COMP2261 ARTIFICIAL INTELLIGENCE / MACHINE LEARNING

# Chapter 1: Data Exploration, Preprocessing, and Problem Framing (4 hours)

Dr Yang Long

Durham University

# Submodule Overview

## Curricula Context

| | |
|---|---|
| Math | COMP1021 Mathematics for Computer Science – Linear Algebra, Calculus |
| | COMP2271 Data Science – Probability |
| Programming | COMP1051 Computational Thinking – Python, NumPy, Matplotlib, Jupyter Notebook |
| | COMP1081 Algorithms and Data Structures – Sorting, Searching, Graph, String |

**Machine Learning**

| | |
|---|---|
| Advancement | COMP2261 Artificial Intelligence – Bias in AI |
| | COMP3547 Deep Learning and Reinforcement Learning |
| Applications | COMP3517 Computational Modelling in the Humanities and Social Sciences |
| | COMP3527 Computer Vision |
| | COMP3647 Human-AI Interaction Design |
| | COMP4157 Learning Analytics |
| | COMP4167 Natural Language Processing |
| | … |

# Submodule Overview

## Content

- Philosophy behind machine learning
- Fundamental concepts in machine learning
- Machine learning workflow
- Defining machine learning tasks
- Data preparation for machine learning
- Model selection and evaluation
- Implement machine learning algorithms using Python and scikit-learn
- Interpreting results

# Submodule Overview

## Learning Outcomes

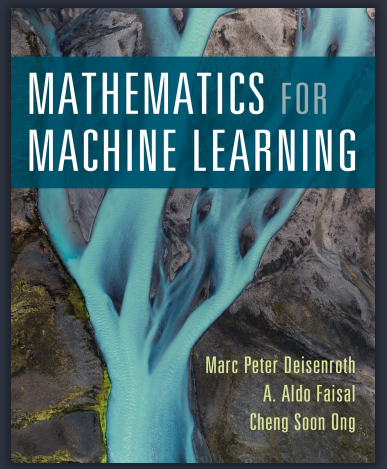
- Understand key principles of ML for use in managing dataset and building models
- Understand differences between supervised learning and unsupervised learning
- Understand the math behind ML models and algorithms
- Be able to select and implement appropriate learning algorithms for real-life problems
- Be able to train, optimise, evaluate, and compare ML models
- Be able to scientifically report the result of machine learning projects

# Submodule Overview

## Coding Environment

# Submodule Overview

## Recommended books




### Mathematics for Machine Learning

Marc Peter Deisenroth, A. Aldo Faisal, and Cheng Soon Ong. Cambridge University Press.

So that you know the math behind the scene.

https://mml-book.github.io


### Mastering Machine Learning with scikit-learn (Second Edition)

Gavin Hackeling. Packt Publishing Limited.

So that you know what you are doing in Practicals and coursework.

http://www.smallake.kr/wp-content/uploads/2017/03/Mastering-Machine-Learning-with-scikit-learn.pdf

# What's Machine Learning (ML)

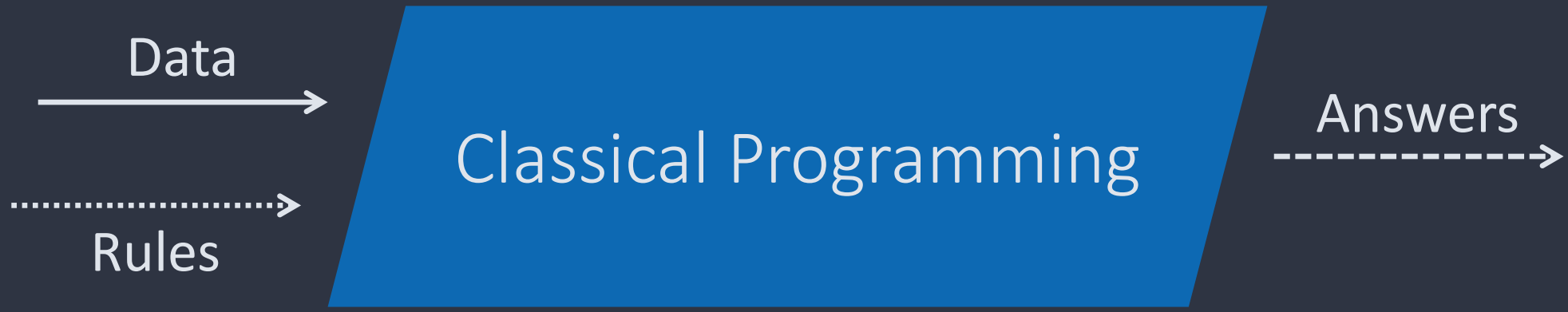
Machine learning arose from a simple question:

Can a computer learn on its own in order to perform a specified task, rather than relying on a programmer explicitly setting up the rules for it?

# What’s Machine Learning (ML)

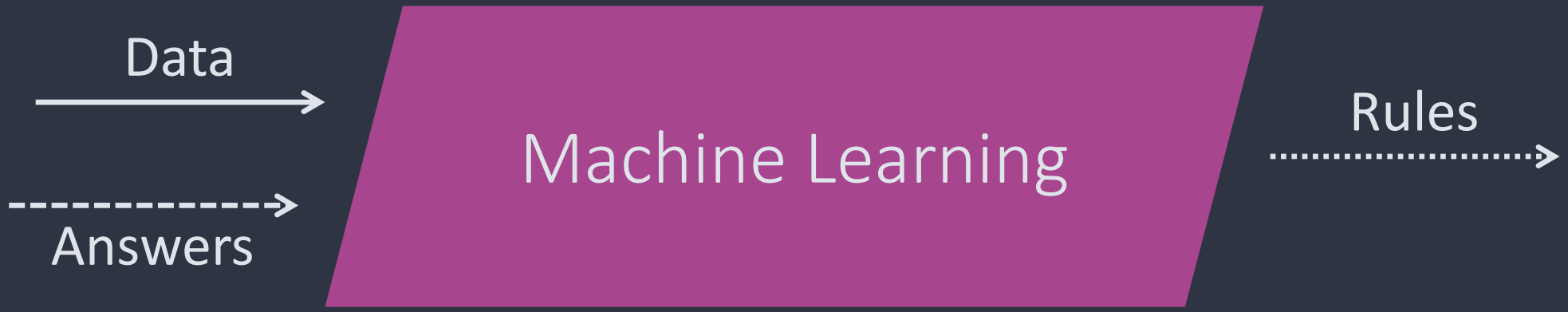
## Classical Programming vs Machine Learning

# What's Machine Learning (ML)



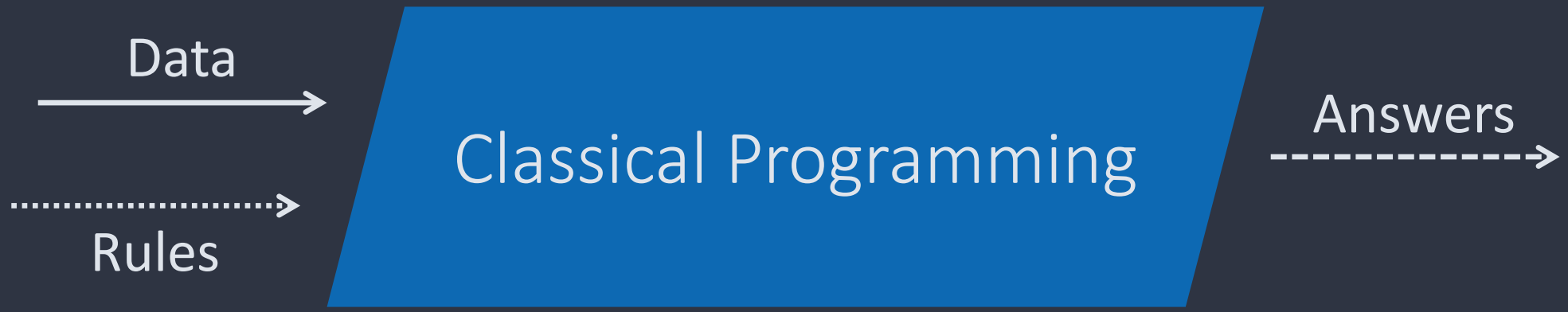

- Rules rely on human understanding and manually coding.
- Not always work, as there are problems not possible to identify and explicitly write down rules.

# What's Machine Learning (ML)

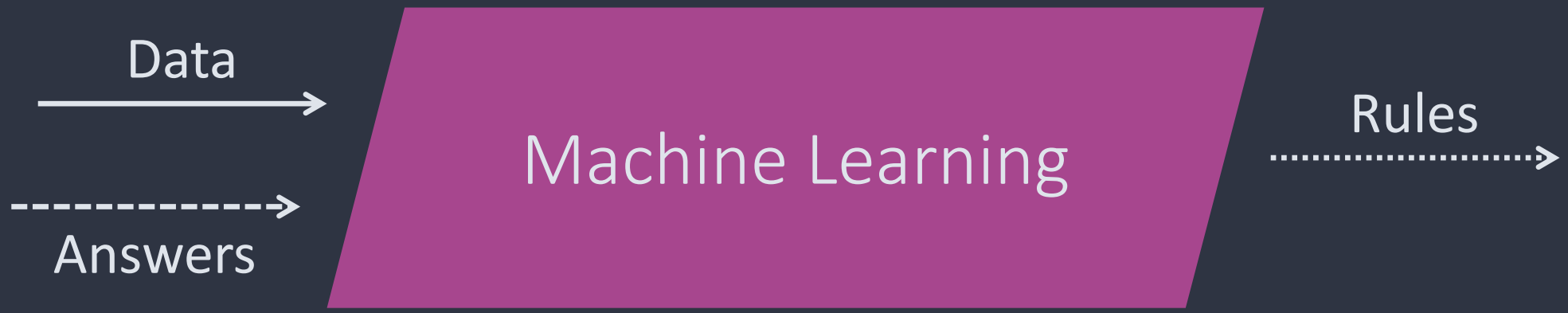



- Human provides data and anticipated answer from the data.
- Computers outputs a set of rules, which can be applied to new data.

# What's Machine Learning (ML)

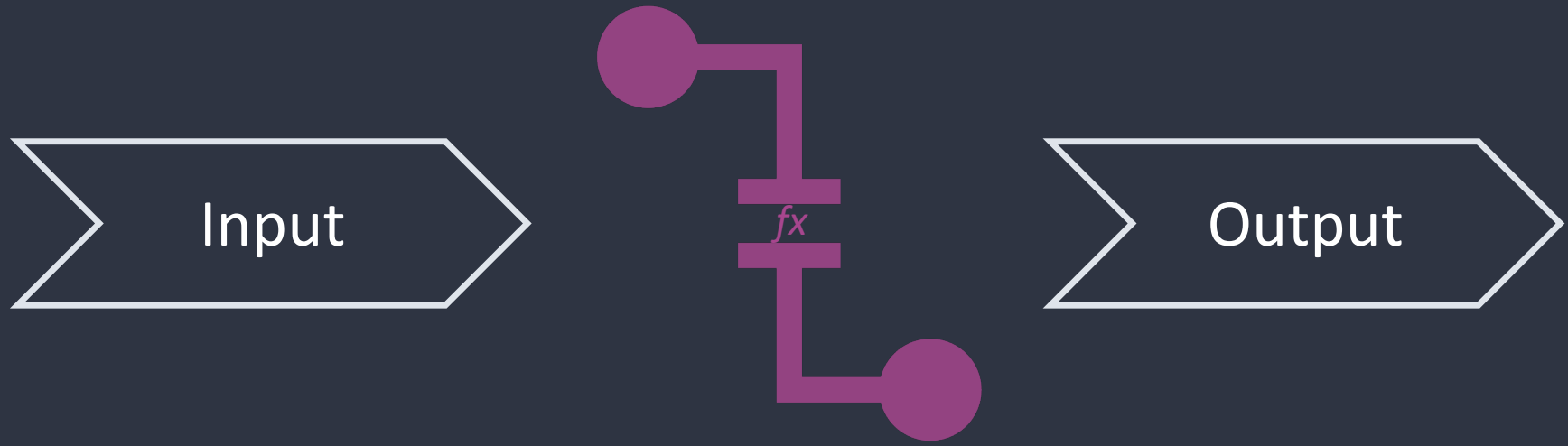



The programmer learns the rules from observing the data




The machine learns the rules from observing the data

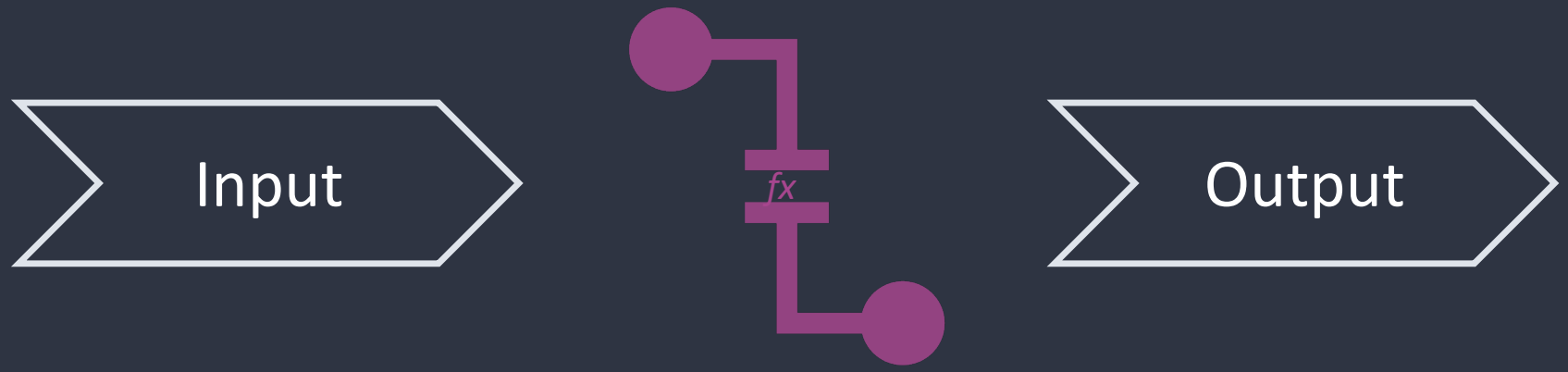
# Machine Learning / Supervised Learning




- Machine learns mappings from input to output.
- When given input data, machine gives prediction of the output for that data.

# Machine Learning / Supervised Learning

EXAMPLE.




| | | |
|---|---|---|
| savings | Credit Scoring | credit score |
| tumour size | Cancer Detection | is malignant or not |
| money transaction | Fraud Detection | is legitimate or not |

… …

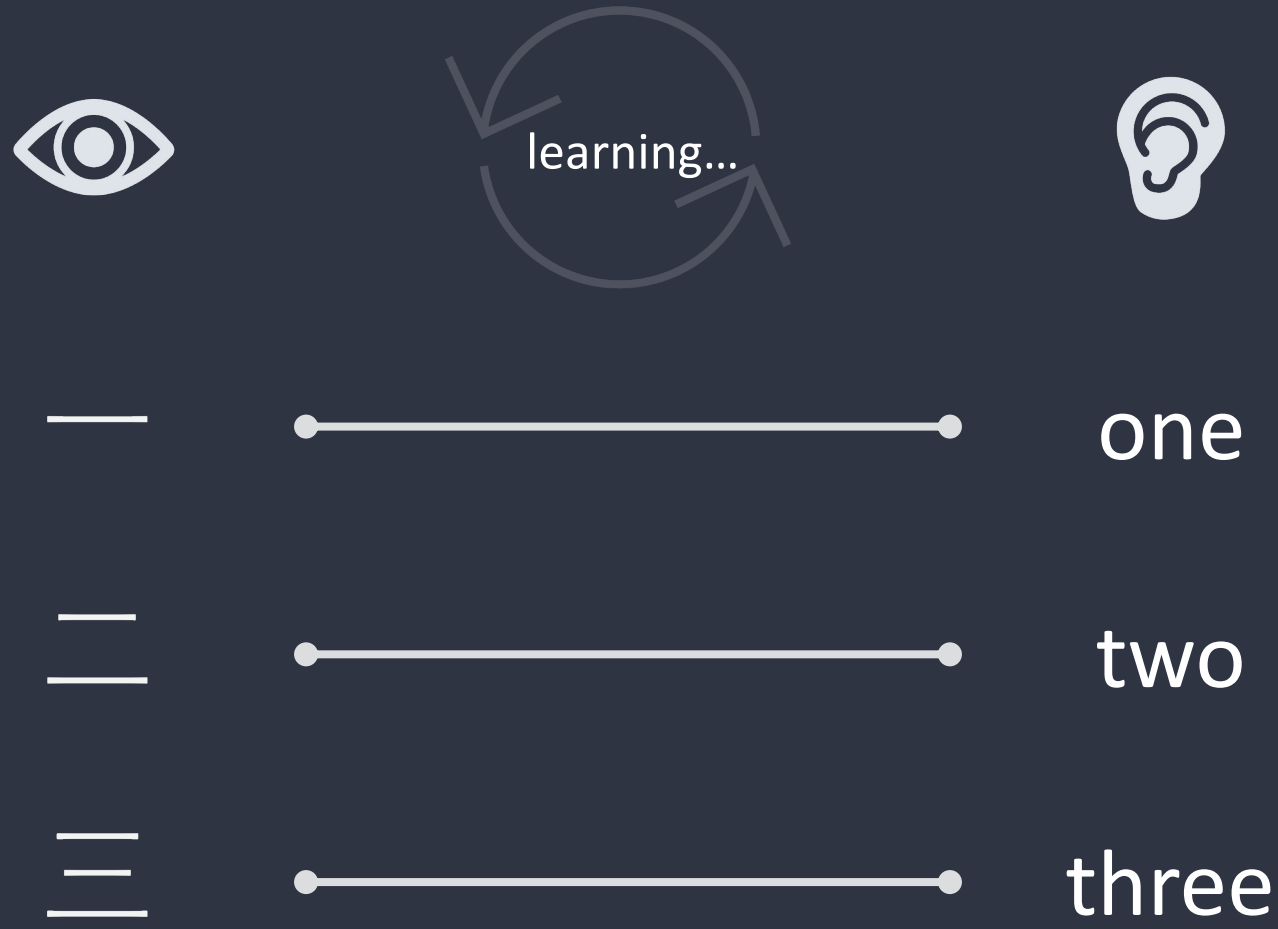
# Machine Learning / Supervised Learning

EXAMPLE.


- The teacher knows the correct answers – supervising the learning process.
- The algorithm iteratively makes predictions on data, corrected by the teacher.
- The learning process stops at a certain acceptable level of performance.

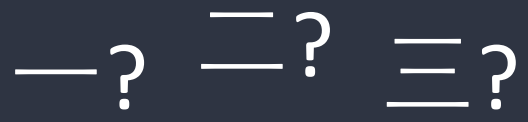
# Machine Learning / Supervised Learning

EXAMPLE. Handwriting Recognition



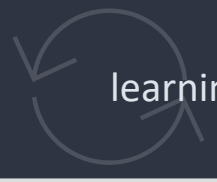
# Machine Learning / Supervised Learning

