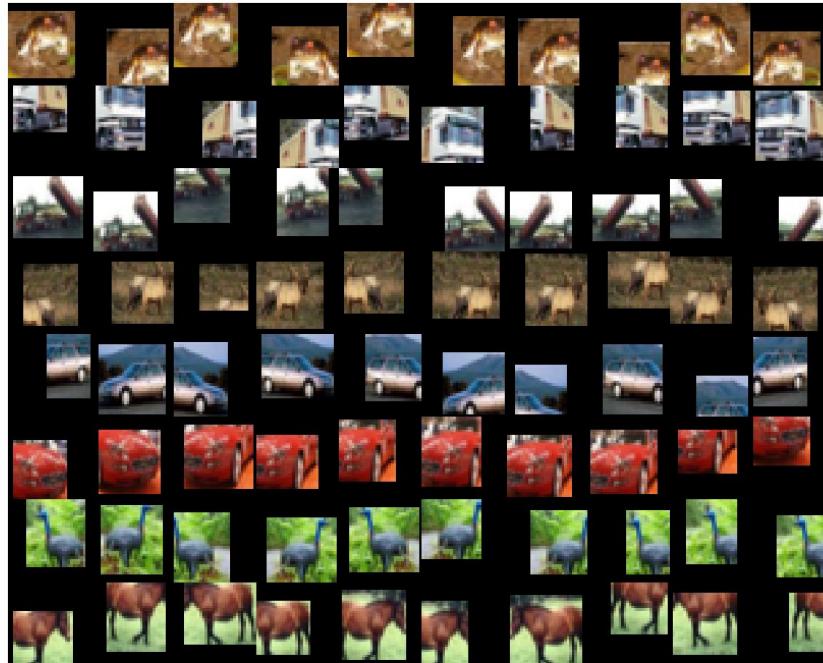
Let's augment the CIFAR-10 dataset with the following steps:

1. Pad the image with a black, four-pixel border
2. Randomly crop a  $32 \times 32$  region from the padded image
3. Flip a coin to determine if the image should be flipped horizontally left/right

# Defining the augment operation

```
def augment(x, height, width, num_channels):
    x = tf.image.resize_with_crop_or_pad(x, height + 8, width + 8)
    x = tf.image.random_crop(x, [height, width, num_channels])
    x = tf.image.random_flip_left_right(x)
    return x
```

# Augmented examples



# Other Advanced techniques

- Semi-supervised data augmentation  
e.g., UDA, semi-supervised learning with GANs
- Policy-based data augmentation e.g., AutoAugment

Given the low budget and production limitations, this movie is very good.

Since it was highly limited in terms of budget, and the production restrictions, the film was cheerful.

There are few budget items and production limitations to make this film a really good one.

# Key points on data augmentation

- It generates artificial data by creating new examples which are variants of the original data
- It increases the diversity and number of examples in the training data
- Provides means to improves accuracy, generalization, and avoiding overfitting

# Preprocessing More Data Types



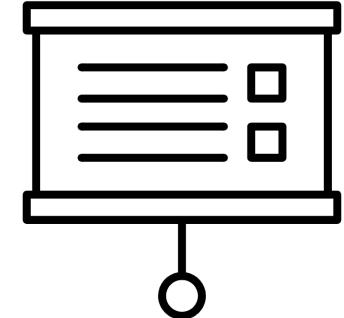
DeepLearning.AI

---

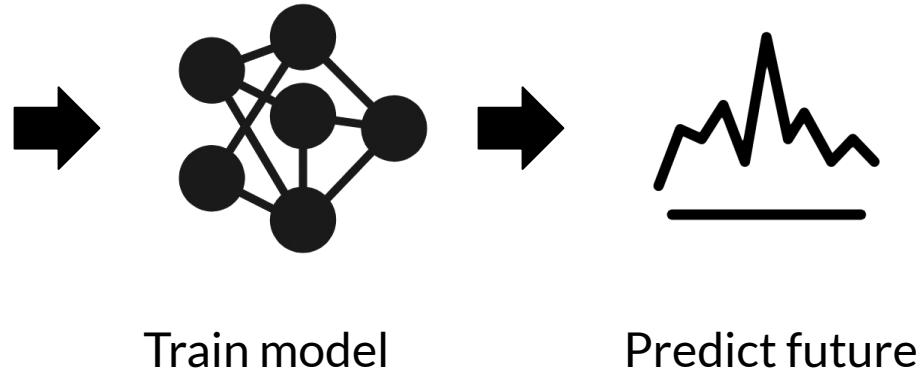
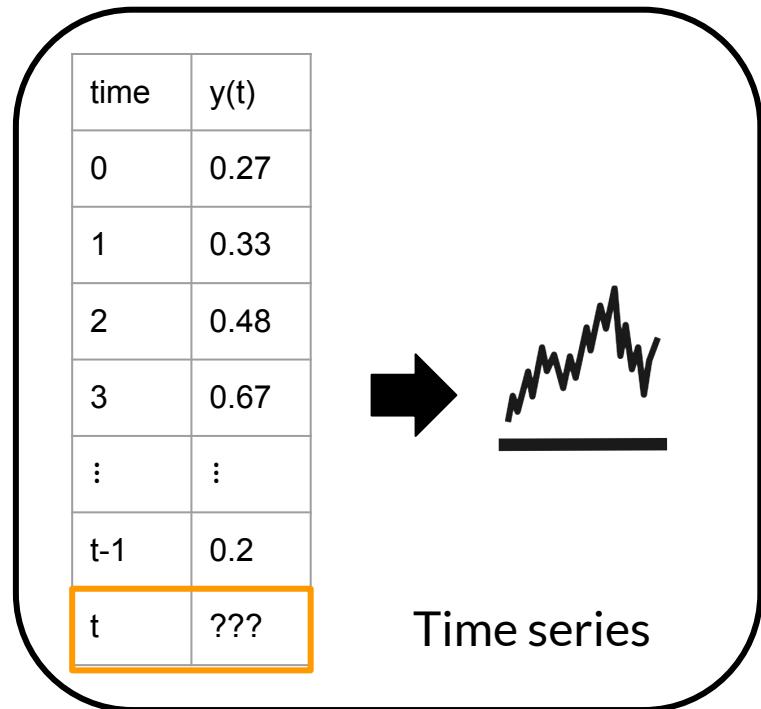
## Time series

# A note on different types of data

- TFX pre-processing capabilities for multiple data types:
  - Images
  - Video
  - Text
  - Audio
  - Time series
- Optional notebook on images
- Two optional notebooks on time series



# Time series data



*“It is difficult to make predictions, especially about the future.”*

- Karl Kristian Steincke

# Time series forecasting

- Time Series forecasting predicts future events by analyzing data from the past
- Time series forecasting makes predictions on data indexed by time
- Example:
  - Predict future temperature at a given location based on historical meteorological data