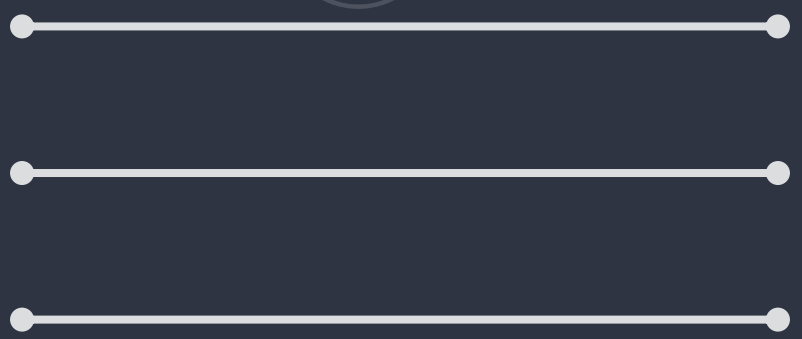
EXAMPLE. Handwriting Recognition





training set



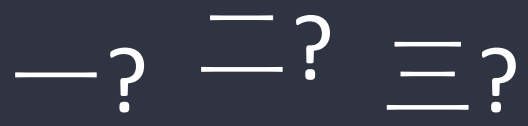
one

two

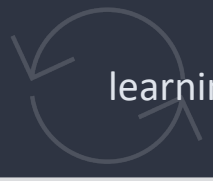
three

# Machine Learning / Supervised Learning

EXAMPLE. Handwriting Recognition








Number of lines
- feature -

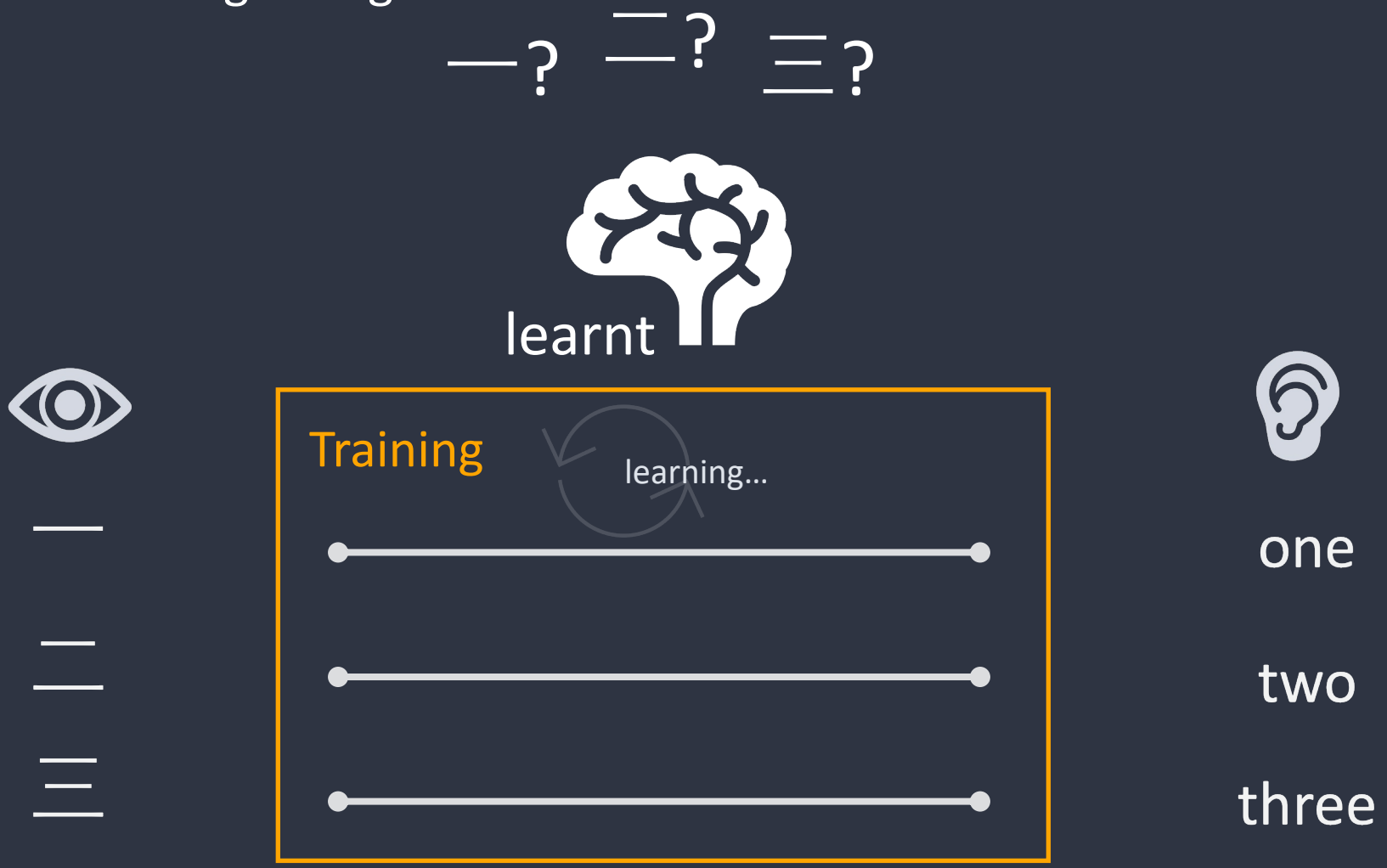
one

two

three

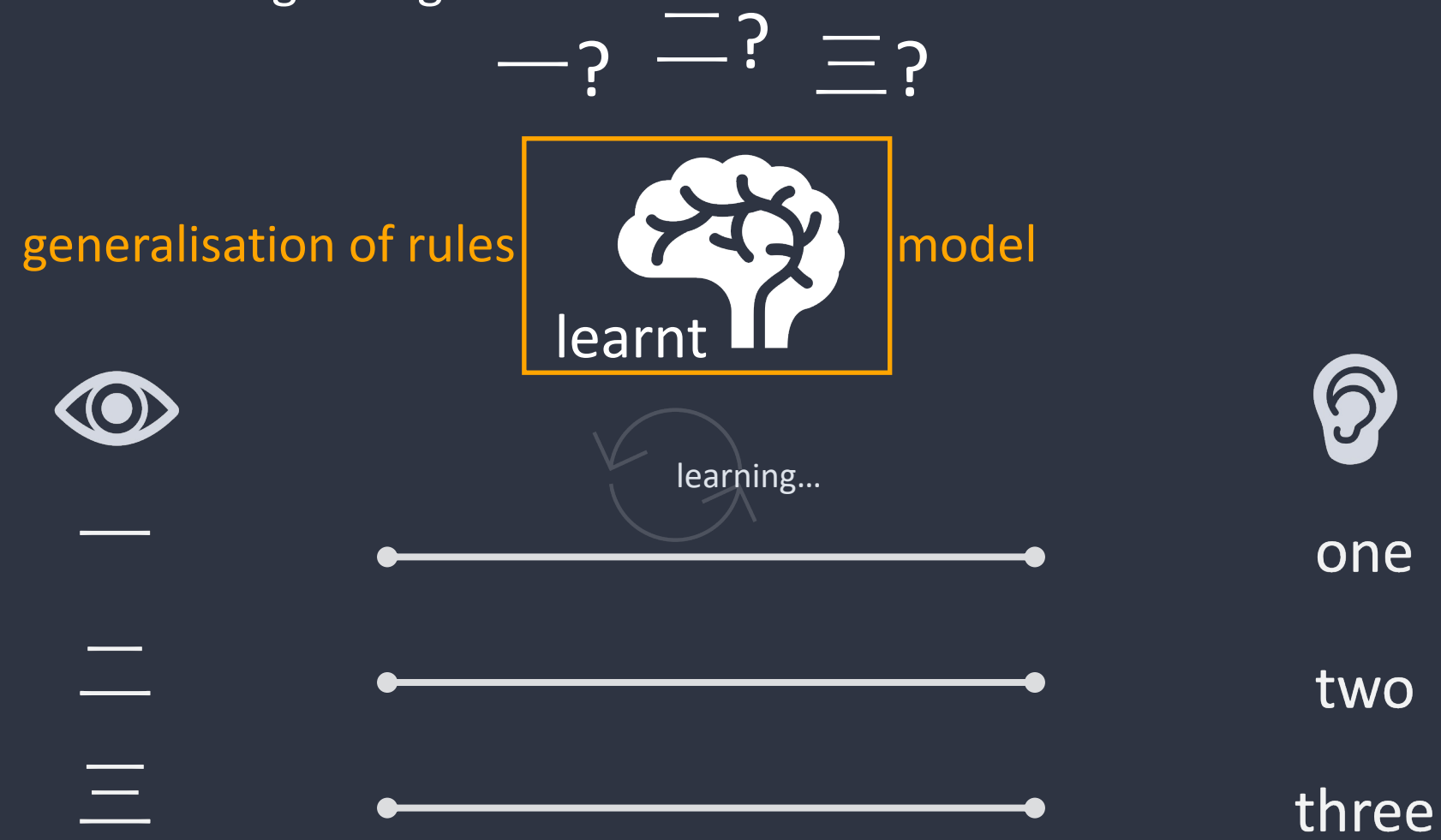
# Machine Learning / Supervised Learning

EXAMPLE. Handwriting Recognition



# Machine Learning / Supervised Learning

EXAMPLE. Handwriting Recognition



# Machine Learning / Supervised Learning

EXAMPLE. Handwriting Recognition

To summarise…

**Machine Learning** is the process of using a training set to learn features, continue modelling, and finally producing an efficient model.

# 3. Three types of Machine Learning

# Three types of Machine Learning

## (1) Supervised Learning


- To learn the mapping (rules) between inputs and outputs
- Labelled data (correct answers) is provided of past input & output pairs during the learning process to train the model how it should behave for previously unseen data.

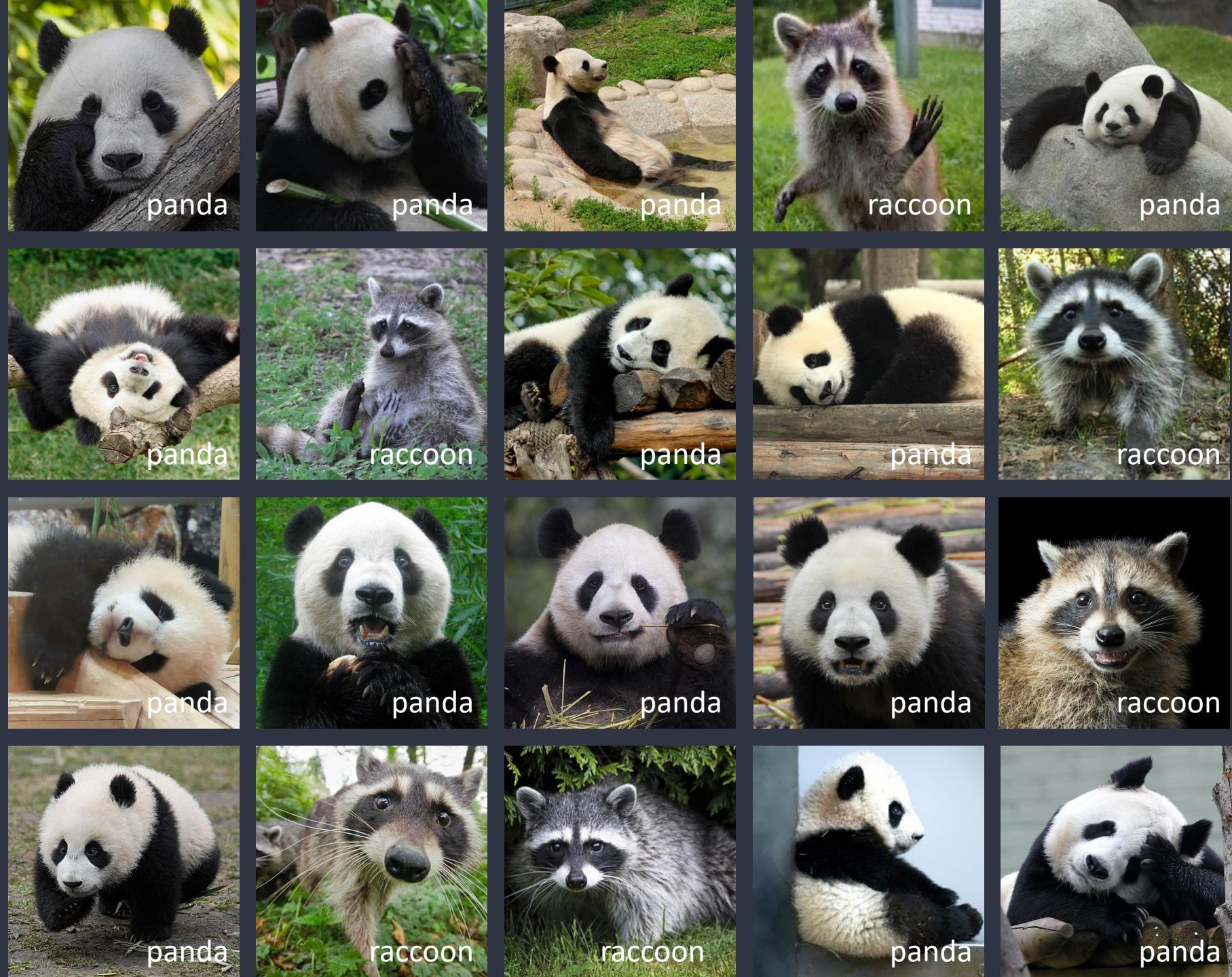
EXAMPLE.

We have many photos of panda and raccoon, and we want our machine to learn to recognise them.

learning…

learnt

to recognise new photos



# Three types of Machine Learning

## (1) Supervised Learning


- To learn the mapping (rules) between inputs and outputs
- Labelled data (correct answers) is provided of past input & output pairs during the learning process to train the model how it should behave for previously unseen data.

# Three types of Machine Learning

## (1) Supervised Learning


- To learn the mapping (rules) between inputs and outputs
- Labelled data (correct answers) is provided of past input & output pairs during the learning process to train the model how it should behave for previously unseen data.

## (2) Unsupervised Learning

- To learn hidden pattern (rules) from a set of inputs (no output).
- Unlabelled data is provided of past input (not a input & output pair) during the learning process. (no correct answers)
- Examples in the same group are more similar to each other than to those in other groups.

EXAMPLE.

We have many photos of panda and raccoon, and we want our machine to learn to recognise them.

learning…

# EXAMPLE.

We have many photos of panda and raccoon, and we want our machine to learn to recognise them.

learning…

learnt

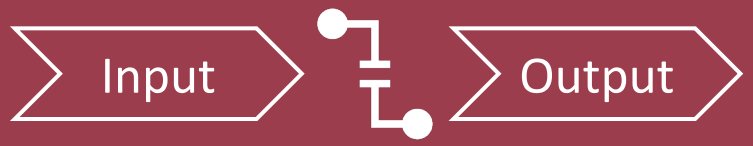
to group new photos

Group 1

Group 2

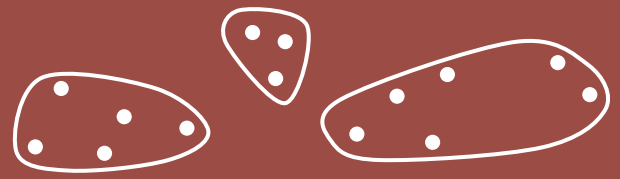
# Three types of Machine Learning

## (1) Supervised Learning




- To learn the mapping (rules) between inputs and outputs
- Labelled data is provided of past input & output pairs during the learning process to train the model how it should behave for previously unseen data.

## (2) Unsupervised Learning



- To learn hidden pattern (rules) from a set of inputs (no output).
- Unlabelled data is provided of past input (not a input & output pair) during the learning process.
- Instances in the same group are more similar to each other than to those in other groups.

## (3)Reinforcement Learning


- Occasional positive & negative feedback to reinforce behaviours.
- Good behaviours are rewarded with treat → more common. Bad ones are punished → less common.
- Balancing between exploration (of uncharted territory) and exploitation (of current knowledge)

# Machine Learning: a Definition

> A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.
>
> -- Tom Mitchell, 1997

# Machine Learning: a Definition

> A computer program is said to learn from experience $E$ with respect to some class of tasks $T$ and performance measure $P$, if its performance at tasks in $T$, as measured by $P$, improves with experience $E$.

Machine learning is the study of algorithms that

- Improve over task $T$ ← Recognise panda / raccoon
- with respect to performance measure $P$ ← % of photos recognised correctly (accuracy)
- based on experience $E$ ← learning from those photos (training data)

A well-defined learning task is given by $<T, P, E>$.

# Summary

Submodule overview – what you should be expecting and doing.

What's Machine Learning <T, P, E>.

Three types of ML: supervised, unsupervised, reinforcement learning.

ML terms: training, training set, example, feature, label, model.

## Next Lecture

Machine Learning Workflow

Coding Environment

COMP2261 ARTIFICIAL INTELLIGENCE / MACHINE LEARNING

# Machine Learning Workflow & Coding Environment

Dr Yang Long

Durham University

# Last Lecture

## What's machine learning?

"field of study that gives computers the ability to learn without being explicitly programmed." -- Arthur Samuel, 1959

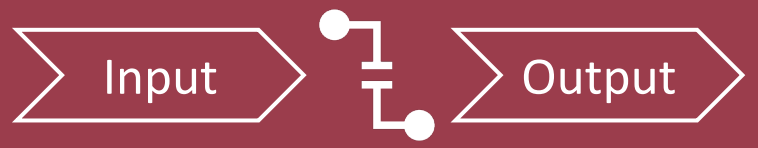
"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E." -- Tom Mitchell, 1997

Classical Programming vs Machine Learning

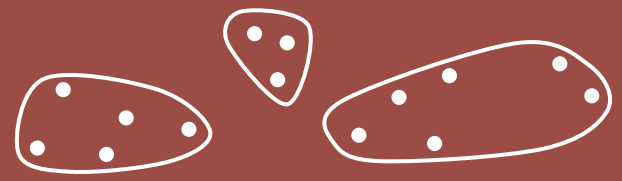
# Last Lecture

## Three types of machine learning

### (1) Supervised Learning




### (2) Unsupervised Learning



### (3)Reinforcement Learning


## Machine learning terms

Training, training set, example, feature, label, model

# Machine Learning Workflow

learn from data

train a model

to

perform tasks

infer / predict

These 2 parts broadly outline the 2 sides of ML – both are equally important.

# Machine Learning Workflow

train a model : use the data to create a model and fine-tune the model to make better predictions.

we can then use this predictive model to make predictions on previously unseen data in order to perform specific tasks.

# Machine Learning Workflow

How does machine learning work in real life?

EXAMPLE. to build a tea detection application

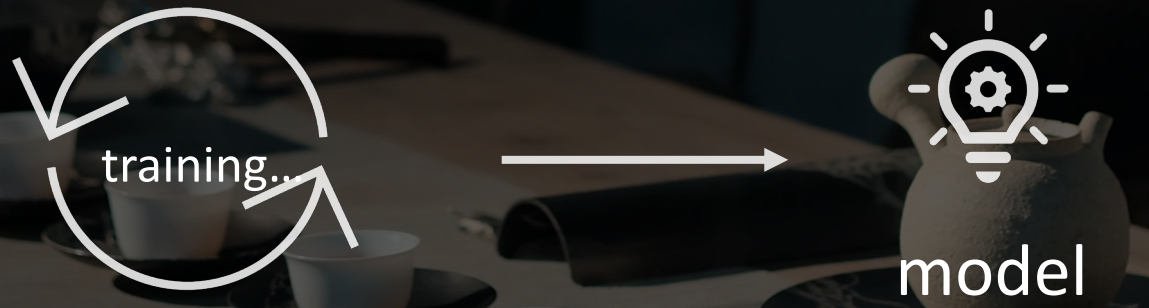
1. Problem Framing

Green tea / Oolong tea

model

1. Problem Framing

# Green tea / Oolong tea



1. Problem Framing

# Green tea / Oolong tea


Measure of success
(prediction accuracy)

# Green tea / Oolong tea

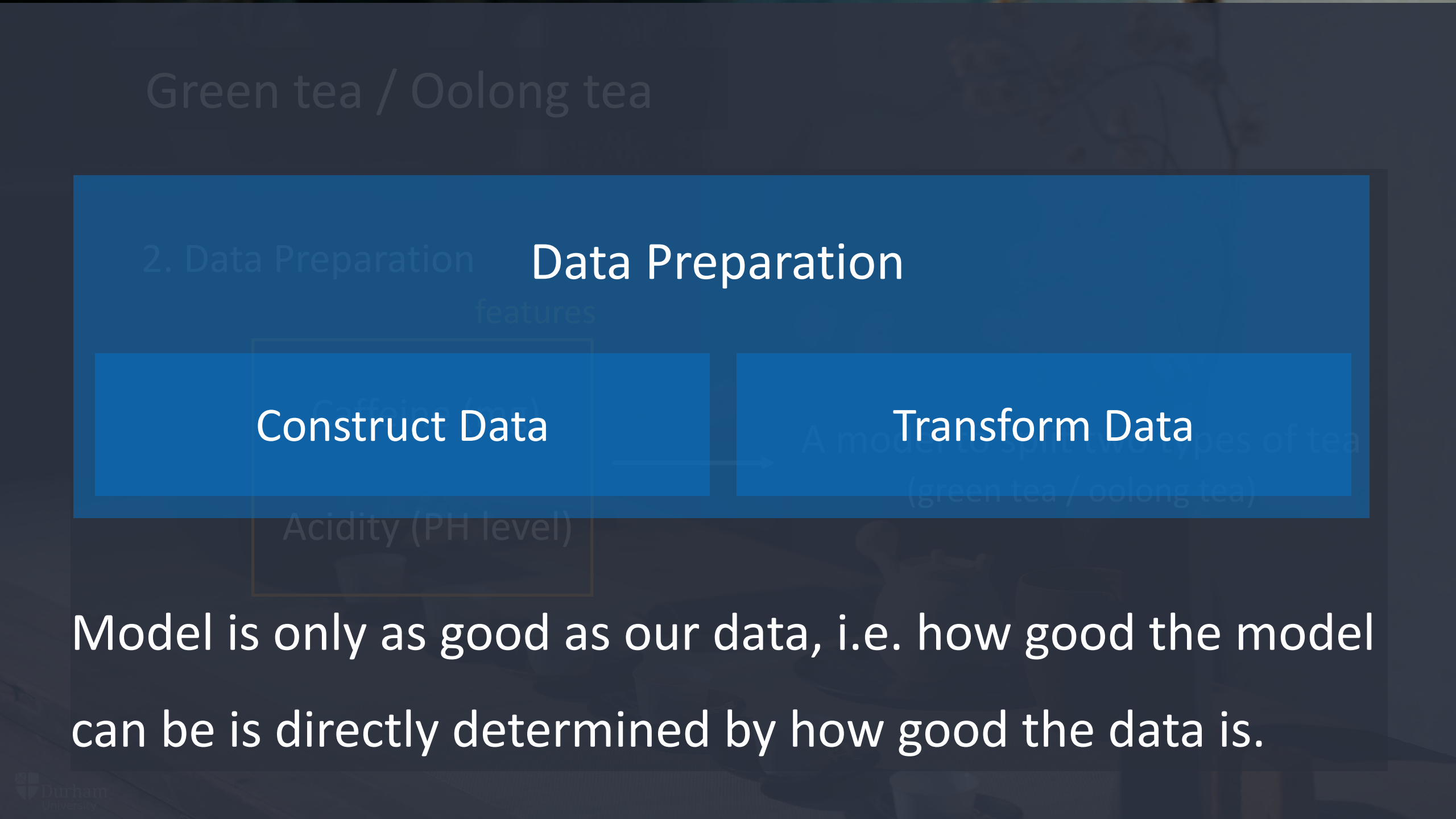
## 2. Data Preparation

features

Caffeine (mg)

Acidity (PH level)

→ A model to split two types of tea (green tea / oolong tea)




## Data Preparation

Construct Data → Transform Data

Model is only as good as our data, i.e. how good the model can be is directly determined by how good the data is.

# Green tea / Oolong tea

## 2. Data Preparation

training set

feature feature label

| Caffeine (mg) | Acidity (PH level) | Green tea or oolong tea? |
|---|---|---|
| 50 | 5.8 | Green tea |
| 0.65 | 6.2 | Oolong tea |
| 40 | 7.9 | |
| 60 | 6.5 | Green tea |
| 70 | | Oolong tea |

example

# Green tea / Oolong tea

## 2. Data Preparation

| Caffeine (mg) | Acidity (PH level) | Green tea or oolong tea? |
|---|---|---|
| 50 | 5.8 | Green tea |
| 0.65 | 6.2 | Oolong tea |
| 40 | 7.9 | |
| 60 | 6.5 | Green tea |
| 70 | | Oolong tea |

# Green tea / Oolong tea

## 2. Data Preparation

Issues from dataset

| Caffeine (mg) | Acidity (PH level) | Green tea or oolong tea? |
|---|---|---|
| 50 | 5.8 | Green tea |
| 0.65 too small | 6.2 | Oolong tea |
| 40 | 7.9 | missing |
| 60 | 6.5 | Green tea |
| 70 | missing | Oolong tea |

# Green tea / Oolong tea

## 2. Data Preparation

| Caffeine (mg) | Acidity (PH level) | Green tea or oolong tea? | |
|---|---|---|---|
| 50 | 5.8 | Green tea | label |
| 65 | 6.2 | Oolong tea | label |
| 40 | 7.9 | Green tea | label |
| 60 | 6.5 | Green tea | label |
| 70 | 7.8 | Oolong tea | label |

# Green tea / Oolong tea

## 2. Data Preparation

transform label: (oolong -> 0; green -> 1)

| Caffeine (mg) | Acidity (PH level) | Green tea or oolong tea? |
|---|---|---|
| 50 | 5.8 | 1 |
| 65 | 6.2 | 0 |
| 40 | 7.9 | 1 |
| 60 | 6.5 | 1 |
| 70 | 7.8 | 0 |

# Green tea / Oolong tea

## 2. Data Preparation

training set

| Caffeine (mg) | Acidity (PH level) | Green tea or oolong tea? |
|---|---|---|
| 50 | 5.8 | 1 |
| 65 | 6.2 | 0 |
| 40 | 7.9 | 1 |
| | | |
| | | |

test set

| Caffeine (mg) | Acidity (PH level) | Green tea or oolong tea? |
|---|---|---|
| | | |
| | | |
| | | |
| 60 | 6.5 | 1 |
| 70 | 7.8 | 0 |

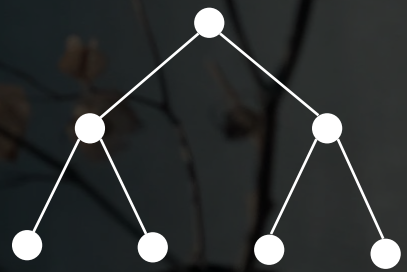
# Green tea / Oolong tea

## 3. Algorithm Selection

Regression

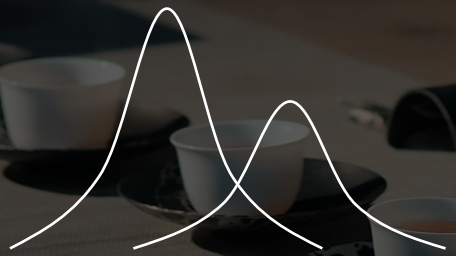
Regularisation

Instance-based

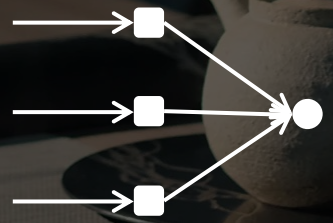


Tree-based

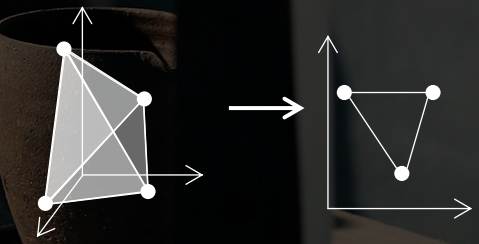

Clustering



Bayesian



Ensemble



Dimensionality Reduction

# Green tea / Oolong tea

## 3. Algorithm Selection

Linear Regression algorithm

# Green tea / Oolong tea

## 4. Model Training

Data preparation and model selection are more important.

A machine learning engineer normally spent much more time in data preparation than in model training.

How good our model can be is directly determined by how good our data is – "garbage in garbage out".

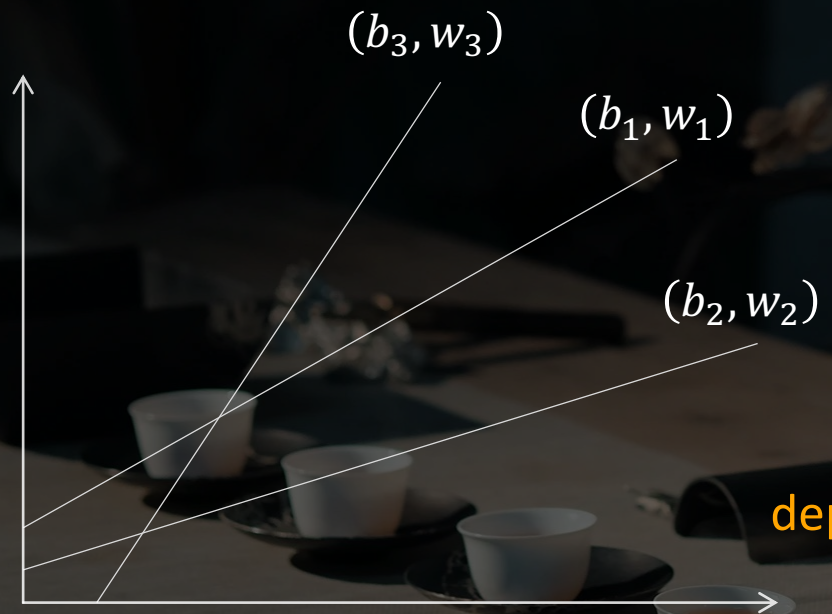
# Green tea / Oolong tea

4. Model Training

The training data is used to incrementally improve the model's performance, i.e. the ability to tell whether a given cup of tea is oolong tea or green tea.

# Green tea / Oolong tea

## 4. Model Training




determine the position of the line

↑

parameters

$$\underbrace{y}_{\text{output}} = \underbrace{b}_{\text{bias}} + \underbrace{w}_{\text{weight}} \cdot \underbrace{x}_{\text{input}}$$

dependent variable (label)

Independent variable (feature)

# Green tea / Oolong tea

## 4. Model Training

- Training aims to find a specific pair of $b$ and $w$ values so that the line can be positioned well to be able to divide tea examples into 2 groups – oolong & green.
- The training process goes iteratively – in each iteration, $b$ and $w$ values are updated; the performance is calculated and compared with the performance in the last iteration in order to determine how to update $b$ and $w$ values for the next iteration.
- This process stops when new $b$ and $w$ values do not improve the model's performance, or the change is very little.

# Green tea / Oolong tea

4. Model Training

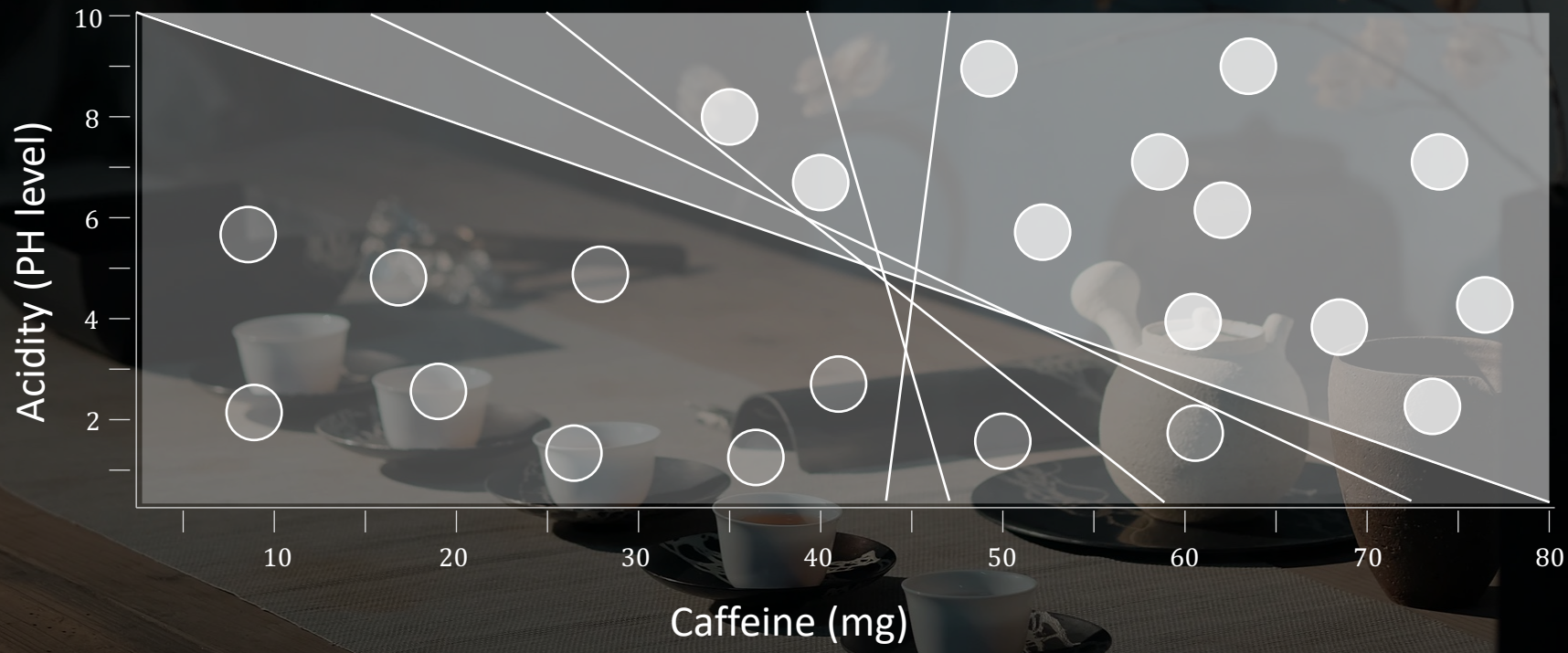
   First iteration: a random pair of $b$ and $w$ values is used to attempt to predict the output with those values.

   It may perform very poorly ☹

   It will get better over iterations 😅

# Green tea / Oolong tea

## 4. Model Training



# Green tea / Oolong tea

5. Model Testing

Once the training process has stopped, the next step is to evaluate the model, using the test set.

# Green tea / Oolong tea

5. Model Testing

Aims to see if the trained model is any good when performing a specified task on previously unseen data, i.e. data from the test set.

"accuracy"

In the tea example, the aim is to test "how well" the trained model can separate oolong tea and green tea.

# Green tea / Oolong tea

6. Hyperparameter Tuning

Aims to see if there is any way to improve the training process thus further improving the model.

$$y = b + w \cdot x$$

parameter (of the model) vs **hyper**parameter (of the training process)

e.g. Learning Rate

# Green tea / Oolong tea

6. Hyperparameter Tuning

Learning Rate: how big the change to make in parameter values.

↓

- how accurate a model can become
- how long the training process takes

- Hyperparameter tuning is an experimental procedure
- Sometimes a bit more of an art than science or engineering

# Green tea / Oolong tea

7. Inference / prediction

A cup of mystery tea → model → Oolong tea

# What is Machine Learning Workflow?

# Machine Learning Workflow


Problem Framing

Data Preparation

Algorithm Selection

Model Training

Model Testing

Hyperparameter Tuning

Inference / Prediction

# Some key Machine Learning Terms

| | | |
|---|---|---|
| label | transforming label | training / learning |
| feature | example | training set |
| parameter | hyperparameter | test set |

# 2. Coding Environment

# Coding Environment


An interpreted & dynamic programming language

## Check version

python --version 3.10 2.x?

## Install/update

Standard way https://www.python.org/downloads/

Miniconda https://docs.conda.io/en/latest/miniconda.html


(minimal installer for conda, which includes only conda, Python, the packages they depend on, and a small number of other useful packages including pip, zlib and a few others)

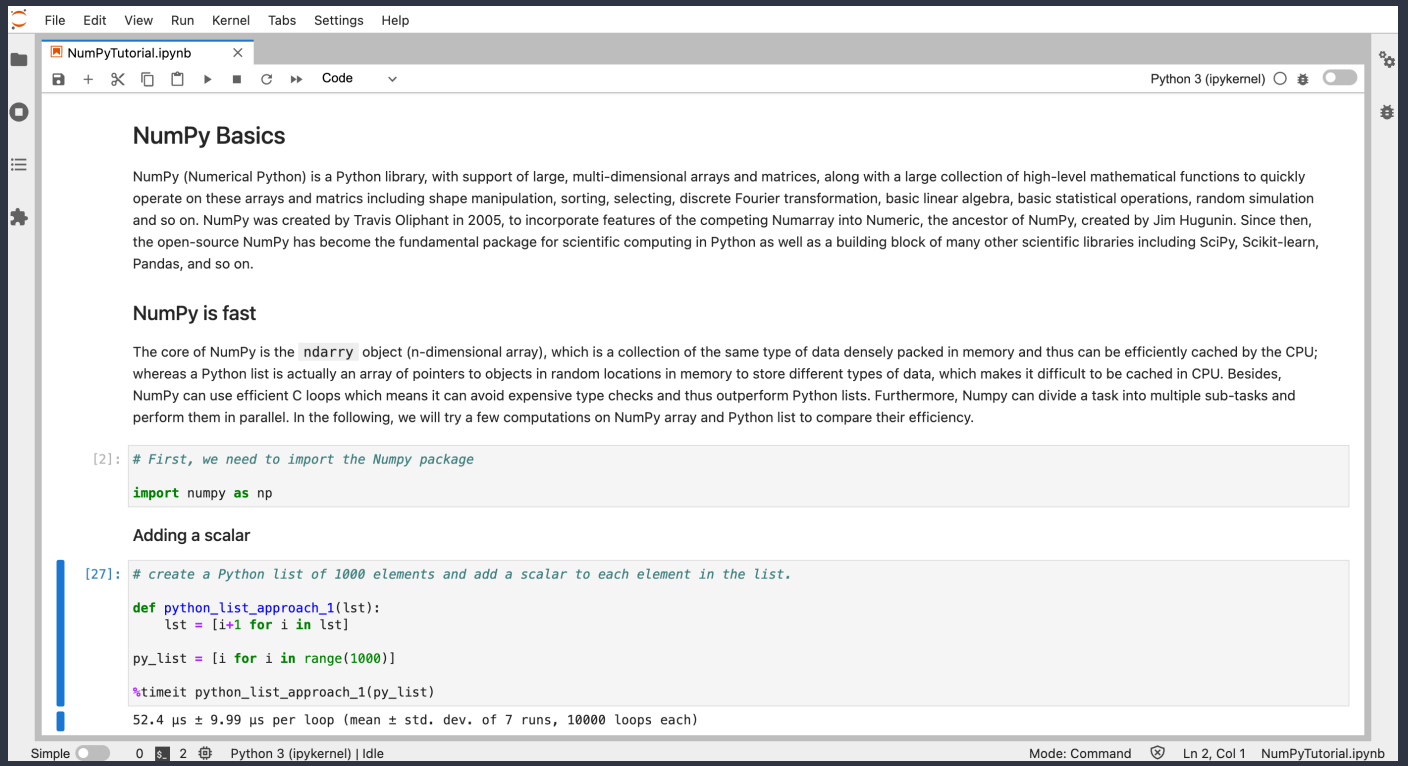
# Coding Environment


Jupyter Lab

install conda install -c conda-forge jupyterlab

https://jupyter.org/install

run jupyter lab



# Coding Environment


conda install -c conda-forge numpy

conda install -c conda-forge matplotlib

conda install -c conda-forge pandas

conda install -c conda-forge scipy

conda install -c conda-forge seaborn

conda install -c conda-forge scikit-learn

conda update numpy

conda update matplotlib

conda update pandas

conda update scipy

conda update seaborn

conda update scikit-learn

# Coding Environment

## Introduction to scikit-learn

### Data representation (tabular) in scikit-learn

For a supervised learning task, the tabular dataset contains X (independent variables, features) and y (dependent variable, target); while for an unsupervised learning task, the tabular dataset contains X only.

# Coding Environment

## Introduction to 


### Toy Datasets

https://scikit-learn.org/stable/datasets/toy_dataset.html

scikit-learn comes with a few small standard datasets that do not require to download any file from some external website.

They can be loaded using the following functions:

load_iris(*[, return_X_y, as_frame])

load_diabetes(*[, return_X_y, as_frame])

load_digits(*[, n_class, return_X_y, as_frame])

load_linnerud(*[, return_X_y, as_frame])

load_wine(*[, return_X_y, as_frame])

load_breast_cancer(*[, return_X_y, as_frame])

# Coding Environment

## Introduction to 


### Toy Datasets

#### The Diabetes dataset

sklearn.datasets.load_diabetes(*[, return_X_y, as_frame])

Loads and returns the diabetes dataset:

"age, sex, body mass index, average blood pressure, and six blood serum measurements were obtained for each of n = 442 diabetes patients, as well as the response of interest, a quantitative measure of disease progression one year after baseline."

11 columns (10 features, 1 target), 442 examples

Original dataset https://www4.stat.ncsu.edu/~boos/var.select/diabetes.html

# Coding Environment

## Introduction to scikit learn

https://scikit-learn.org/stable/

> - Simple and efficient tools for predictive data analysis
> - Accessible to everybody, and reusable in various contexts
> - Built on NumPy, SciPy, and matplotlib
> - Open source, commercially usable - BSD license

Scikit-learn focuses on modelling data.

For loading, manipulating and summarising data, use NumPy & Pandas

# Coding Environment

## Introduction to scikit learn

### Toy Datasets

#### The Diabetes dataset

##### Load the dataset

```
from sklearn.datasets import load_diabetes
dataset = load_diabetes()
X, y = dataset.data, dataset.target
print(X)
```

```
[[ 0.03807591 0.05068012 0.06169621 ... -0.00259226 0.01990842 -
0.01764613] [-0.00188202 -0.04464164 -0.05147406 ... -0.03949338 -
0.06832974 -0.09220405] [ 0.08529891 0.05068012 0.04445121 ... -
0.00259226 0.00286377 -0.02593034] ...
```

# Coding Environment

## Introduction to scikit-learn

### Load tabular datasets from csv files

```
import pandas as pd
df = pd.read_csv('data/diabetes.csv')
print(df.head(10))
```

| | AGE | SEX | BMI | BP | S1 | S2 | S3 | S4 | S5 | S6 | Y |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 59 | 2 | 32.1 | 101 | 157 | 93.2 | 38 | 4 | 4.8598 | 87 | 151 |
| 2 | 48 | 1 | 21.6 | 87 | 183 | 103.2 | 70 | 3 | 3.8918 | 69 | 75 |
| 3 | 72 | 2 | 30.5 | 93 | 156 | 93.6 | 41 | 4 | 4.6728 | 85 | 141 |
| 4 | 24 | 1 | 25.3 | 84 | 198 | 131.4 | 40 | 5 | 4.8903 | 89 | 206 |
| 5 | 50 | 1 | 23 | 101 | 192 | 125.4 | 52 | 4 | 4.2905 | 80 | 135 |
| 6 | 23 | 1 | 22.6 | 89 | 139 | 64.8 | 61 | 2 | 4.1897 | 68 | 97 |
| 7 | 36 | 2 | 22 | 90 | 160 | 99.6 | 50 | 3 | 3.9512 | 82 | 138 |
| 8 | 66 | 2 | 26.2 | 114 | 255 | 185 | 56 | 4.55 | 4.2485 | 92 | 63 |
| 9 | 60 | 2 | 32.1 | 83 | 179 | 119.4 | 42 | 4 | 4.4773 | 94 | 110 |

# Coding Environment

## Introduction to scikit learn

### Exploratory Data Analysis (EDA)

```
type(df)
```

pandas.core.frame.DataFrame

```
df.shape
```

(442, 11)

```
df.tail()
```

| | AGE | SEX | BMI | BP | S1 | S2 | S3 | S4 | S5 | S6 | Y |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 437 | 60 | 2 | 28.2 | 112.00 | 185 | 113.8 | 42.0 | 4.00 | 4.9836 | 93 | 178 |
| 438 | 47 | 2 | 24.9 | 75.00 | 225 | 166.0 | 42.0 | 5.00 | 4.4427 | 102 | 104 |
| 439 | 60 | 2 | 24.9 | 99.67 | 162 | 106.6 | 43.0 | 3.77 | 4.1271 | 95 | 132 |
| 440 | 36 | 1 | 30.0 | 95.00 | 201 | 125.2 | 42.0 | 4.79 | 5.1299 | 85 | 220 |
| 441 | 36 | 1 | 19.6 | 71.00 | 250 | 133.2 | 97.0 | 3.00 | 4.5951 | 92 | 57 |

# Coding Environment

## Introduction to scikit-learn

### Exploratory Data Analysis (EDA)

```
df.describe( )
```

| | AGE | SEX | BMI | BP | S1 | S2 | S3 | S4 | S5 | S6 | Y |
|---|---|---|---|---|---|---|---|---|---|---|---|
| count | 442 | 442 | 442 | 442 | 442 | 442 | 442 | 442 | 442 | 442 | 442 |
| mean | 48.5181 | 1.468326 | 26.37579 | 94.64701 | 189.1403 | 115.4391 | 49.78846 | 4.070249 | 4.641411 | 91.26018 | 152.1335 |
| std | 13.10903 | 0.499561 | 4.418122 | 13.83128 | 34.60805 | 30.41308 | 12.9342 | 1.29045 | 0.522391 | 11.49634 | 77.09301 |
| min | 19 | 1 | 18 | 62 | 97 | 41.6 | 22 | 2 | 3.2581 | 58 | 25 |
| 25% | 38.25 | 1 | 23.2 | 84 | 164.25 | 96.05 | 40.25 | 3 | 4.2767 | 83.25 | 87 |
| 50% | 50 | 1 | 25.7 | 93 | 186 | 113 | 48 | 4 | 4.62005 | 91 | 140.5 |
| 75% | 59 | 2 | 29.275 | 105 | 209.75 | 134.5 | 57.75 | 5 | 4.9972 | 98 | 211.5 |
| max | 79 | 2 | 42.2 | 133 | 301 | 242.4 | 99 | 9.09 | 6.107 | 124 | 346 |

# Coding Environment

## Introduction to scikit-learn

### Exploratory Data Analysis (EDA)

```
df.info( )
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 442 entries, 0 to 441
Data columns (total 11 columns):
#  Column Non-Null Count Dtype
--- --------  ------------------- --------
0   AGE     442 non-null      int64
1   SEX     442 non-null      int64
…           …                 …
8   S5      442 non-null      float64
9   S6      442 non-null      int64
10 Y        442 non-null      int64
dtypes: float64(6), int64(5)
memory usage: 38.1 KB
```

```
df.isnull( ).sum( )
```

```
AGE     0
SEX     0
BMI     0
BP      0
S1      0
S2      0
S3      0
S4      0
S5      0
S6      0
Y       0
dtype: int64
```

# Coding Environment

## Introduction to scikit learn

Split data into features X and target y.