# Time series dataset: Weather prediction

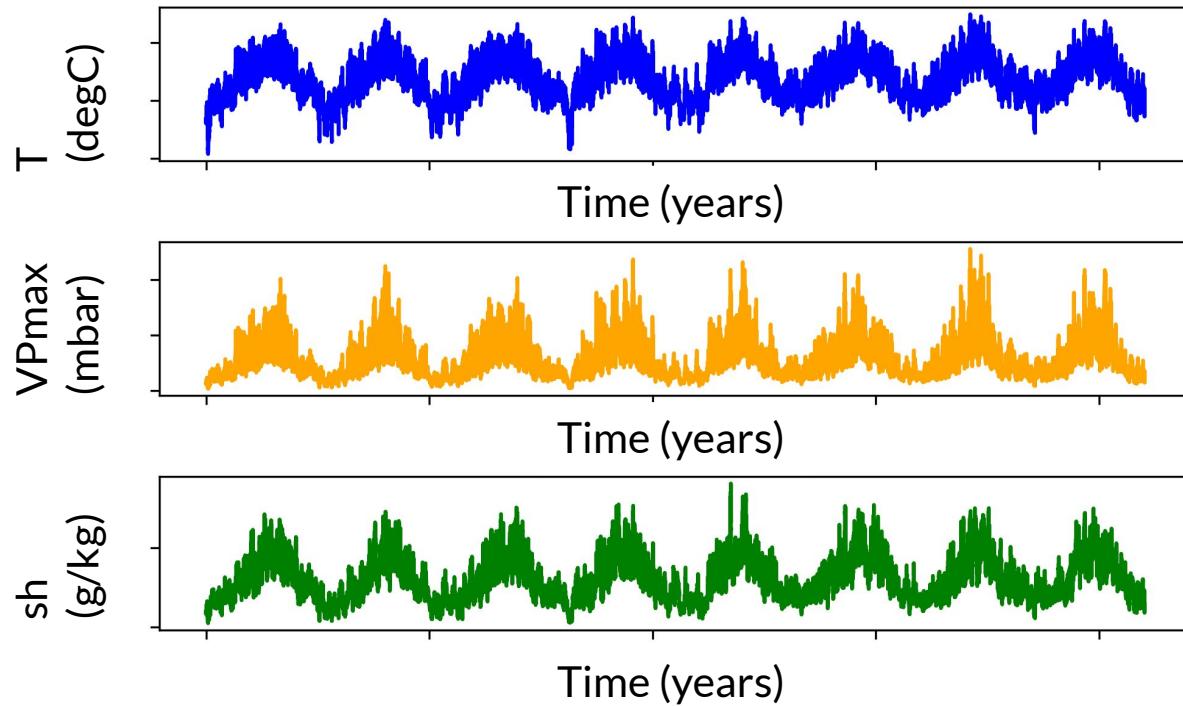
We will use the weather time series dataset recorded by the Max Planck Institute for Biogeochemistry:

- to preprocess time series data with TensorFlow Transform
- to convert data into sequences of time steps:
  - Making data ready to train a long short-term memory (LSTM)  
recurrent neural network (RNN)

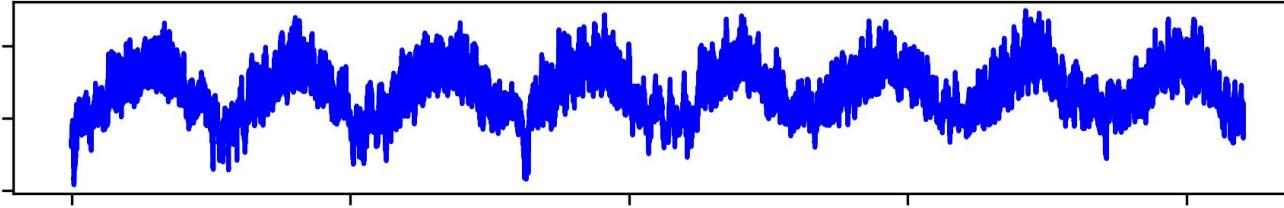
# Weather time series dataset

- There are 14 variables:
  - P(mbar), T (degC), Tdew (degC), rh (%), VPmax (mbar), VPact (mbar), VPdef (mbar), sh (g/kg), H20C (mmol/mol), rho (g/m<sup>\*\*</sup>3), wv (m/s), max.xv (m/s), wd (deg)
  - Target is T (degC)
- Observations recorded every 10 minutes
  - 6 observations per hour
  - 144 (6 X 24) observations in a day.