# Coding Environment

## Introduction to scikit learn

Split data into features $X$ and target $y$.

```
y = df['Y'].values
y
```

```
array([151, 75, 141, 206, 135, 97, 138, 63, 110, 310, 101, 69, 179, 185, 118, 171, 166,
       144, 97, 168, 68, 49, 68, 245, 184, 202, 137, 85, 131, 283, 129, 59, 341, 87, 65,
       102, 265, 276, 252, 90, 100, 55, 61, 92, 259, 53, 190, 142, 75, 142, 155, 225, 59,
       104, 182, 128, 52, 37, 170, 170, 61, 144, 52, 128, 71, 163, 150, 97, 160, 178, 48,
       270, 202, 111, 85, 42, 170, 200, 252, 113, 143, 51, 52, 210, 65, 141, 55, 134, 42,
       111, 98, 164, 48, 96, 90, 162, 150, 279, 92, 83, 128, 102, 302, 198, 95, 53, 134,
       144, 232, 81, 104, 59, 246, 297, 258, 229, 275, 281, 179, 200, 200, 173, 180, 84,
       121, 161, 99, 109, 115, 268, 274, 158, 107, 83, 103, 272, 85, 280, 336, 281, 118,
       317, 235, 60, 174, 259, 178, 128, 96, 126, 288, 88, 292, 71, 197, 186, 25, 84, 96,
       195, 53, 217, 172, 131, 214, 59, 70, 220, 268, 152, 47, 74, 295, 101, 151, 127, ...
```

# Coding Environment

## Introduction to scikit learn

Split data into features X and target y.

```
X = df.iloc[:, :-1].values
X
```

```
array([[59.  , 2.  , 32.1  , ... , 4.  , 4.8598  , 87.  ],
       [48.  , 1.  , 21.6  , ...,  3.  , 3.8918.  , 69.  ],
       [72.  , 2.  , 30.5  , ...,  4.  , 4.6728.  , 85.  ],
       ...,
       [60.  , 2.  , 24.9  , ...,  3.77  , 4.1271.  , 95.  ],
       [36.  , 1.  , 30.  , ...,  4.79  , 5.1299.  , 85.  ],
       [36.  , 1.  , 19.6  , ...,  3.  , 4.5951.  , 92.  ]])
```

# Coding Environment

## Introduction to scikit learn

Split data into features X and target y.

```
y.shape
```

(442,)

```
X.shape
```

(442, 10)

# Coding Environment

## Introduction to scikit-learn

### Permute/shuffle data

```
import numpy as np

indices = np.arange(df.shape[0])
rng = np.random.RandomState(0)
permuted_indices = rng.permutation(indices)
permuted_indices
```

442

```
df.shape
```

(442, 11)

array([362, 249, 271, 435, 400, 403, 12, 399, 198, 205, 78, 144, 298, 171, 268, 21, 194, 1, 10, 208, 76, 37, 388, 411, 289, 100, 261, 401, 319, 206, 375, 283, 373, 344, 287, 360, 179, 238, 434, 302, 158, 118, 339, 397, 330, 296, 164, 54, 157, 71, 284, 6, 343, 326, 159, 15, 320, 60, 282, 142, 56, 124, 107, 132, 382, 102, 160, 141, 186, 441, 90, 59, 347, 233, 427, 65, 438, 155, 122, 276, 188, 96, 170, 113, 381, 213, 134, 49, 52, 74, 26, 45, 389, 154, 4, 386, 225, 327, 264, 8, 190, 5, 371, 200,...

# Coding Environment

## Introduction to scikit learn

### Split data into Training Set and Test Set

```
train_size = round(0.8*df.shape[0])
test_size = df.shape[0]) - train_size
print(train_size, test_size)
```

354 88

0:354

```
train_ind = permuted_indices[:train_size]
test_ind = permuted_indices[train_size:]
```

354:442

```
X_train, X_test = X[test_ind], X[train_ind]
y_train, y_test = y[test_ind], y[train_ind]
```

# Coding Environment

## Introduction to scikit-learn

### Estimator API

Estimator:

Classes for supervised learning

- Regressors
- Classifiers

fit = train

estimator = model

```
class SupervisedEstimator(...):
    def _init_(self, hyperparameter_1, ...):
        self.hyperparameter_1
        ...
    def fit(self, X, y):
        ...
        self.fit_attribute_
        return self
    def predict(self, X):
        ...
        return y_predicted
    def score(self, X, y):
        ...
        return score
    def _private_method(self):
        ...
    ...
```

```
...
# Create linear regression object
regr = linear_model.LinearRegression()

# Train the model using the training sets
regr.fit(diabetes_X_train, diabetes_y_train)

# Make predictions using the testing set
diabetes_y_pred = regr.predict(diabetes_X_test)

# The coefficients
print('Coefficients: \n', regr.coef_)
# The mean squared error
print('Mean squared error: %.2f' % mean_squared_error(diabetes_y_test, diabetes_y_pred))
# The coefficient of determination: 1 is perfect prediction
print('Coefficient of determination: %.2f' % r2_score(diabetes_y_test, diabetes_y_pred))
...
```

```
Coefficients:
 [938.23786125]
Mean squared error: 2548.07
Coefficient of determination: 0.47
```

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

# Summary

# Summary

## Machine Learning Workflow


Coding Environment - Jupyter Lab, NumPy, matplotlib, pandas, SciPy, seaborn, scikit-learn

## Next Lecture

Data Preparation

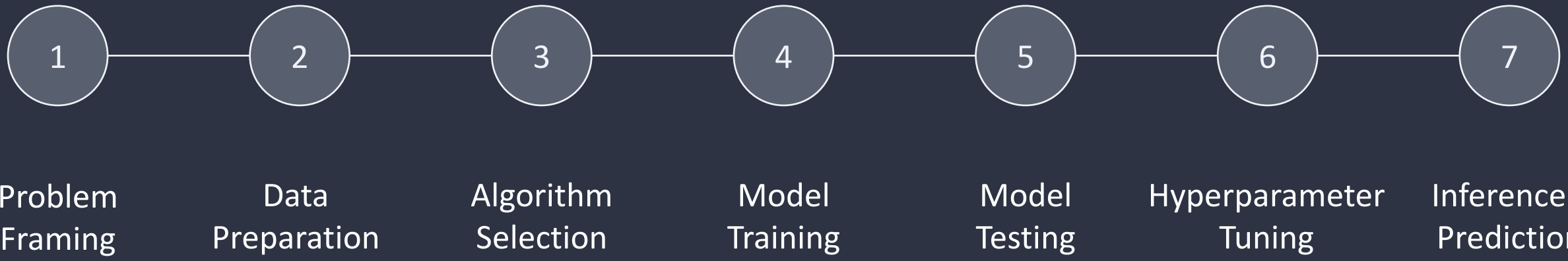
COMP2261 ARTIFICIAL INTELLIGENCE / MACHINE LEARNING

# Data Construction

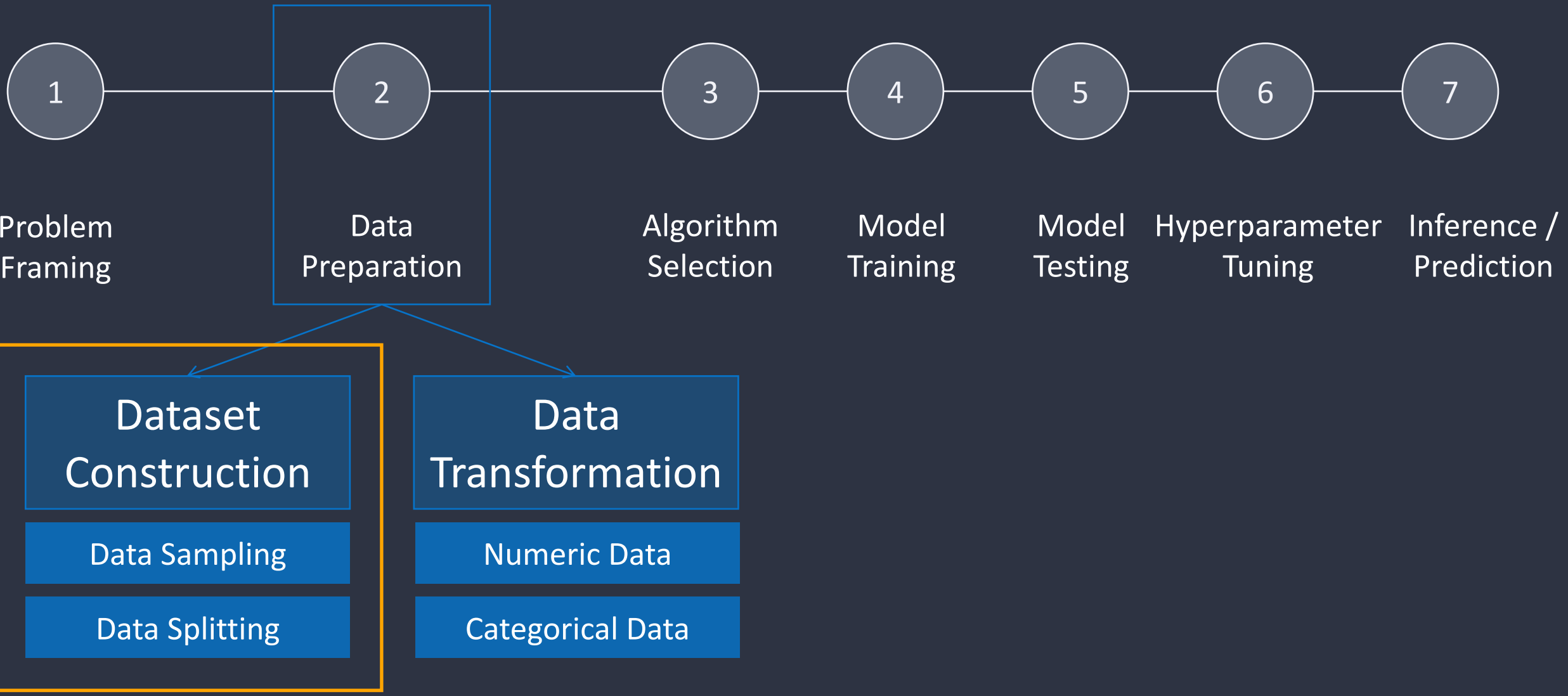
Dr Yang Long

Durham University

# Last Lecture



# Last Lecture



# Lecture Overview

1. What's Data
2. Data Sampling
3. Data Splitting
4. Data Splitting Procedure in scikit-learn

# 1. What’s Data

# EXAMPLE. to build a happiness predictor

input X | output y

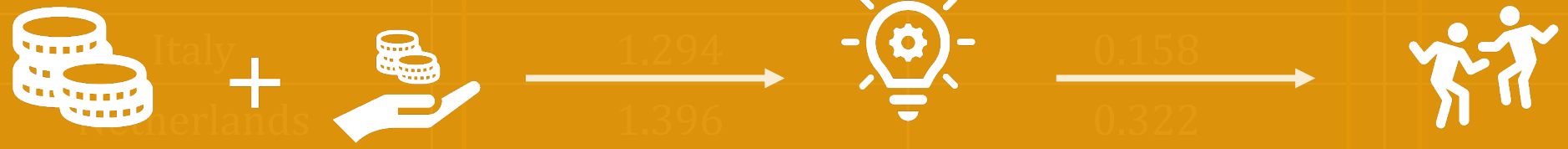
| Country | GDP/capita | Happiness Score |
|---|---|---|
| Austria | 1.376 | 7.246 |
| Finland | 1.340 | 7.769 |
| France | 1.324 | 6.592 |
| Germany | 1.373 | 6.985 |
| Greece | 1.181 | 5.287 |
| Netherlands | 1.396 | 7.488 |
| Poland | 1.206 | 6.182 |
| Spain | 1.286 | 6.354 |
| Sweden | 1.387 | 7.343 |
| United Kingdom | 1.333 | 7.054 |


Source: https://www.kaggle.com/unsdsn/world-happiness

EXAMPLE. to build a happiness predictor

input X output y

| Country | GDP/capita | Generosity | Happiness Score |
|---|---|---|---|
| Austria | 1.376 | 0.244 | 7.246 |
| Finland | 1.340 | 0.153 | 7.769 |
| France | 1.324 | 0.111 | 6.592 |
| Germany | 1.373 | 0.261 | 6.985 |
| Greece | 1.181 | 0.000 | 5.287 |
| Italy | 1.294 | 0.158 | 6.223 |
| Netherlands | 1.396 | 0.322 | 7.488 |
| Portugal | 1.221 | 0.047 | 5.961 |
| Poland | 1.206 | 0.117 | 6.182 |
| Spain | 1.286 | 0.153 | 6.354 |
| Sweden | 1.387 | 0.267 | 7.343 |
| United Kingdom | 1.333 | 0.288 | 7.054 |




We can decide what to be considered as input X and output y, as well as the definitions of X and y, to make it helpful in a specific task or use case.

Source: https://www.kaggle.com/unsdsn/world-happiness

EXAMPLE. to build a happiness predictor

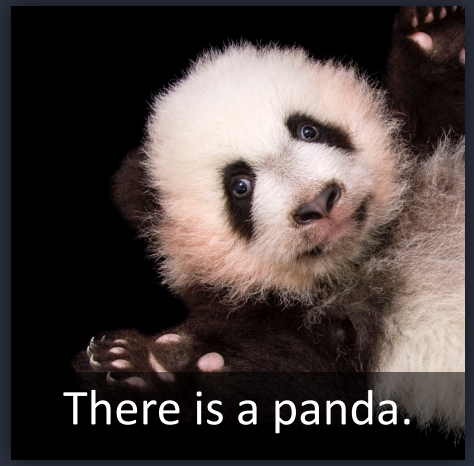
output y

input X

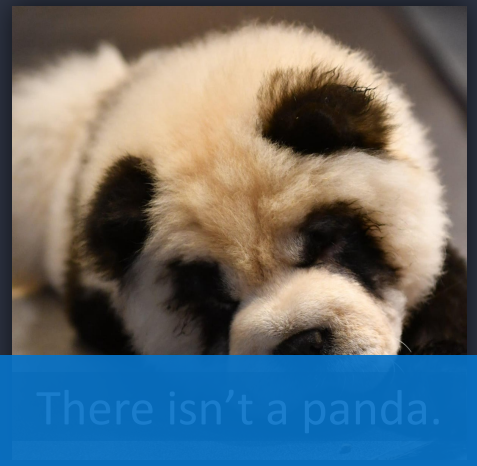
| Country | GDP/capita | Generosity | Happiness Score |
|---|---|---|---|
| Austria | 1.376 | 0.244 | 7.246 |
| Finland | 1.340 | 0.153 | 7.769 |
| France | 1.324 | 0.111 | 6.592 |
| Germany | 1.373 | 0.261 | 6.985 |
| Greece | 1.181 | 0.000 | 5.287 |
| Italy | 1.294 | 0.158 | 6.223 |
| Netherlands | 1.396 | 0.322 | 7.488 |
| Portugal | 1.221 | 0.047 | 5.603 |
| Poland | 1.206 | 0.117 | 6.182 |
| Spain | 1.286 | 0.153 | 6.354 |
| Sweden | 1.387 | 0.267 | 7.343 |
| United Kingdom | 1.333 | 0.348 | 7.054 |

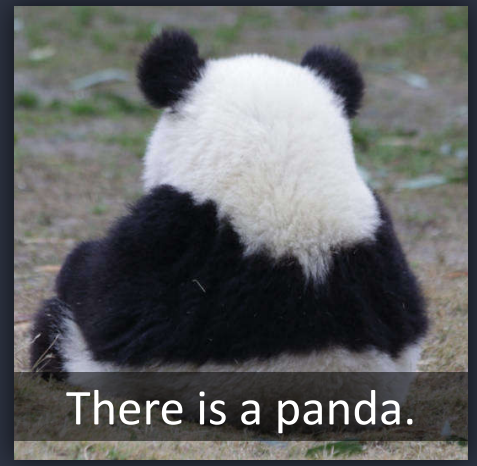

Source: https://www.kaggle.com/unsdsn/world-happiness

# EXAMPLE. to build a panda detector

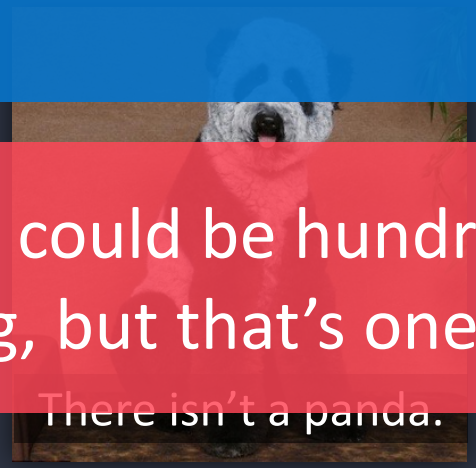


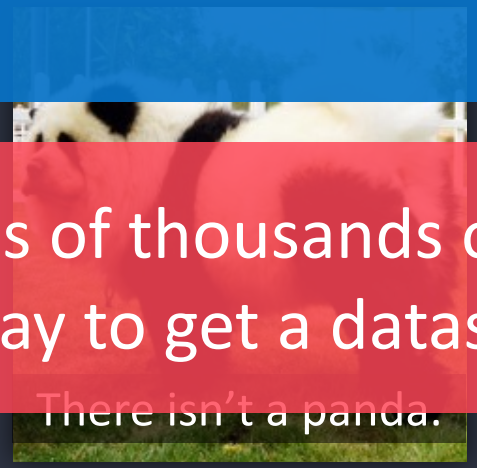





## Manual Labelling



In a dataset there could be hundreds of thousands of photos to label.
Time consuming, but that's one way to get a dataset with X and y.

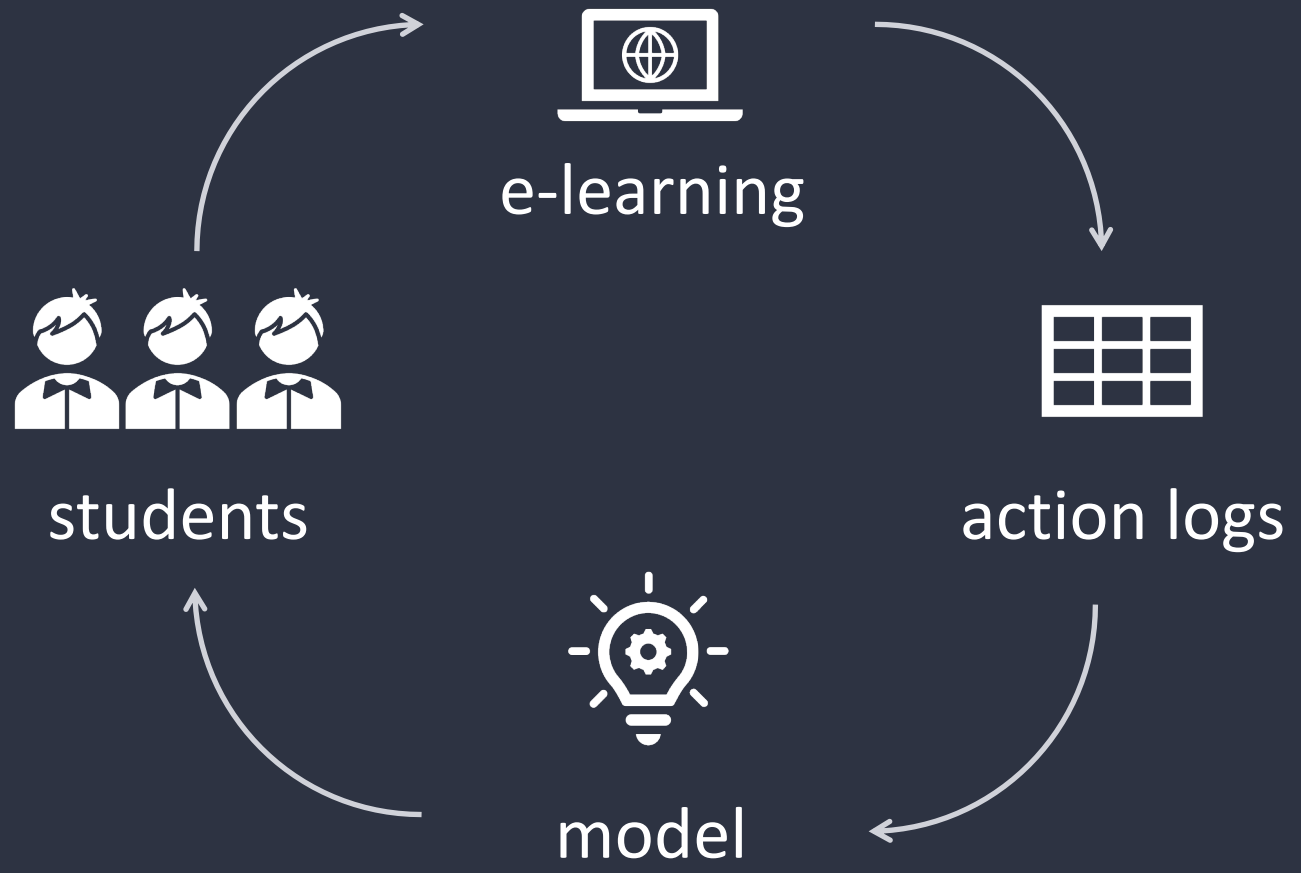
Manual Labelling

Behaviour Tracking

EXAMPLE. to build a learning material recommender

Behaviour Tracking

Instead of labelling data manually, students can generate data for us.




To predict a webpage a student may visit.
To recommend a webpage a student should visit.