# Data visualizations



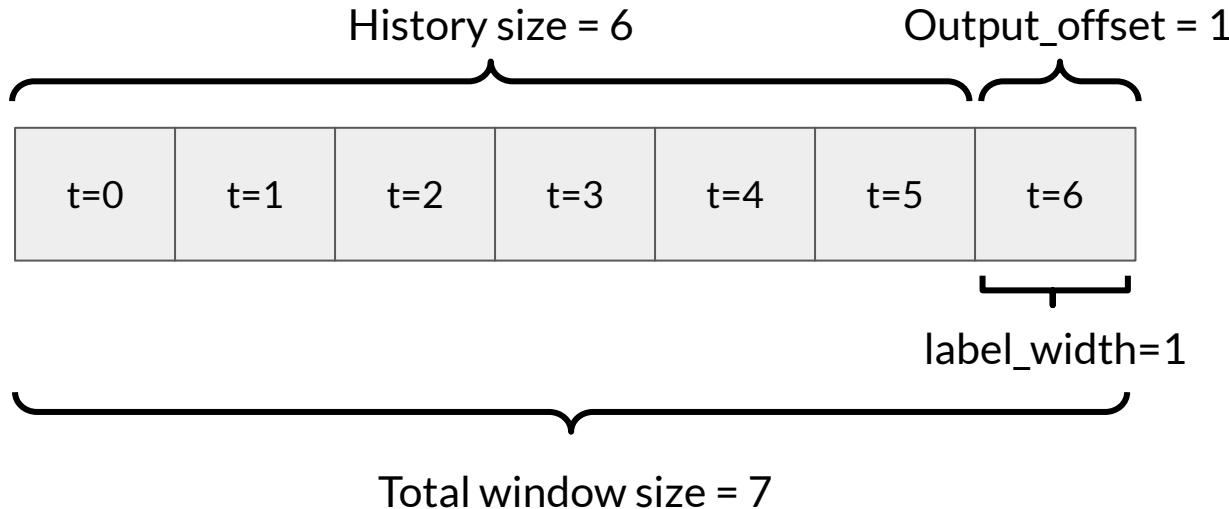
# Windowing data



- Windowing makes sense.
- `tf.data.Datasets.window()` converts times series data to depend on past observations.

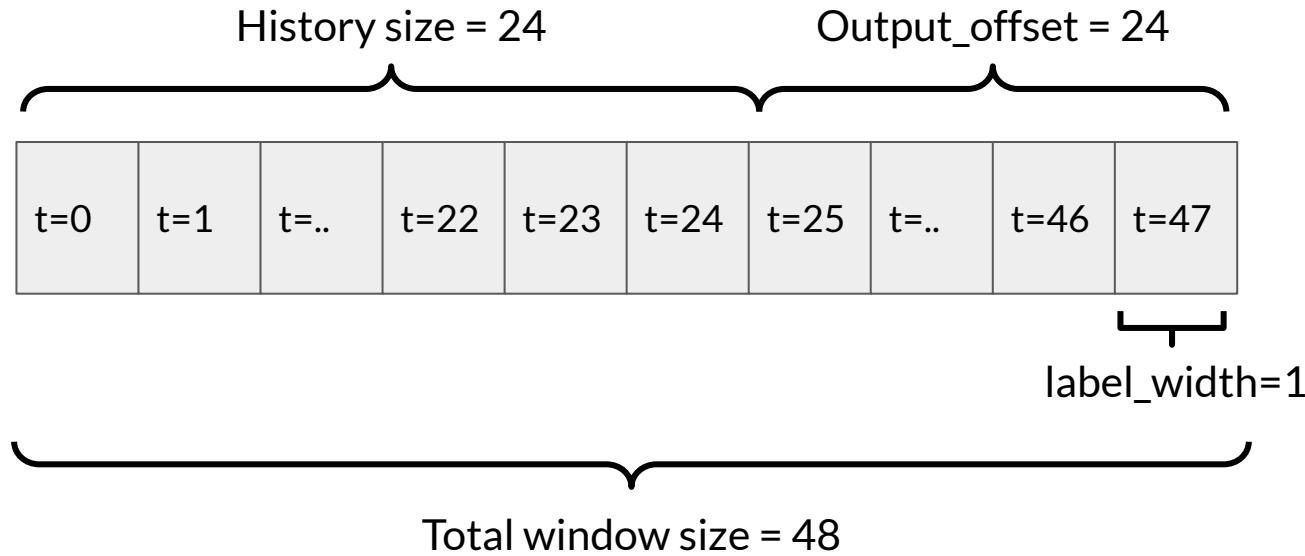
# Windowing strategies in single step time series

A model that makes a prediction 1h into the future, given 6h of history would need a window like this:



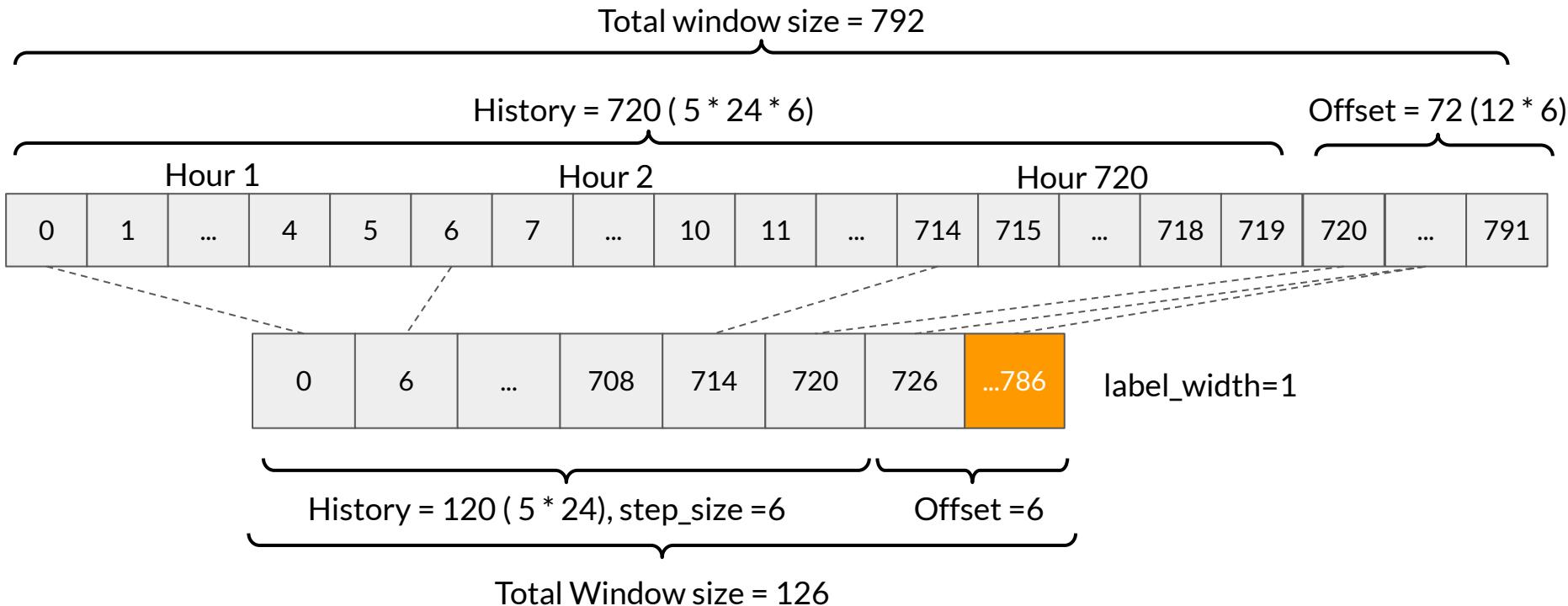
# Windowing strategies in single step time series

Predict next 24 hours given 24 hours of history



# Windowing strategy for the problem

Sample one observation every hour with step size = 6



# Optional notebook: what will you do?

- Data processing with TFX to extract features
- Segment data into windows
- Save data in TFRecord format
- Make it ready for training an LSTM model

# Preprocessing More Data Types



DeepLearning.AI

---

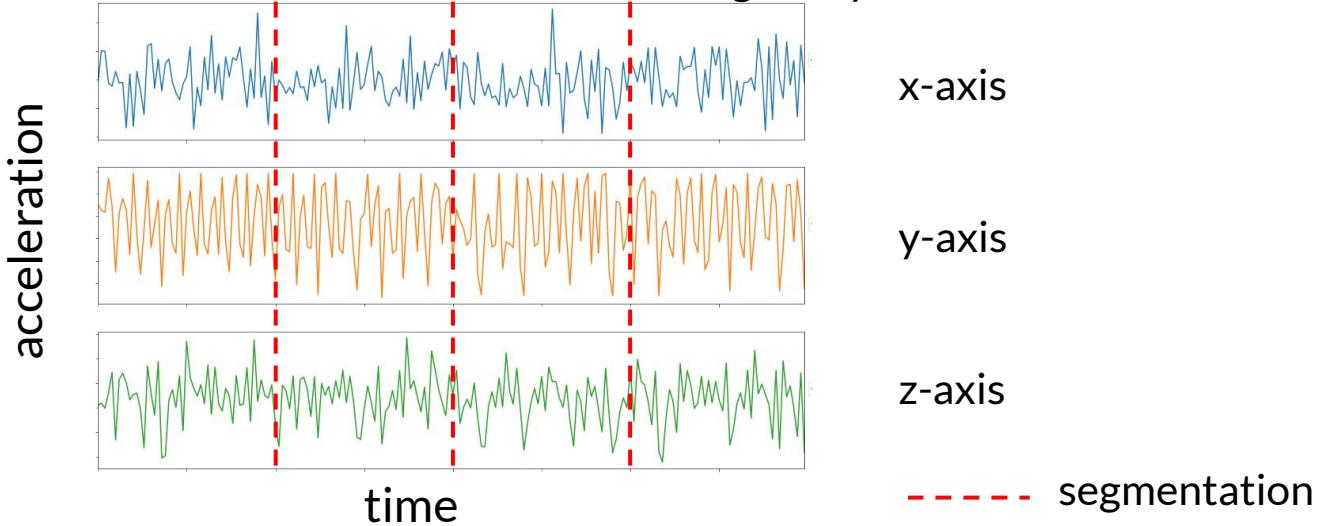
## Sensors and Signals

# Sensors and Signals

- Signals are sequences of data collected from real time sensors
- Each data point is indexed by a timestamp
- Sensors and signals data is thus time series data
- Example: classify sequences of accelerometer data recorded by the sensors on smartphones to identify the associate activity

# Human activity recognition (HAR)

- HAR tasks require segmentation operations
  - Raw inertial data from wearables fluctuate greatly over time



# Human activity recognition (HAR)

- Segmented data should be **transformed** for modeling
- Different methods of transformation:
  - Spectrograms (commonly used)  
光谱的，声谱的
  - Normalization and encoding
  - Multichannel
  - Fourier transform

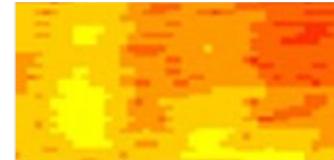
Raw Data



Multichannel Data



Spectrogram



# Optional notebook: what will you do?

- Work with Human Activity Recognition Dataset (WISDM):
  - Preprocess with TensorFlow Transform
  - Use `tf.data.Datasets.window()` for converting times series data to depend on past observations