# Public datasets

https://www.kaggle.com/datasets

https://archive.ics.uci.edu/ml/datasets.php

https://en.wikipedia.org/wiki/List_of_datasets_for_machine-learning_research

https://ukdataservice.ac.uk

https://data.worldbank.org

https://paperswithcode.com/datasets

https://data.europa.eu

https://datasetsearch.research.google.com

https://www.data.gov

https://guides.library.cmu.edu/machine-learning/datasets

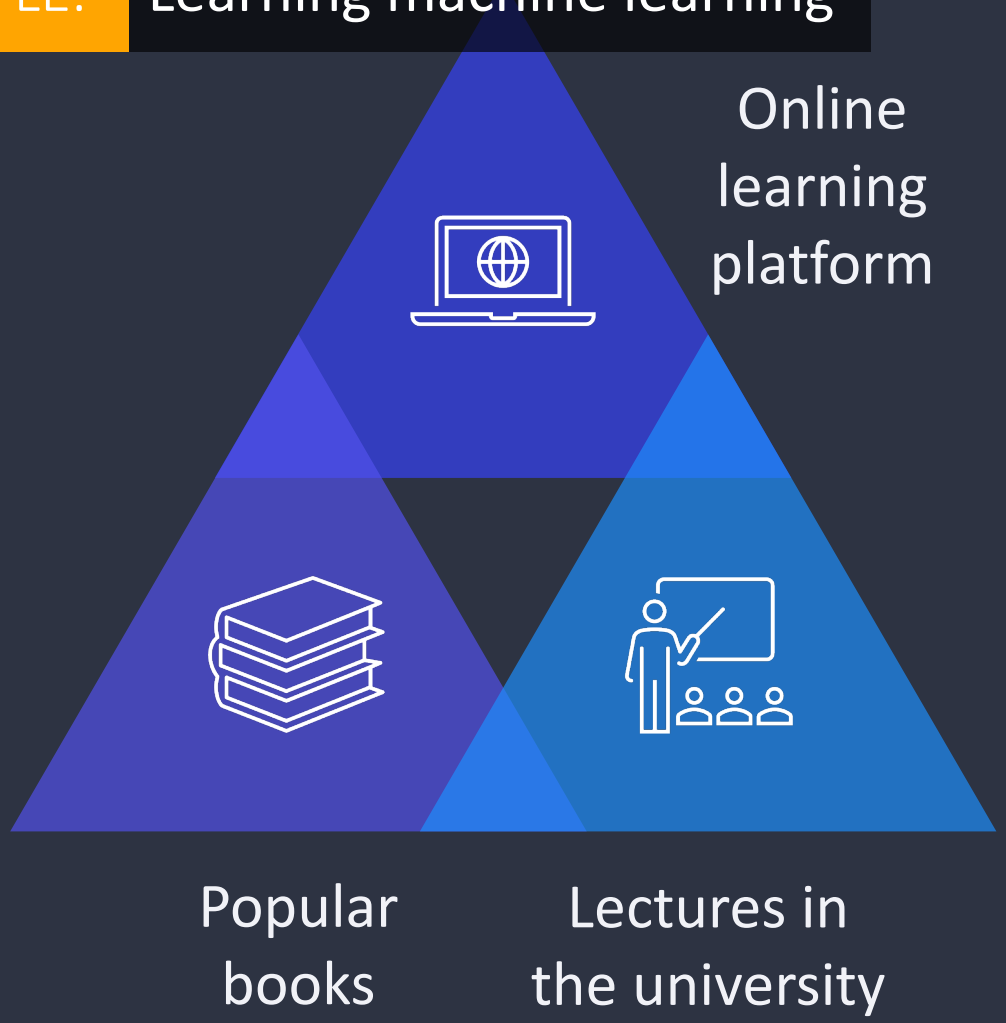
https://www.openml.org

…

# 2. Data Sampling

# Data Sampling

EXAMPLE. Learning machine learning




The best way is to pick up only one option, rather than being lost in the "information overload" or spending too much time just to find the "best option" which possibly doesn't exists.

-- A "good enough" option is perhaps our "best option".

# Data Sampling

In Machine Learning (with limited resources)

We want the minimum amount of data containing sufficient information required to learn properly from the phenomenon without wasting time.

One way is to reduce information redundancy that doesn’t give us useful information or doesn’t contain any business value for a particular task.

# Data Sampling

How can we ensure our sample is not redundant or incomplete,

i.e. our sample represents the real word phenomenon?

# Data Sampling

How can we ensure our sample is not redundant or incomplete,

i.e. our sample represents the real word phenomenon?

- High-level comparison between our sample and other sample or the whole population, to check if the chosen sample represents the whole population.
- Instead of learning from a large dataset, we could just make a sub-sampling of it yet keeping all the statistics intact.

# Data Sampling

Key to sampling, i.e. picking up a small, easy to handle sample…

Ensure the chosen sample does not lose statistical significance with respect to the whole population.

| | |
|---|---|
| Sample size too small | not carry enough information to learn from. |

We must compromise between the size & the information.

| | |
|---|---|
| Sample size too large | time-consuming to proceed. |

# Data Sampling

## Statistical framework

Sample to keep probability distribution of the population under reasonable significant level.

# Data Sampling

## Statistical framework

VS

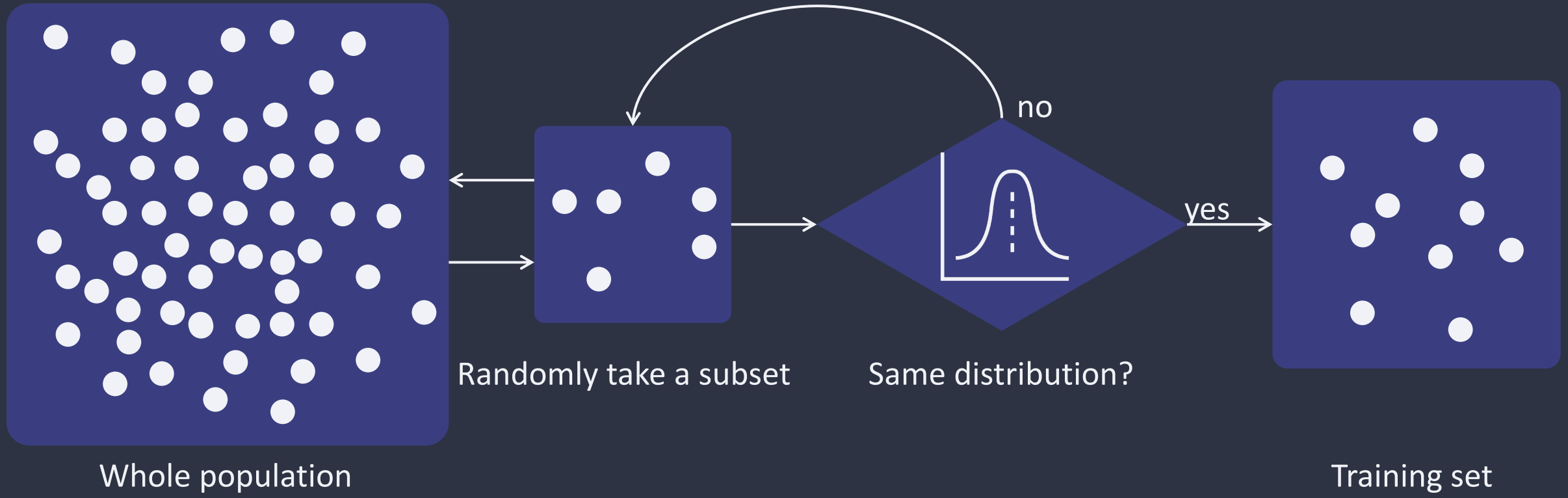
sample

whole population

Have a look at the histogram of the sample and ensure it to be the same as the histogram of the whole population.

# Data Sampling

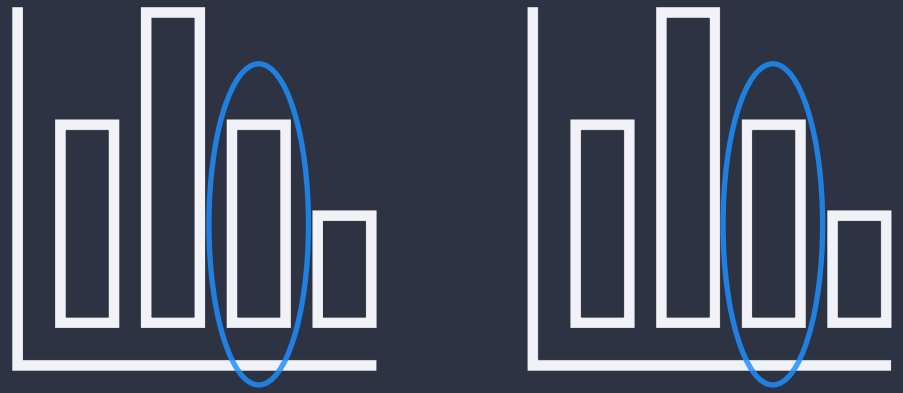
## Statistical framework



# Data Sampling

## Statistical framework

1. What if our dataset is multivariate, i.e. having N variables (N > 1), especially that they could be a mix of numerical and categorical variables?



Consider each variable independently. If each one of single univariate histograms of the sample's columns is comparable with the correspondent histogram of the whole population's columns, we can assume the chosen sample is not biased.

# Data Sampling

## Statistical framework

2. How to compare a sample with the whole population?

| | |
|---|---|
| Categorical Variables | Pearson's chi-square test |
| Numerical Variables | Kolmogorov-Smirnov test |

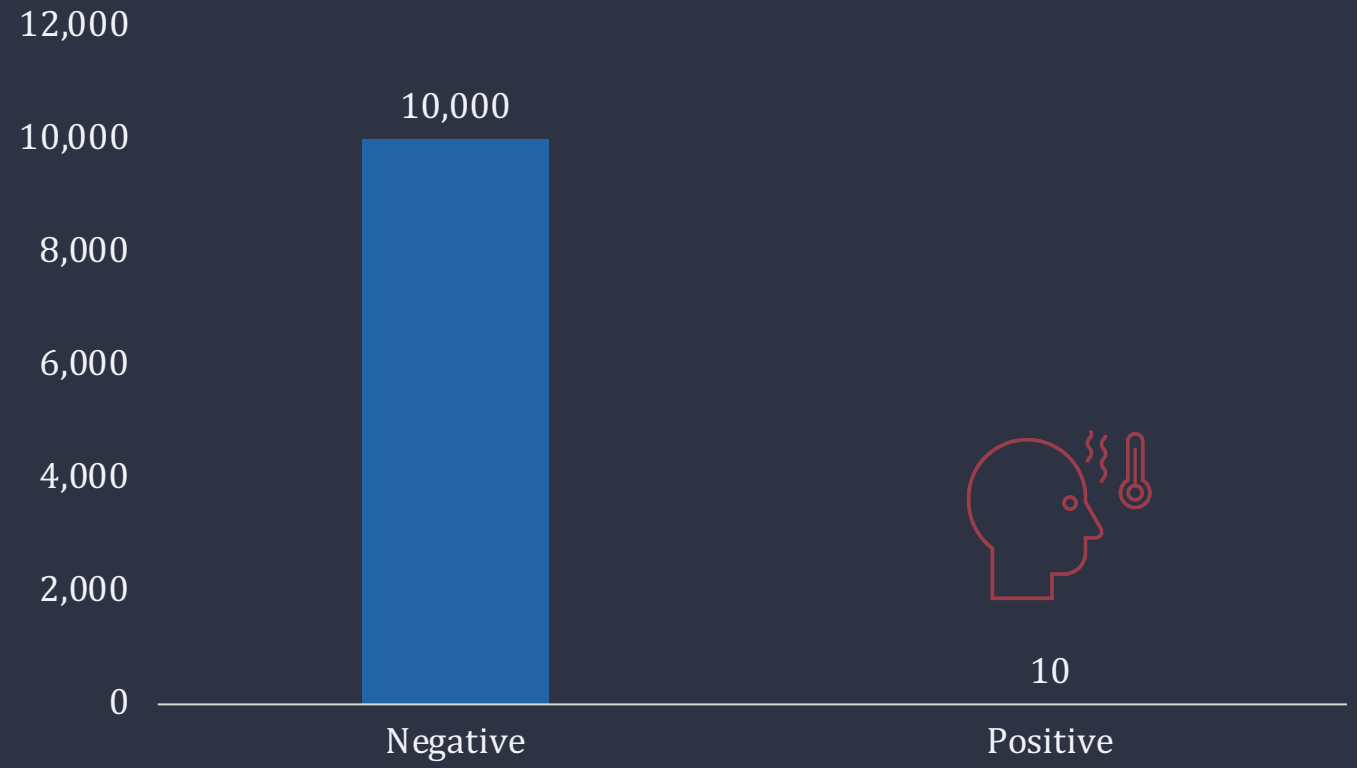
Test a *null* hypothesis – the frequency distribution of certain instances in a sample is consistent with the whole population of the dataset.

In multivariate case, all the variables need to be significative, and we reject the *null* hypothesis if the p-value of at least one of the tests is lower than the usual 5% confidence level, i.e. to accept the sample, we must have all the variables pass the significance test.

# Data Sampling

## Imbalanced Data

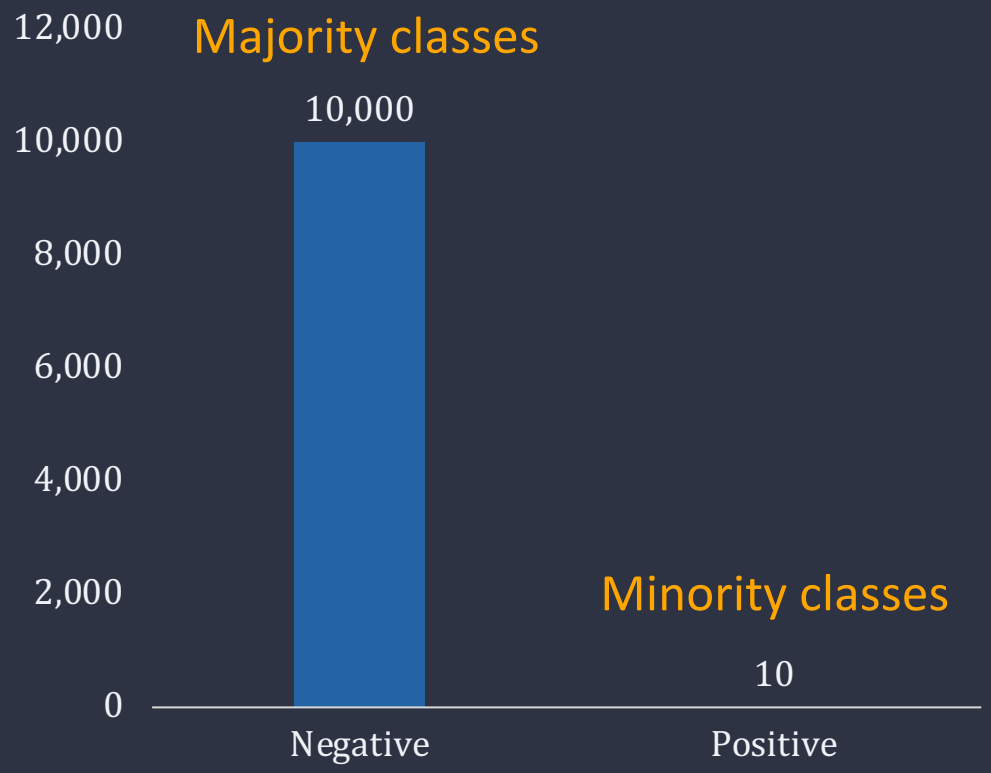
EXAMPLE. to classify positive and negative COVID-19 cases



# Data Sampling

## Imbalanced Data

EXAMPLE. to classify positive and negative COVID-19 cases 




Much more negative examples than the positive ones

The classification dataset is with skewed class proportions – imbalanced.

Imbalanced dataset could be very problematic!

# Data Sampling

## Imbalanced Data

Imbalanced dataset could be very problematic!

Not learnt enough from the minority class

- low efficiency in detecting them
- but, often the minority class is of the most interests

Misleading high accuracy

- excellent accuracy does not necessarily mean the model performs very well
- it may be just reflecting the underlying class distribution

# Data Sampling

## Imbalanced Data

Imbalanced dataset could be very problematic!

Most machine learning algorithms for classification were designed under the assumption that there were equal numbers of examples for each class.

However, it is common that our dataset does not have exactly equal number of examples in each class.

e.g. fraud-detection, spam detection, outlier detection, anomaly detection…
claim prediction, default prediction, conversation prediction, churn prediction…

# Data Sampling

## Imbalanced Data

There isn't a clear definition or division, we could roughly have the following categories:

| Degree of imbalance | Minority % |
|---|---|
| mild | 20% ~ 40% |
| moderate | 1% ~ 20% |
| extreme | < 1% |

# Data Sampling

## Imbalanced Data

Techniques to combat imbalanced dataset:

As a start: try training on the true distribution, e.g. using the whole dataset

If the trained model works well and generalises well – well done!

If not…

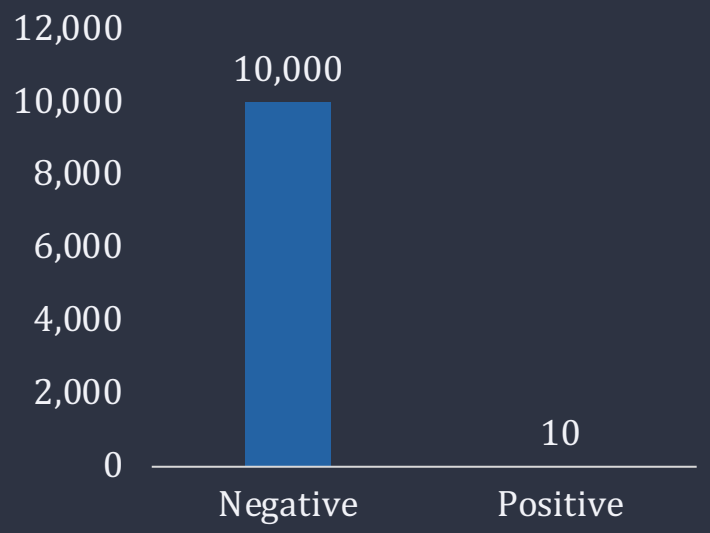
| | |
|---|---|
| Down-sampling | training model on a disproportionately low subset of the majority class examples. |
| Up-weighting | adding an example weight to the down-sampled class equal to the factor by which we down-sampled. |

# Data Sampling

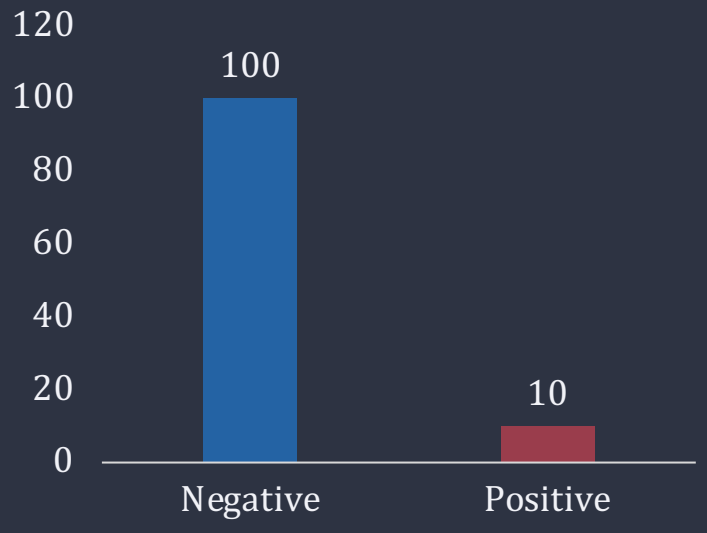
## Imbalanced Data

| | |
|---|---|
| 1 | Down-sampling |

down-sample the majority class, the positive cases in our covid-19 dataset. Since we have 10 positives to 10,000 negatives, we can down-sample by a factor of 100, thus taking one out of a hundred negatives. So now, we have 1:10 instead of 1:1,000, and our sample is less biased, i.e. about 10/(100+10), 9%, instead of 0.1% of our data is positive.




Extremely biased



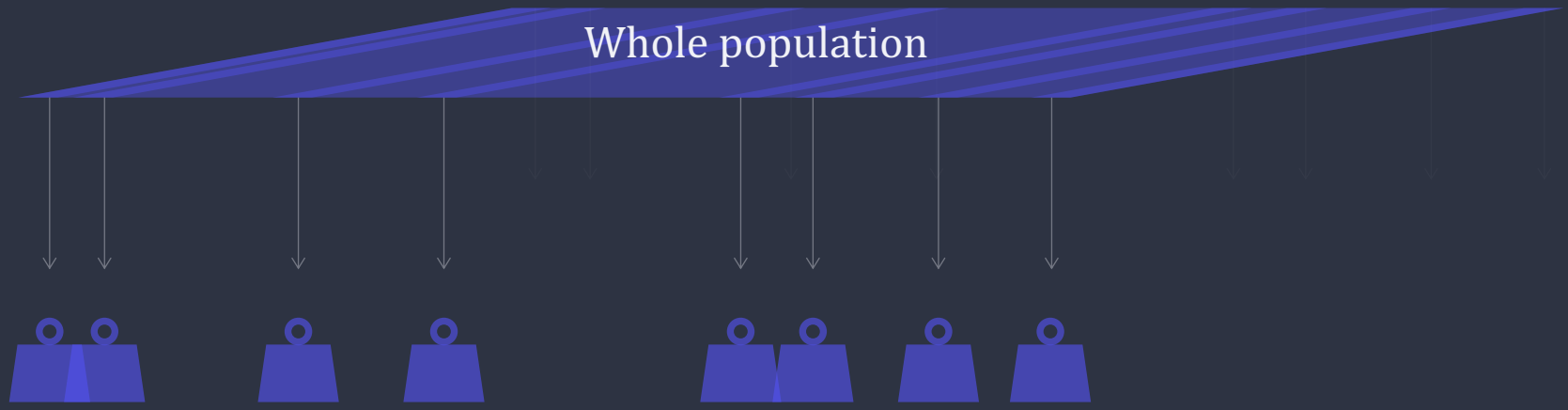

Less biased

# Data Sampling

## Imbalanced Data

| 2 | Up-weighting |
|---|---|

up-weight the down-sampled majority class. As our down-sample factor is 100, the example weight should also be 100. This way we count an individual negative example more importantly during training. A weight of 100 means the model treats those negative examples 100 times as important as it would a positive example with a weight of 1.




Down-sampling

Randomly pick example from the whole population

Up-weighting

Add weight to the down-sampled example

Weight should be equal to the factor used for down-sampling:

$$\{\text{example weight}\} = \{\text{original example weight}\} \times \{\text{down-sampling factor}\}$$

# 3. Data Splitting

EXAMPLE. to build a tea detection application

# Green tea / Oolong tea

Training set Test set

Why do we need to split the data set?

EXAMPLE. to build a tea detection application

# Green tea / Oolong tea

Randomise examples

Why do we need to randomise examples?

# Data Splitting

How large should we make different splits?

## The larger Training Set

the better model we will be able to learn.

## The larger Test Set

the better we will be able to have confidence in evaluation metrics, and tighter confidence intervals.

# Data Splitting

For now… make sure our Test Set meets the following 2 conditions:

- large enough to yield statistically meaningful results.
- representative of the dataset as a whole. In other words, don't pick a Test Set with different characteristics than the Training Set.

# Data Splitting

## A Solution to Overfitting

# Data Splitting

## A Solution to Overfitting

To summarise:

1. Put the Test Set aside and completely unused.
2. Pick the model that works best on the Validation Set.
3. Double-check that model against the Test Set.

This is a better approach because it creates fewer exposures to the Test Set.

# Data Splitting

## A Solution to Overfitting

Why need Validation Set AND Test Set both to evaluate model?

# Data Splitting

## A Solution to Overfitting

### Why need Validation Set AND Test Set both to evaluate model?

| Validation Set | Test Set |
| --- | --- |
| Compare hyperparameter combinations<br>• We want to train a model whose performance depends on a set of hyperparameters e.g. learning rate.<br>• Validation Set is used to evaluate model performance for different combinations of hyperparameter values. | Compare different models<br>• We want to compare trained models in an unbiased way, by comparing model performance using unseen data.<br>• Test Set is kept apart from the training process, thus being the unseen data, for comparing different trained models. |

# 4. Data Splitting Procedure in scikit-learn

# Data Splitting procedure in scikit-learn

## Train-Test Split

```
# split the dataset into training set and test set
train, test = train_test_split(dataset, ...)
```

```
from sklearn.datasets import make_blobs
from sklearn.model_selection import train_test_split

# create dataset (To generate isotropic Gaussian blobs)
X, y = make_blobs(n_samples=2000)

# split the dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8)
```

... test_size=0.2

```
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

(1600, 2) (400, 2) (1600, ) (400, )

# Data Splitting procedure in scikit-learn

## Repeatable Train-Test Split

```
# split the dataset into training set and test set
train, test = train_test_split(dataset, ...)
```

Rows are randomly assigned into the training set and test set.

To ensure the subsets are a representative sample of the original dataset.

It is desirable to use the same subsets of the original dataset to fit and evaluate the machine learning models, when making comparisons.

```
# split the dataset into training set and test set
train, test = train_test_split(dataset, ... , random_state=0)
```

An arbitrary integer value (i.e. any value)

# Data Splitting procedure in scikit-learn

## Repeatable Train-Test Split

```
# demonstrate that the train-test split procedure is repeatable
from sklearn.datasets import make_blobs
from sklearn.model_selection import train_test_split
# create dataset (To generate isotropic Gaussian blobs)
X, y = make_blobs(n_samples=2000)
# split into train test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=9)
# summarise first 3 rows
print(X_train[:3, :])
# split again, and we should see the same split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=9)
# summarise first 3 rows
print(X_train[:3, :])
```

```
[[ 5.05649459 4.88266131]
 [ 9.80137957 -4.37162654]
 [ 9.17161937 -4.01807917]]
[[ 5.05649459 4.88266131]
 [ 9.80137957 -4.37162654]
 [ 9.17161937 -4.01807917]]
```

# Data Splitting procedure in scikit-learn

## Stratified Train-Test Split

### (for classification tasks only)

To preserve the same proportions of examples in each class as observed in the original dataset.

```
# split the dataset into training set and test set
train, test = train_test_split(dataset, ... , stratify=y)
```

# Data Splitting procedure in scikit-learn

## Stratified Train-Test Split (for classification tasks only)

```
# split imbalanced dataset into train and test sets without stratification
from collections import Counter
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
# create dataset
X, y = make_classification(n_samples=1000, weights=[0.95], flip_y=0, random_state=4)
print(Counter(y))
# split into train test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=4)
print(Counter(y_train))
print(Counter(y_test))
```

```
Counter({0: 950, 1: 50})
Counter({0: 478, 1: 22})
Counter({0: 472, 1: 28})
```

# Data Splitting procedure in scikit-learn

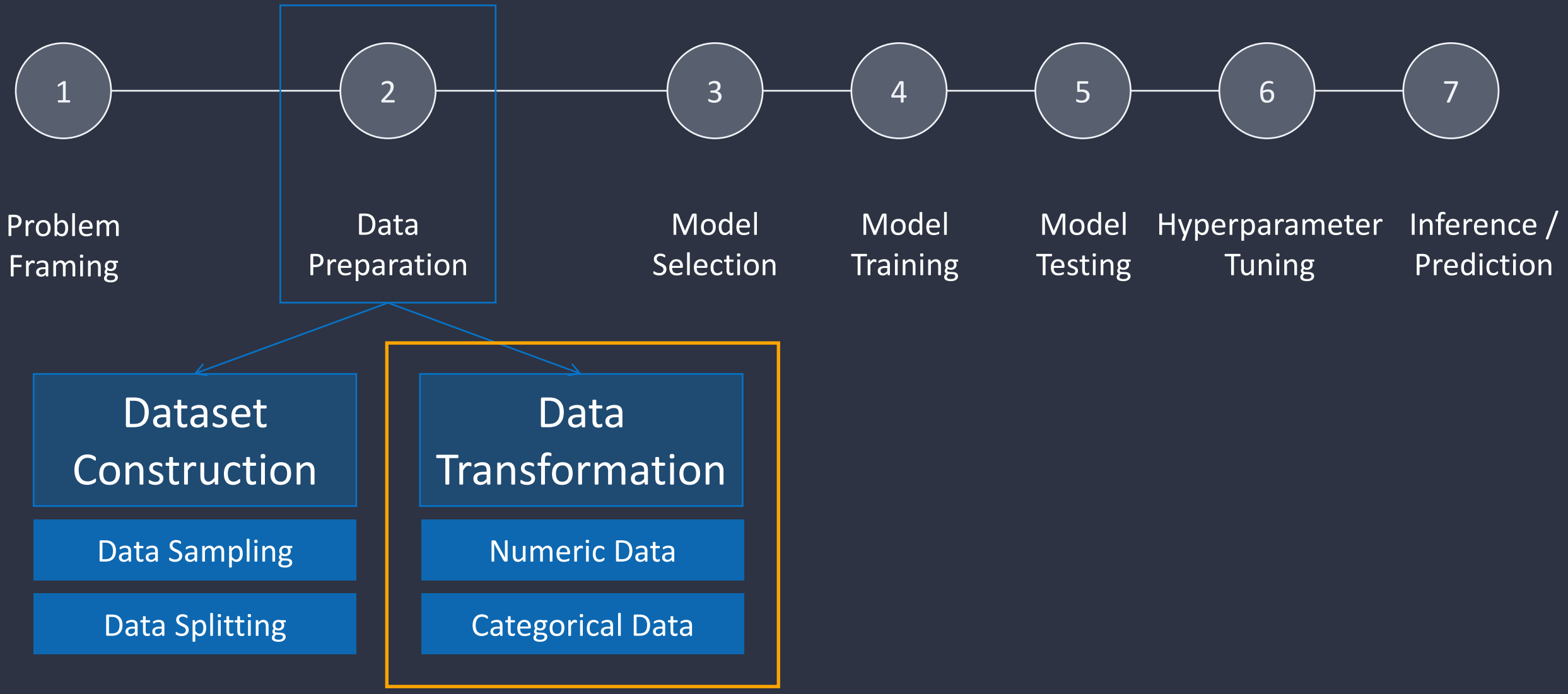
## Stratified Train-Test Split (for classification tasks only)

```
# split imbalanced dataset into train and test sets without stratification
from collections import Counter
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
# create dataset
X, y = make_classification(n_samples=1000, weights=[0.95], flip_y=0, random_state=4)
print(Counter(y))
# split into train test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=4 , stratify=y)
print(Counter(y_train))
print(Counter(y_test))
```

```
Counter({0: 950, 1: 50})
Counter({0: 475, 1: 25})
Counter({0: 475, 1: 25})
```

# Summary

# Summary


1
2
3
4
5
6
7
Problem Framing
Data Preparation
Model Selection
Model Training
Model Testing
Hyperparameter Tuning
Inference / Prediction
Dataset Construction
Data Sampling
Data Splitting
Data Transformation
Numeric Data
Categorical Data
Next lecture

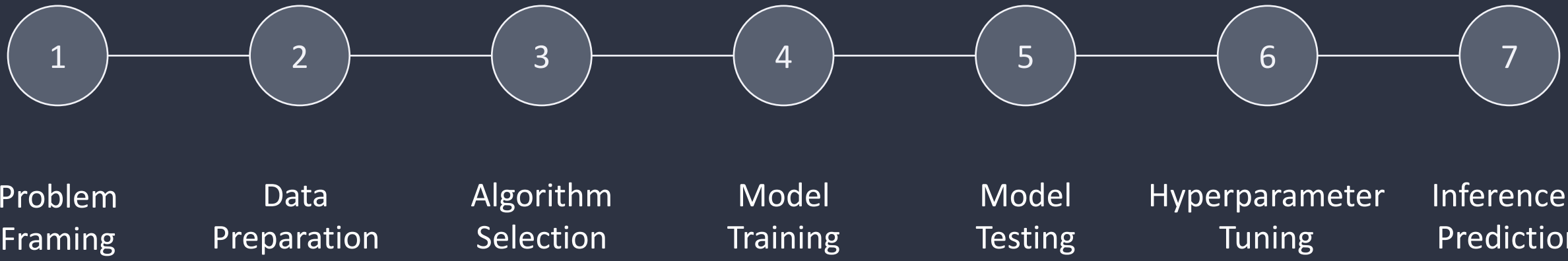
COMP2261 ARTIFICIAL INTELLIGENCE / MACHINE LEARNING

# Data Transformation

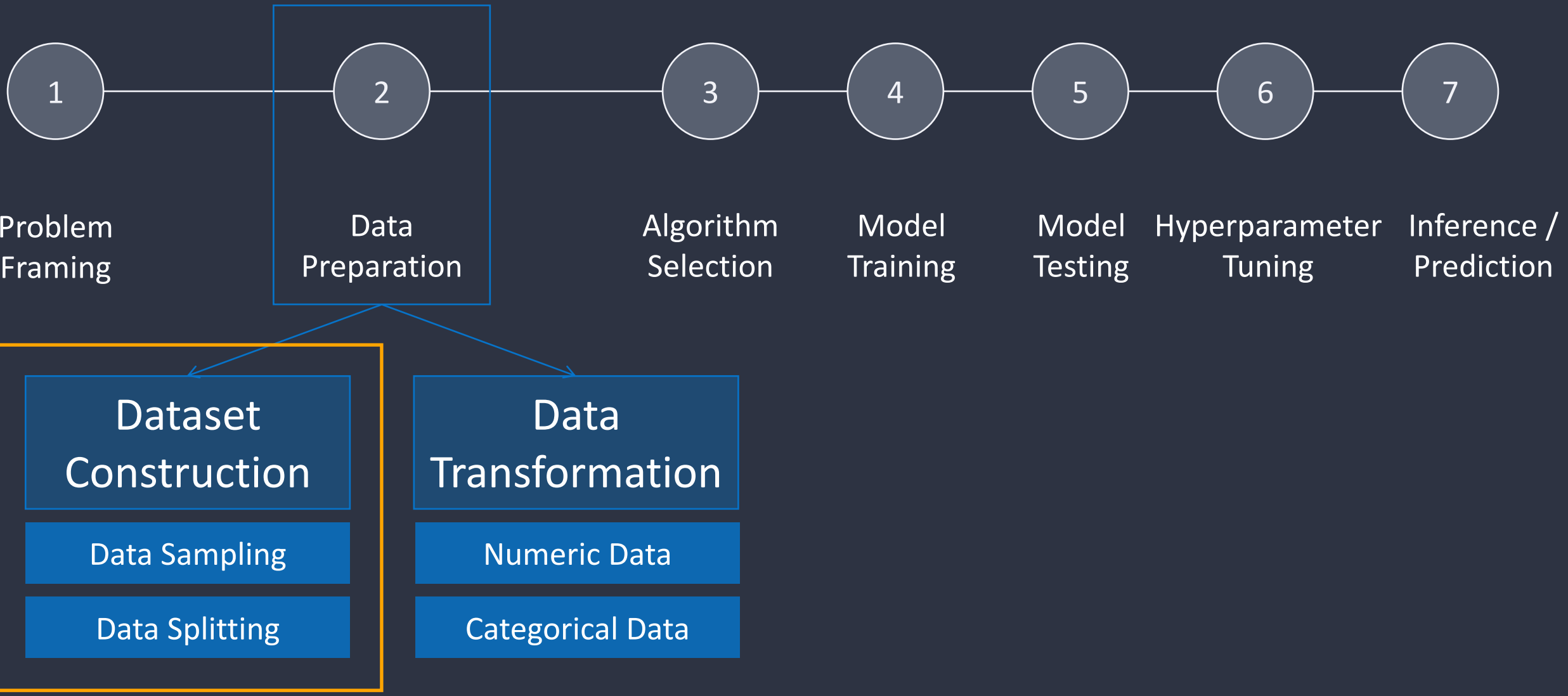
Dr Yang Long

Durham University

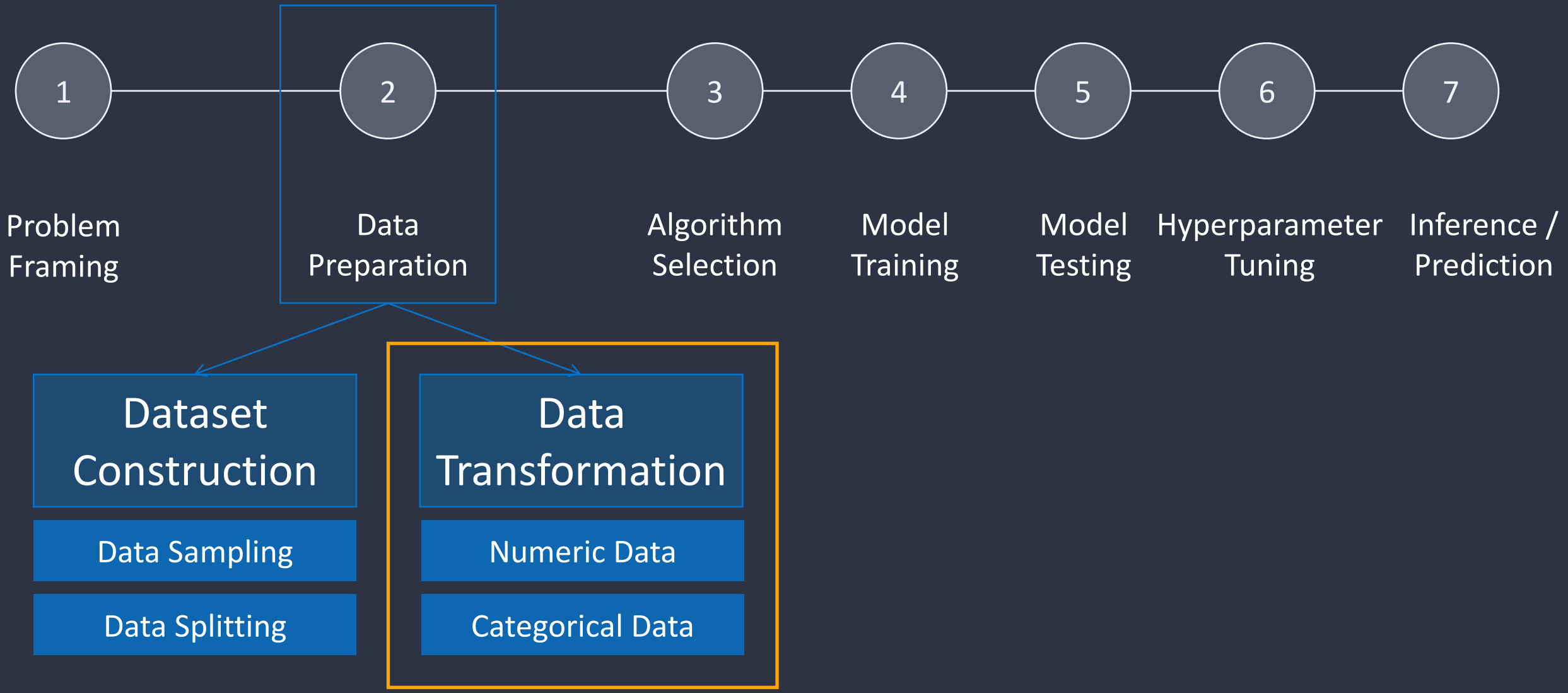
# Last Lecture



# Last Lecture



# Last Lecture



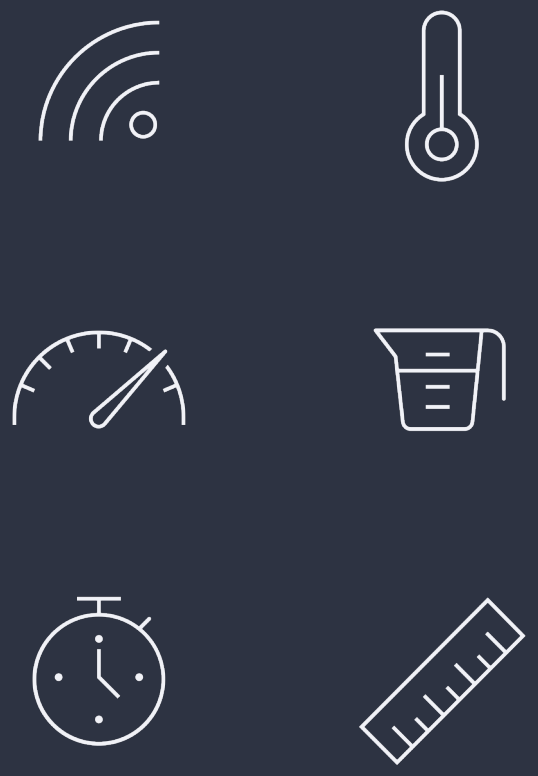
# Lecture Overview

1. Data Transformation Overview
2. Transforming Numeric Data
3. Transforming Categorical Data
4. Dealing with Missing Data

# 1. Data Transformation Overview

In Machine Learning projects, we need Data to train models.

# Data in real world doesn't come in a nice format that can be directly used.



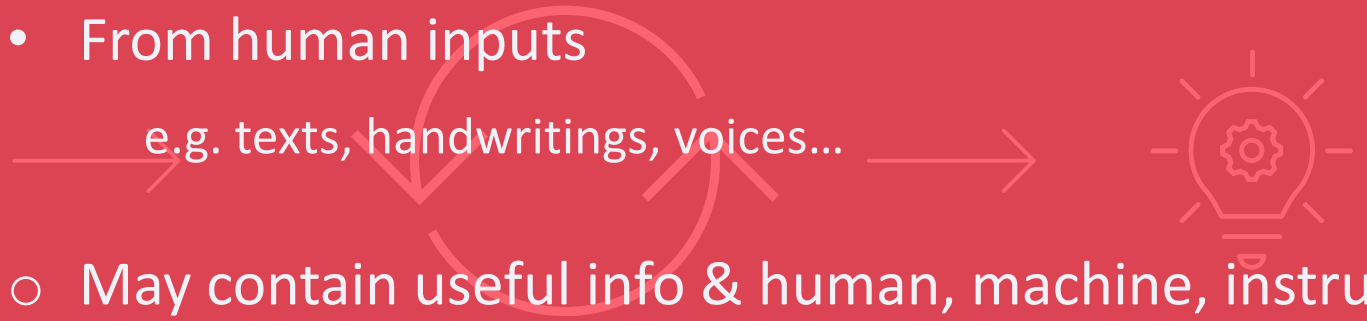
## Raw / source data (directly collected from a source)

- From an electronic sensor which measures physical input

  e.g. light, temperature, humidity, pressure, radiation level…

- From human inputs

  e.g. texts, handwritings, voices…



- May contain useful info & human, machine, instrument errors.
- May be in different colloquial formats, unformatted, uncoded or suspected such as outliers.
- May be in XML, JSON, relational database records, etc.

We need to put those heterogeneous data sources together and create **Feature Vectors** from it for learning algorithms.

# Feature Engineering

The process of selecting and extracting features from raw data.

art

?

science / engineering

Machine learning engineers often spend 70%~80% of their time in data preparation phase before modelling.

# Data Transformation

# Data Transformation

EXAMPLE. transforming raw data to feature vector

## Raw Data (in JSON)

```
0: {
  student_info: {
    first_name: Bruce,
    last_name: Lee,
    age: 18,
    sex: male,
    height: 1.72,
    num_module: -1
    …
  }
}
```

Feature Engineering

## Feature Vector