# Preprocessing More Data Types



DeepLearning.AI

---

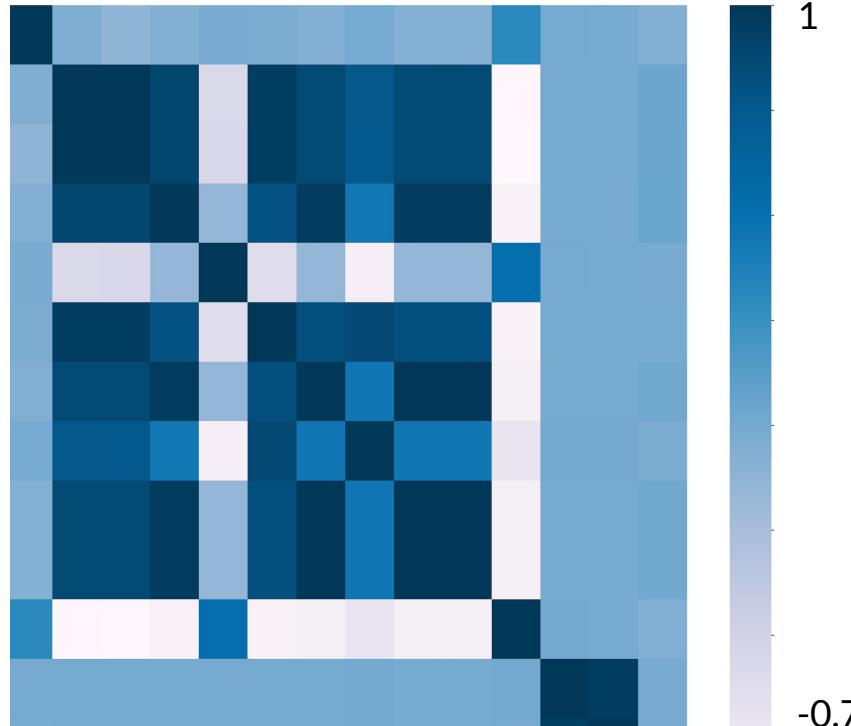
# Time Series Assignment Walkthrough

# Detecting inconsistent data

	count	mean	std	min	25%	50%	75%	max
wv (m/s)	420551	1.702	65.44	-9999.0	0.99	1.76	2.86	28.49
max. wv (m/s)	420551	3.06	69.01	-9999.0	1.76	2.96	4.74	23.5

# Feature correlation

p (mbar)  
T (degC)  
Tpot (K)  
Tdew (degC)  
rh (%)  
VPmax(mbar)  
VPact (mbar)  
VPdef (mbar)  
sh (g/kg)  
H2OC(mmol/mol)  
rho (g/m\*\*3)  
wv (m/s)  
max. wv (m/s)  
wd (deg)

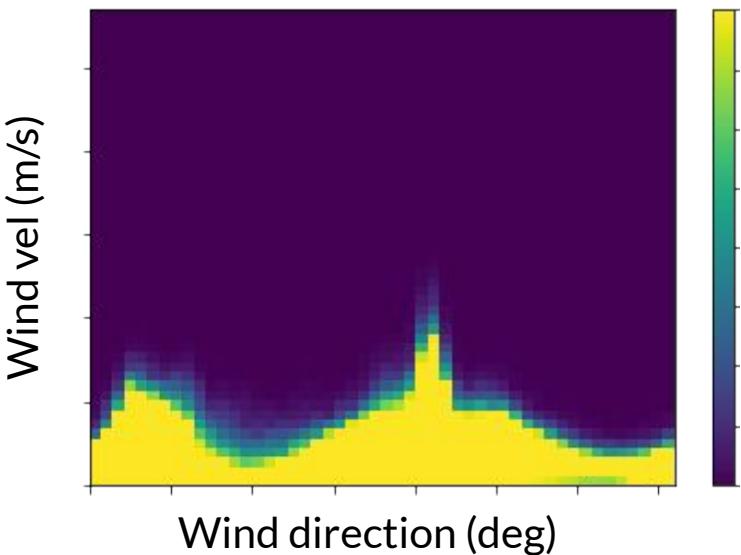


**Features to Remove due to high correlations:**

1. Tpot (k)
2. Tdew (degC)
3. VPact
4. H2OC
5. max. wv

# Consistent wind direction and velocity

Distribution of Wind Data



- Wind direction is in units of degrees
- $360^0$  and  $0^0$  should be close to each other and wrap around smoothly
- Wind direction doesn't matter if wind is not blowing
- Using TF Transform we will convert wind direction and wind velocity into wind vector

# Preprocessing the date time feature

- Date Time columns is in string format
- Weather data has clear daily and yearly periodicity

Transformation Needed:

- Convert string date time to timestamp
- Use sin and cos to convert it into 2 features:
  - Time of the day
  - Time of the year

# Reading and cleaning input data

- Read the Input data using `beam.io.ReadFromText()`
- Clean data using a Beam Transform:
  - Decode the input lines to transform into feature value pairs using a schema
  - Remove extreme min values of -9999.0 from wind velocity and max wind velocity features
  - Convert Date Time feature to timestamp

# Train test splits

- First 300,000 records are used for training, the remaining for testing
- You will partition the dataset using the `beam.Partition` transform
- `beam.Partition` needs a partition function which defines logic of partition

# Preprocessing the dataset

- Delete unwanted features (Feature Selection)
- Transform Wind Direction and Wind Velocity to a wind vector
- Transform timestamp DateTime to ‘Time of Year’ and ‘Time of Day’
- Normalize float features

# Advanced Labeling, Augmentation, and Preprocessing

## Semi-supervised labeling

- Graph-based approach
- Active learning

## Weak supervision

- Snorkel

## Data augmentation

- Image transformations
- Policy-based

## Time series

- Windowing
- Sensors and signals