```
[
  32.0,
  96.0,
  18.0,
  0.0,
  1.72,
  -1.0,
  0.0,
  3.142,
  …
]
```

Feature engineering: the process of selecting and extracting features from raw data.

# Before feature transformation

Need to explore and clean up data.

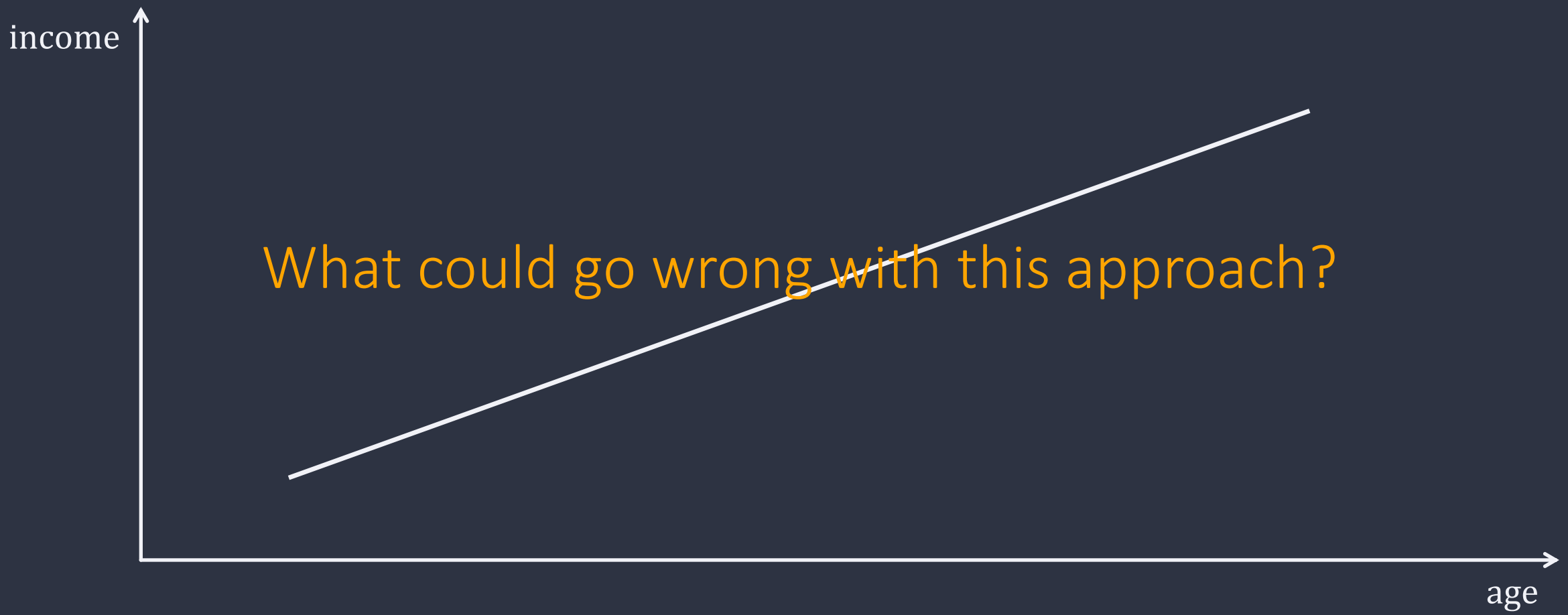
e.g., explore several rows of data and check basic statistics or fix missing entries.

Data can look one way in basic statistics and another when in graph

Visualise data in graphs and charts such as scatter plots,  lines and histograms.

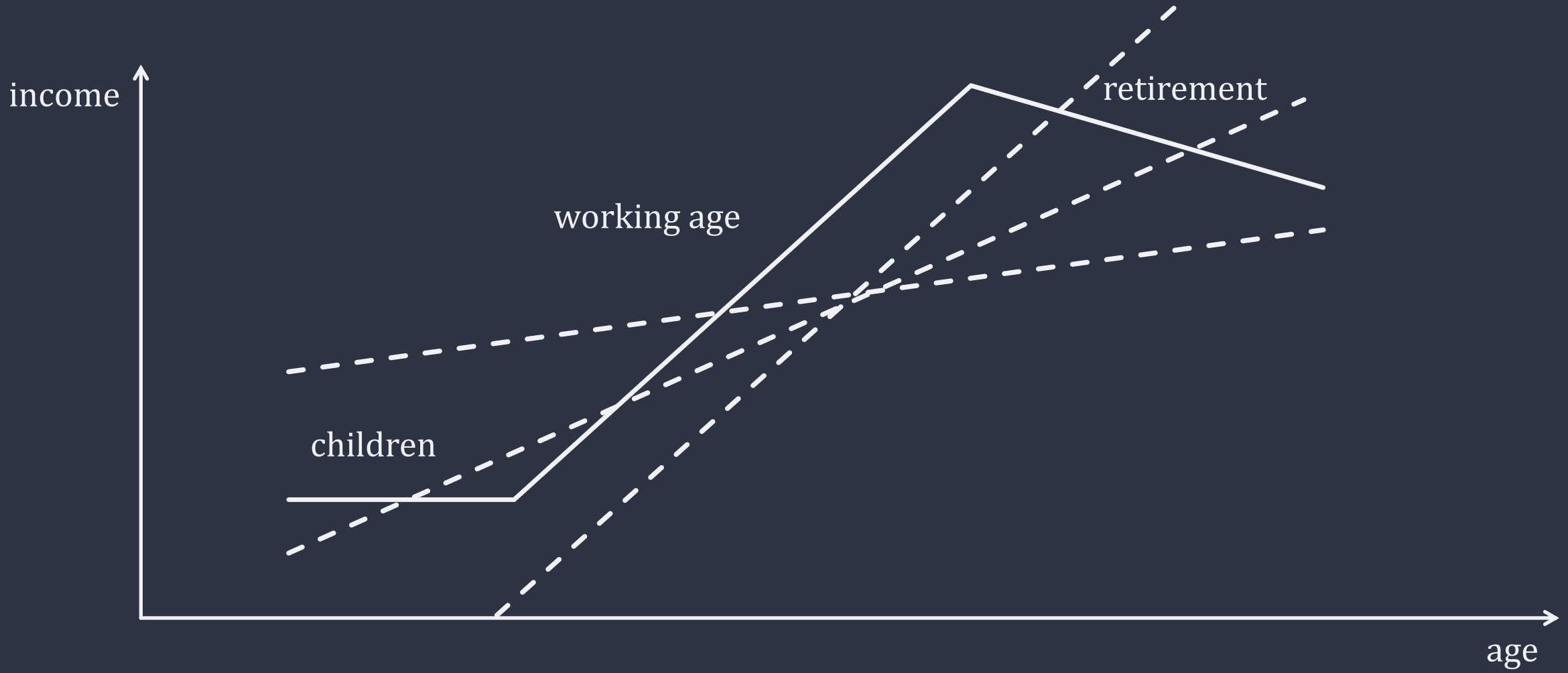
# 2. Transforming Numerical Data

EXAMPLE. use age to predict income




What could go wrong with this approach?

EXAMPLE. use age to predict income

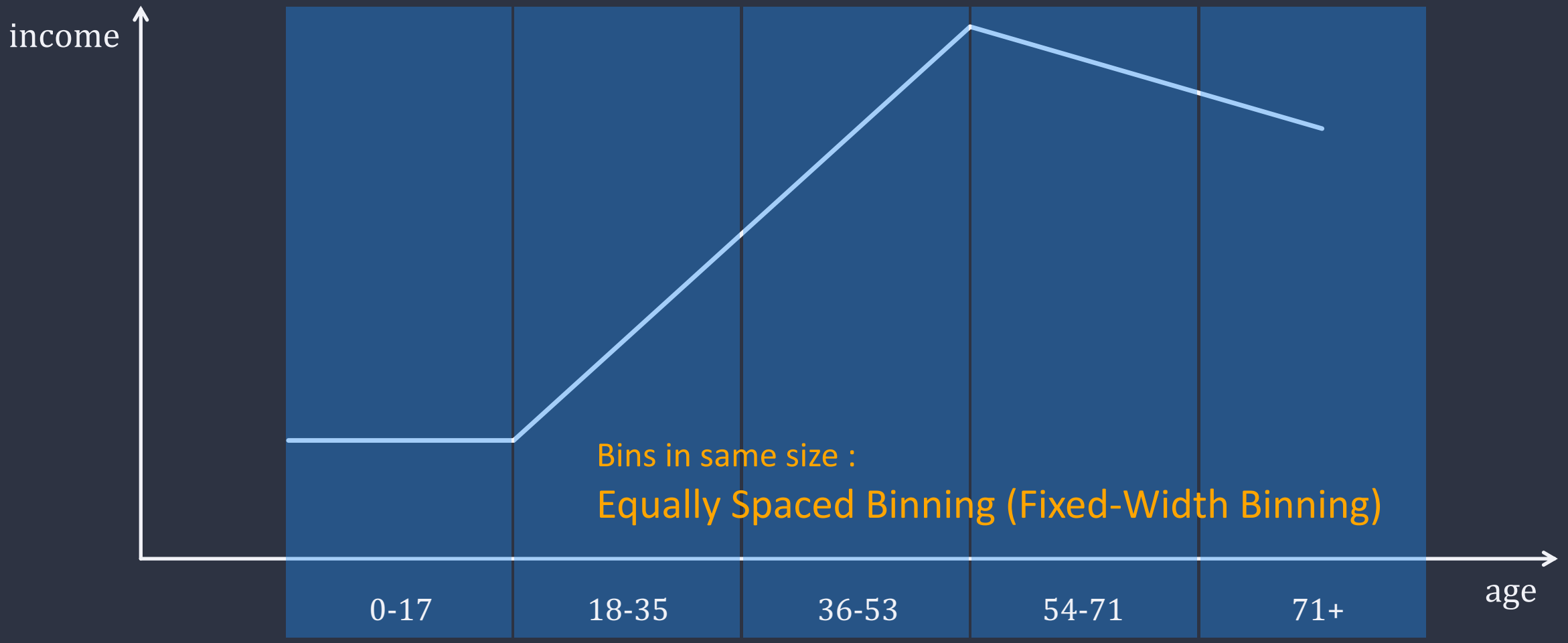


# Data Binning

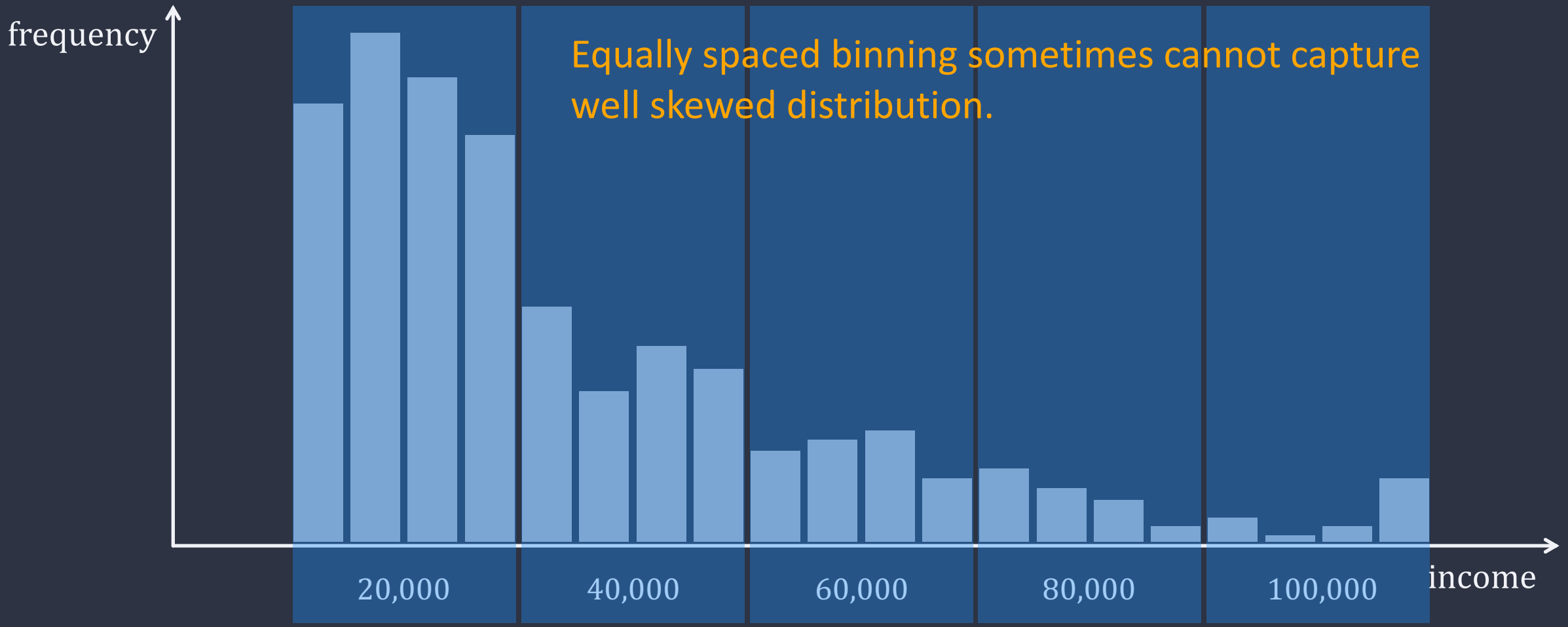
# Data Binning

EXAMPLE. use age to predict income

Transform numeric features into categorical ones based on range it falls into.

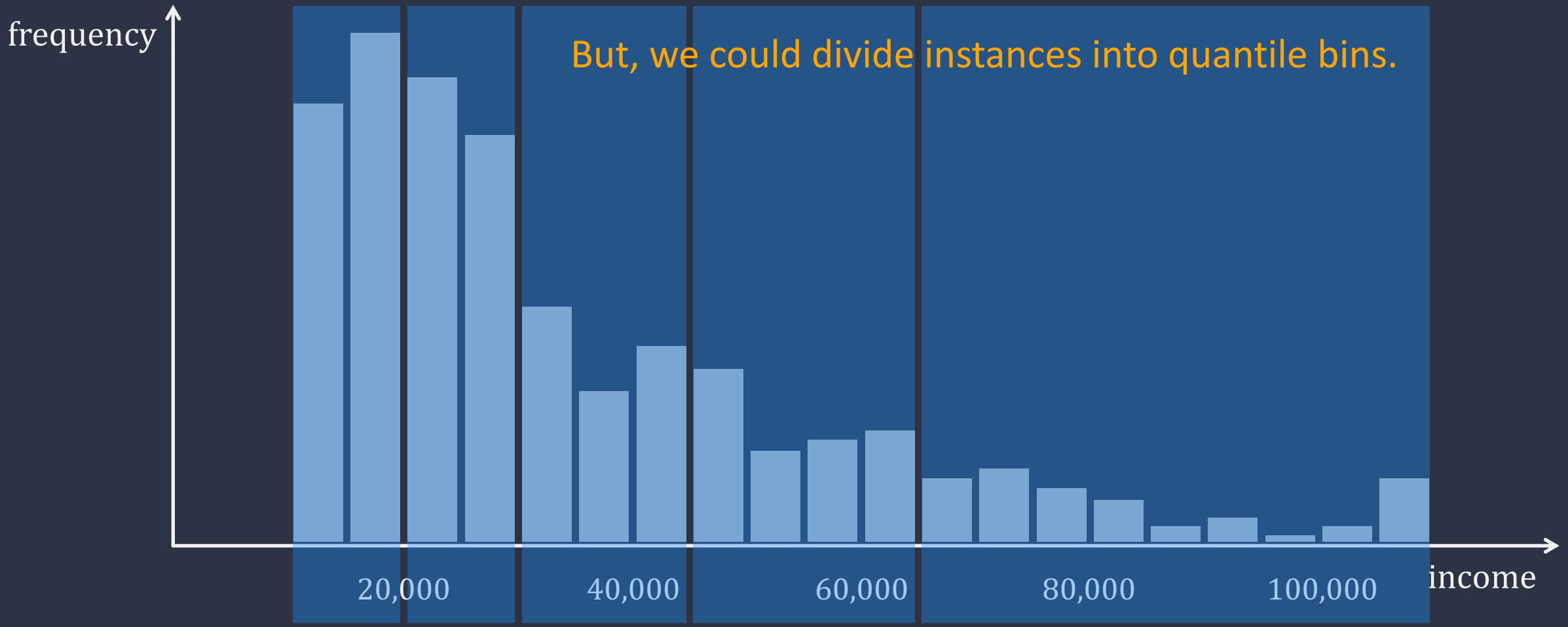


# Data Binning




Equally Spaced Binning (Fixed-Width Binning)

# Data Binning




~~Equally Spaced Binning (Fixed-Width Binning)~~

Quantile Binning

# Data Binning

When choosing data binning techniques, we must be clear about how to set the boundaries and which type of binning we want to use.

## Equally Spaced Binning

The boundaries are fixed and encompass the same range (e.g. age range: 18-35, 35-53,54-71). In this case, some bins may contain a lot more instances, and others may contain very few.

## Quantile Binning

The boundaries are not fixed and encompass a wide or narrow range (e.g. income: ~20k, 20k~30k, 30k~45k, 45k~65k, 65k~110k). In this case, each bin contains equal (or similar) number of instances.

## EXAMPLE. Features in very different ranges

A dataset containing two features, age and income.

Age ranges from 18-71; income ranges from 22,000-92,000


Income will intrinsically influence the result much more.

But it's not necessary that income is more important as a predictor than age.

# Feature Scaling

# Feature Scaling

- Can help transform the values of numeric features to be on a similar scale without distorting differences in ranges of values.

| | | |
|---|---|---|
| Min-Max Normalisation | Mean Normalisation | Standardisation |
| Unit-Length Scaling | Log Scaling | Clipping |

- It is necessary for many machine learning algorithms, e.g., many classifiers calculate the distance between 2 instances by e.g. Euclidean distance, so if one of the features is in a much larger range, it will dominate the distance.
- Gradient Descent converges faster with feature scaling (to cover later).

# Min-Max Normalisation

# Feature Scaling - Min-Max Normalisation

- The simplest technique to scale features in similar ranges.
- Can be used to rescale feature into the range of $[0, 1]$, via

$$x_{norm}^{(i)} = \frac{x^{(i)} - \min(x)}{\max(x) - \min(x)}$$

e.g. $age \in [18,71] \longrightarrow age_{norm}^{(i)} \in \left[\frac{18-18}{71-18}, \frac{71-18}{71-18}\right] = [0,1]$

$income \in [22000,92000] \longrightarrow income_{norm}^{(i)} \in \left[\frac{22000-22000}{92000-22000}, \frac{92000-22000}{92000-22000}\right] = [0,1]$

- Can be used to rescale feature into other ranges e.g. [-1, 1], depending on the nature of data and the learning algorithms to be used.

# Feature Scaling - Min-Max Normalisation

```
import numpy as np
x = np.random.uniform(5.0, 10.0, 10)
print(x)
```

```
[8.33230157  5.5394765  6.23918518  9.47122119  5.92845293  8.26185852
5.5694755   6.90963554 9.96401511  5.18957078]
```

```
x_norm = (x - x.min()) / (x.max() - x.min())
print(x_norm)
```

```
[0.65824012  0.07328721  0.21984012  0.89678507  0.15475773  0.64348593
0.07957046  0.36026491  1.          0.          ]
```

# Feature Scaling - Mean Normalisation (Normalisation)

- Rescale the feature values around the mean value.

$$x_{norm}^{(i)} = \frac{x^{(i)} - \text{mean}(x)}{\max(x) - \min(x)}$$

```
x_norm = (x - x.mean()) / (x.max() - x.min())
```

# Feature Scaling - Standardisation (Z-score Normalisation)

- Rescale the feature values to have zero-mean and unit-variance, i.e. $\mu = 0$ & $\sigma=1$

$$x_{std}^{(i)} = \frac{x^{(i)} - mean(x)}{std(x)} \quad \text{or} \quad z = \frac{x - \mu}{\sigma}$$

- Commonly used in many machine learning algorithms e.g. K-Nearest Neighbours and Support Vector Machines, Principal Component Analysis, Clustering, LASSO and Ridge regressions.
- Not necessary to machine learning algorithms which are not sensitive to the magnitude of features, e.g. Logistic Regression, Naive Bayes, and Tree-based algorithms such as Decision Tree, Random Forest and Gradient Boosting.

## Normalisation

$$x_{norm}^{(i)} = \frac{x^{(i)} - \text{mean}(x)}{\max(x) - \min(x)}$$

VS

## Standardisation

$$x_{std}^{(i)} = \frac{x^{(i)} - mean(x)}{std(x)}$$

- Normalisation can generate smaller standard deviations than Standardisation, so can scale out data to be more concentrated around the mean value.
- Normalisation doesn't require Gaussian distribution, so good for K-Nearest Neighbours and Neural Networks, but it doesn't handle well outliers; whereas Standardisation can help with cases where data follows Gaussian distribution, and it can better deal with outliers and facilitate convergence for e.g. Gradient Descent.
- We can always try fitting model to raw, normalised and standardised data and then compare their performances for the best results.

# Feature Scaling - Standardisation (Z-score Normalisation)

```
import numpy as np
x = np.random.uniform(5.0, 10.0, 10)
print(x)
```

```
[8.33230157  5.5394765  6.23918518  9.47122119  5.92845293  8.26185852
 5.5694755   6.90963554  9.96401511  5.18957078]
```

```
x_std = (x - x.mean()) / x.std()
print(x_std)
```

```
[ 0.72190025  -0.96980228  -0.54596659  1.4117799  -0.73418695  0.67923066
 -0.95163094  -0.13985359  1.71028078  -1.18175125]
```

# Feature Scaling - Unit-Length Scaling

- Rescale the components of a feature vector, so the complete vector's length is one.

$$x^{(i)}_{unit\_len} = \frac{x^{(i)}}{\left\| x^{(i)} \right\|}$$

if $\overrightarrow{x^{(i)}} = (x_1^{(i)}, x_2^{(i)}, \ldots, x_n^{(i)})$ then $\left\| \overrightarrow{x^{(i)}} \right\| = \sqrt{\left(x_1^{(i)}\right)^2 + \left(x_2^{(i)}\right)^2 + \cdots + \left(x_n^{(i)}\right)^2}$

- This is to normalise N-dimensional vector features to have unit length (length 1), similar to normalising 1-dementional features to have a range of $(0,1)$.

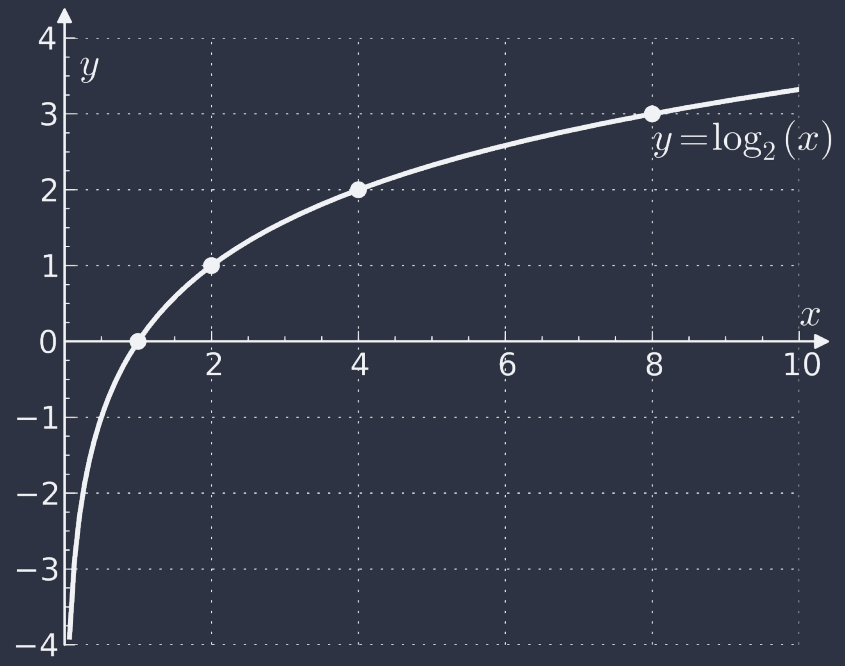
```
x_unit_len = x / np.linalg.norm(x)
```

# Feature Scaling - Log Scaling

- Rescale feature values to a narrow range.
- May make skewed numeric features to become normally distributed.

$$x_{scaled}^{(i)} = log(x^{(i)})$$

- The output of log function for positive values increases very slowly, and so higher values are marginalised more as compared to lower values.
- Very useful when dataset has many instances sharing a small range of values, but very few instances sharing a large range of values.




```
x_scaled = np.log(x)
```

# Feature Scaling - Clipping

- Caps all the feature values which are either above a specific max value or below a specific min value.
- Formula: set max/min values to avoid outliers.
- To be used when dataset containing extreme outliers.

  e.g. clip all height values above 2 meters to be exact 2 meters.

```
x = [1, 2, 3, 4, 5]
x_clipped = np.clip(x, 2, 4)
print(x_clipped)
```

```
[2 2 3 4 4]
```

# 3. Transforming Categorical Data

# What is Categorical Data?

# Categorical Data

- Categorical data is discrete.
- Each value of a categorical feature is called a category or a group, it could be represented as a string.

e.g.

Role: student | lecturer | demonstrator

Two types of categorical data: Nominal and Ordinal

# Categorical Data - Nominal (labelled)

- Nominal data is also called labelled data.
- Values are not comparable or calculatable.
- e.g. race, gender, sexuality, religion, nationality, blood type…

Labour vs Conservative longer, heavier, warmer??

Liberal Democrats + Scottish National = ??

Democratic Unionist - Sinn Féin = ??

# Categorical Data - Ordinal

- Ordinal data is with a set order or scale to it.
- Values are comparable.
- e.g. initial | intermediate | advanced

  XS | S | M | L | XL

  First | Second | Third
- There is an ordered relationship between categories but this order does not have a standard scale to be measured.

  intermediate = initial + 1

  Third = First x 3

# Categorical Data

- Sometimes, we must use categorical data instead of numeric data to present certain features even if they contain integer values.

# Categorical Data

- Machine learning algorithms require features to be numbers.
- Encoding categorical data in order to be able to fit and evaluate models.
- Encoding techniques

| | |
|---|---|
| Ordinal Encoding | Nominal Encoding |
| One-Hot Encoding | Feature Crossing |

# Ordinal Encoding

- To transform categorical features into ordinal numbers in ordered set, i.e. to assign a sequence of numerical values as per the order of data.
- In most cases, we map each unique label to an integer value.
- We can only use ordinal encoding when there is a known relationship between the categories.
- If we map inherent ordinality of an ordinal feature to a wrong scale, there will be a very negative impact on model's performance.

# EXAMPLE. Customer feedback survey

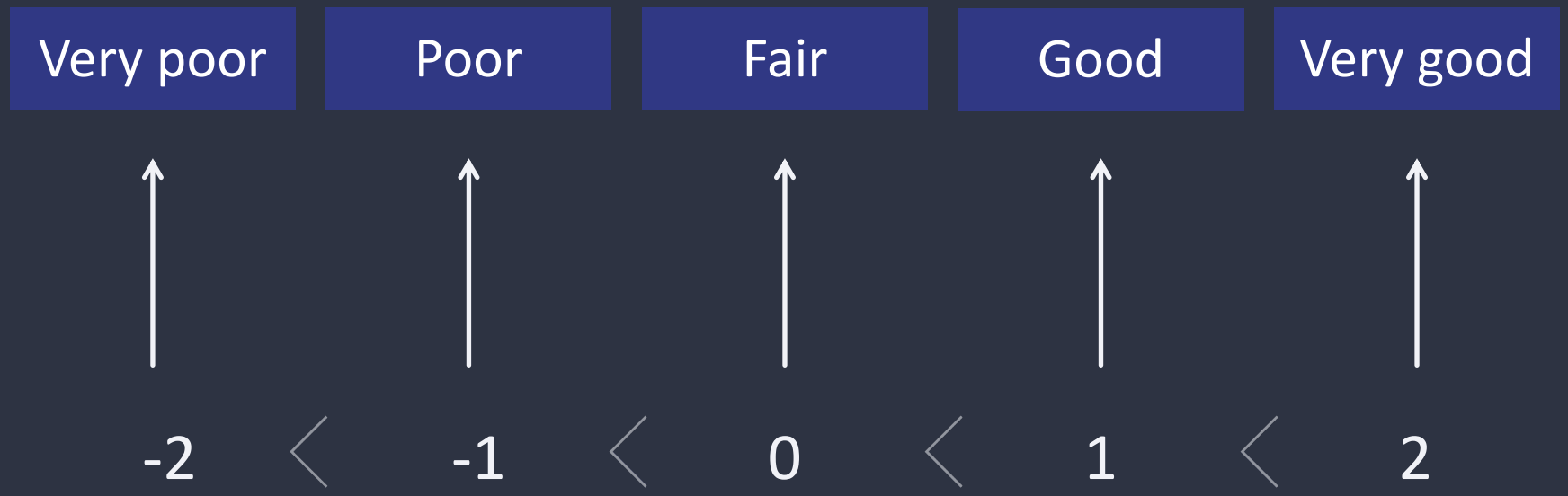
Customer rating for service, in an ordered relationship

| Very poor | Poor | Fair | Good | Very good |
|---|---|---|---|---|

(e.g. good is better than fair; fair is better than poor…)

EXAMPLE. Customer feedback survey

Customer rating for service, in an ordered relationship




(e.g. good is better than fair; fair is better than poor…)

# EXAMPLE. Customer feedback survey

Customer rating for service, in an ordered relationship

| Very poor | Poor | Fair | Good | Very good |
|---|---|---|---|---|

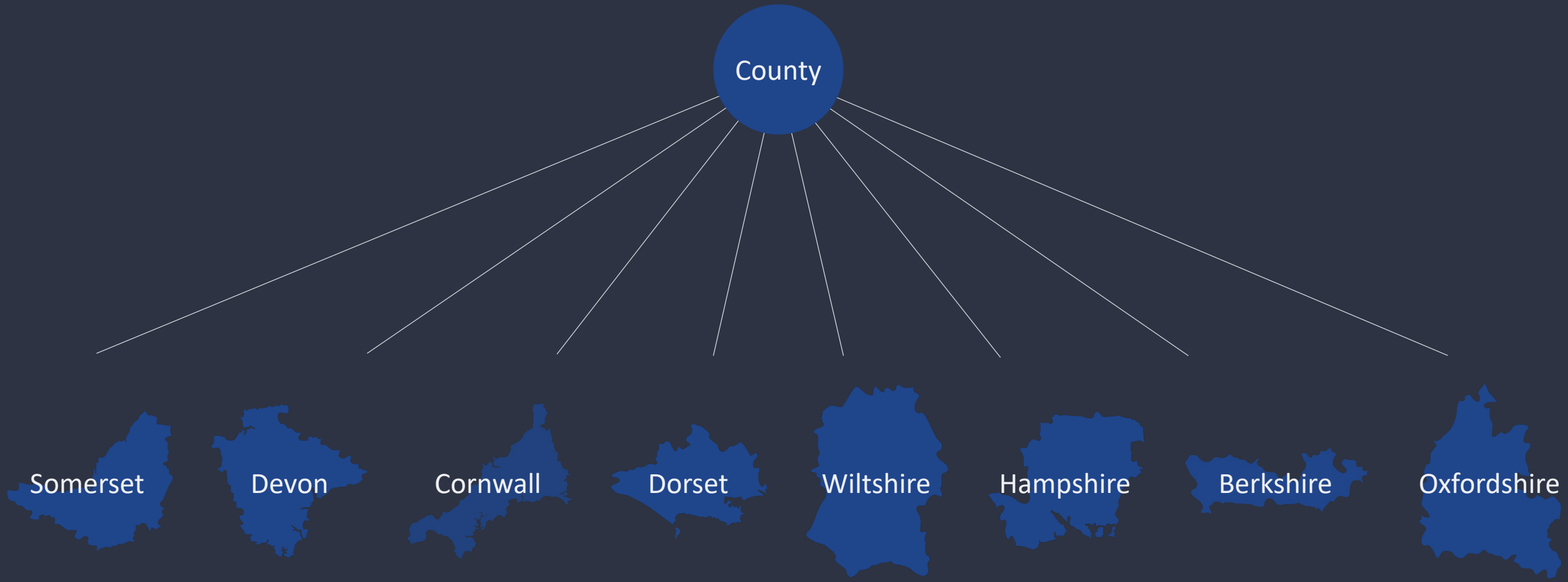
```
mapping_dict = { 'very_poor': -2,
                 'poor': -1,
                 'fair': 0,
                 'good': 1,
                 'very_good': 2}

df['size'] = df['size']. map(mapping_dict)
```

# Nominal Encoding

- Machine learning algorithms don’t assume an ordering in class labels.
- Use LabelEncoder from scikit-learn to transform class labels into integers.

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['classlabel'] = le.fit_transform(df['classlabel'])
```

# EXAMPLE. Counties of England



# One-Hot Encoding

EXAMPLE. Counties of England

| County |
| --- |
| Berkshire |
| Cornwall |
| Devon |
| Dorset |
| Hampshire |
| Oxfordshire |
| Somerset |
| Wiltshire |

# One-Hot Encoding

EXAMPLE. Counties of England

| County | Berkshire | Cornwall | Devon | Dorset | Hampshire | Oxfordshire | Somerset | Wiltshire |
|---|---|---|---|---|---|---|---|---|
| Berkshire | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Cornwall | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| Devon | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| Dorset | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| Hampshire | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| Oxfordshire | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| Somerset | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| Wiltshire | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

# One-Hot Encoding

EXAMPLE. Counties of England

Out-Of-Vocabulary Bin (OOV bin)

| Berkshire | Cornwall | Devon | Dorset | Hampshire | Oxfordshire | Somerset | Wiltshire | Other |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

# One-Hot Encoding

**EXAMPLE.** Counties of England

Cornwall ⟶ $[0, \mathbf{1}, 0, 0, 0, 0, 0, 0, 0]$

Dorset ⟶ $[0, 0, 0, \mathbf{1}, 0, 0, 0, 0, 0]$

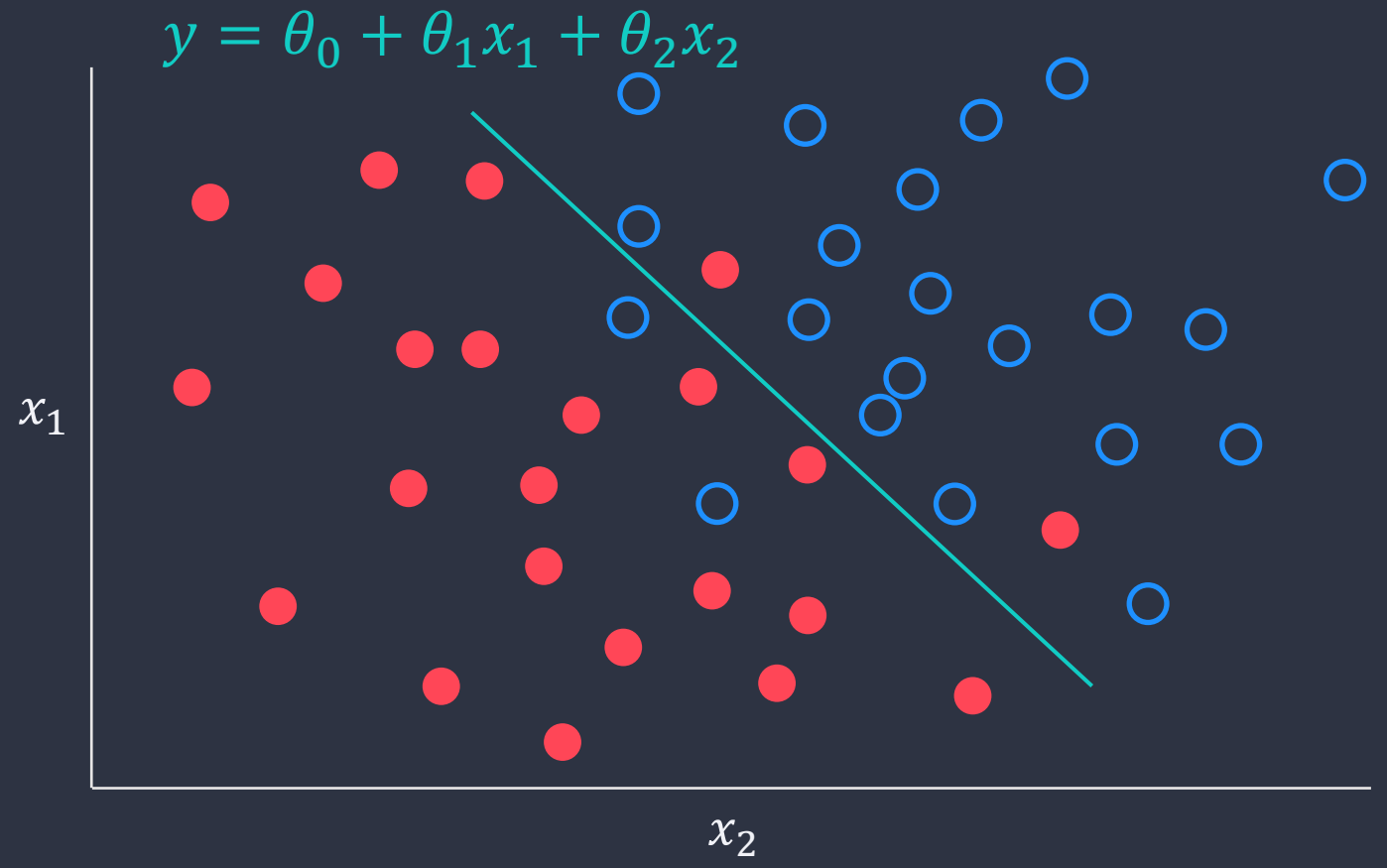
Somerset ⟶ $[0, 0, 0, 0, 0, 0, \mathbf{1}, 0, 0]$

Durham ⟶ $[0, 0, 0, 0, 0, 0, 0, 0, \mathbf{1}]$ (“other”)

```
df = pd.get_dummies (df, columns=['county'])
```

# Feature Crossing

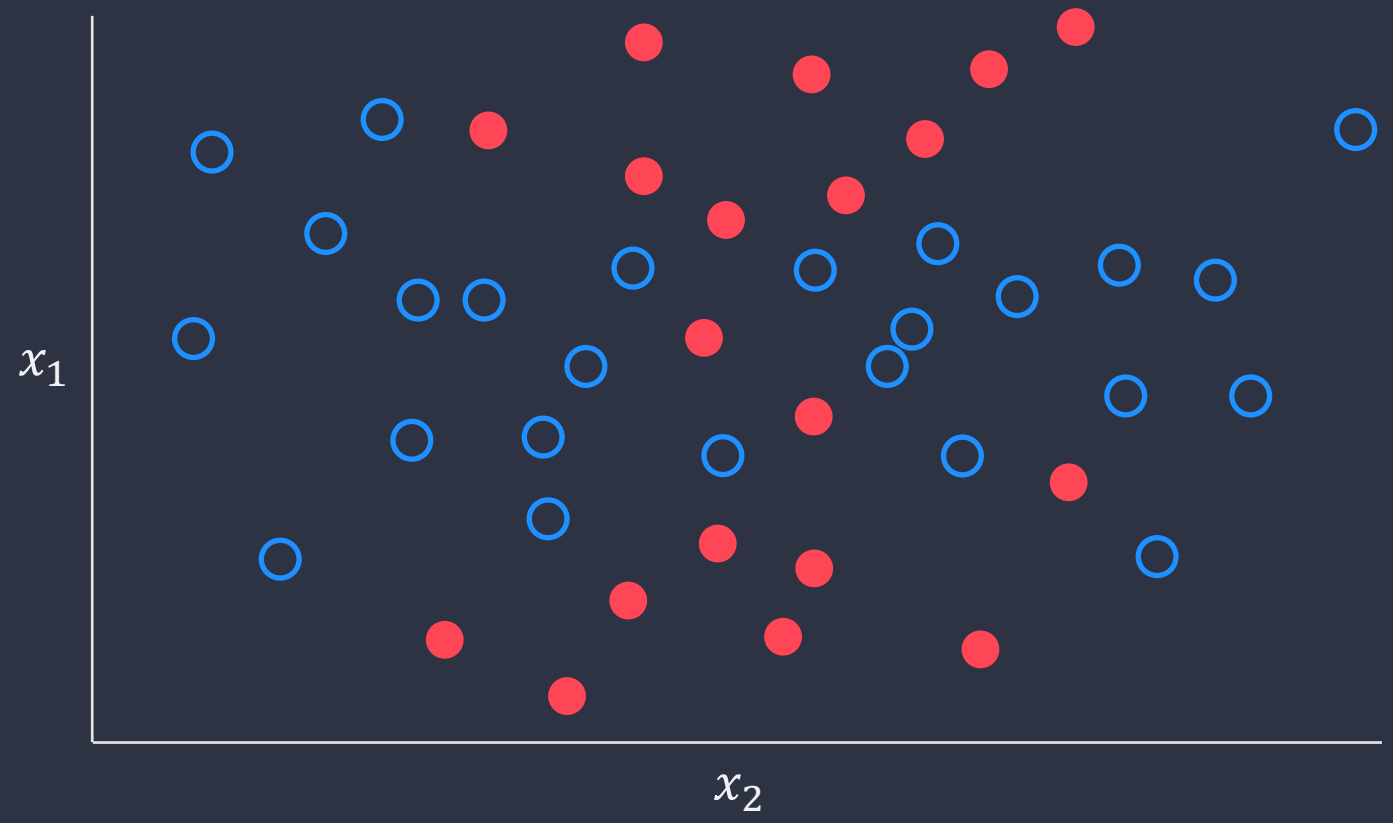
- A feature cross is a synthetic feature formed by multiplying 2 or more features.
- Can bring predictive abilities beyond what those individual features can provide.




Linearly separable

# Feature Crossing

- A feature cross is a synthetic feature formed by multiplying 2 or more features.
- Can bring predictive abilities beyond what those individual features can provide.




Can you try to draw a single straight line to separate them?

nonlinear problem

# Feature Crossing

- To solve a nonlinear problem

  Create a feature cross $x_3$.

$$x_3 = x_1 x_2$$

  Treat the new synthetic feature cross $x_3$ just like any other feature.

  The linear model becomes:

$$y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3$$

  $\theta_3 x_3$ encodes nonlinear information, so no need to change how the linear model trains to determine the value for $\theta_3$.

# Feature Crossing

- Different types

| | |
|---|---|
| $[A \times B]$ | formed by multiplying the values of two features. |
| $[A \times B \times C \times D]$ | formed by multiplying the values of multiple features. |
| $[A \times A]$ | formed by squaring a single feature. |
| $[A \times A \times A \times A]$ | formed by multiplying the values of a single feature multiple times. |

# 4. Dealing with Missing Data

# Dealing with Missing Data

- Missing data (or missing values), occur when no value is stored for the variable in an example.
- Common & can significantly affect conclusions drawn from the data.
- Many different ways for dealing with missing data.

Diabetes Data

| | AGE | SEX | BMI | BP | S1 | S2 |
|---|---|---|---|---|---|---|
| 1 | 59 | 2 | 32.1 | 101 | 157 | 93.2 |
| 2 | 48 | 1 | 21.6 | 87 | 183 | 103.2 |
| 3 | 72 | 2 | 30.5 | missing | 156 | 93.6 |
| 4 | 24 | 1 | 25.3 | 84 | 198 | missing |
| 5 | 50 | 1 | 23 | missing | 192 | 125.4 |
| 6 | missing | 1 | 22.6 | 89 | 139 | 64.8 |
| 7 | 36 | 2 | 22 | 90 | 160 | 99.6 |
| 8 | 66 | 2 | 26.2 | 114 | 255 | 185 |

```
df.isnull().sum()
```

| | |
|---|---|
| AGE | 1 |
| SEX | 0 |
| BMI | 0 |
| BP | 2 |
| S1 | 0 |
| S2 | 1 |

# Dealing with Missing Data

- Dropping rows/columns with missing values

Diabetes Data

| | AGE | SEX | BMI | BP | S1 | S2 |
|---|---|---|---|---|---|---|
| 1 | 59 | 2 | 32.1 | 101 | 157 | 93.2 |
| 2 | 48 | 1 | 21.6 | 87 | 183 | 103.2 |
| ~~3~~ | ~~72~~ | ~~2~~ | ~~30.5~~ | ~~missing~~ | ~~156~~ | ~~93.6~~ |
| ~~4~~ | ~~24~~ | ~~1~~ | ~~25.3~~ | ~~84~~ | ~~198~~ | ~~missing~~ |
| ~~5~~ | ~~50~~ | ~~1~~ | ~~23~~ | ~~missing~~ | ~~192~~ | ~~125.4~~ |
| ~~6~~ | ~~missing~~ | ~~1~~ | ~~22.6~~ | ~~89~~ | ~~139~~ | ~~64.8~~ |
| 7 | 36 | 2 | 22 | 90 | 160 | 99.6 |
| 8 | 66 | 2 | 26.2 | 114 | 255 | 185 |

- Drop rows with missing values

```
df.dropna(axis=0)
```

| | AGE | SEX | BMI | BP | S1 | S2 |
|---|---|---|---|---|---|---|
| 1 | 59 | 2 | 32.1 | 101 | 157 | 93.2 |
| 2 | 48 | 1 | 21.6 | 87 | 183 | 103.2 |
| 7 | 36 | 2 | 22 | 90 | 160 | 99.6 |
| 8 | 66 | 2 | 26.2 | 114 | 255 | 185 |

# Dealing with Missing Data

- Dropping rows/columns with missing values

Diabetes Data

| | AGE | SEX | BMI | BP | S1 | S2 |
|---|---|---|---|---|---|---|
| 1 | 59 | 2 | 32.1 | 101 | 157 | 93.2 |
| 2 | 48 | 1 | 21.6 | 87 | 183 | 103.2 |
| 3 | 72 | 2 | 30.5 | missing | 156 | 93.6 |
| 4 | 24 | 1 | 25.3 | 84 | 198 | missing |
| 5 | 50 | 1 | 23 | missing | 192 | 125.4 |
| 6 | missing | 1 | 22.6 | 89 | 139 | 64.8 |
| 7 | 36 | 2 | 22 | 90 | 160 | 99.6 |
| 8 | 66 | 2 | 26.2 | 114 | 255 | 185 |

- Drop columns with missing values

```
df.dropna(axis=1)
```

| | SEX | BMI | S1 |
|---|---|---|---|
| 1 | 2 | 32.1 | 157 |
| 2 | 1 | 21.6 | 183 |
| 3 | 2 | 30.5 | 156 |
| 4 | 1 | 25.3 | 198 |
| 5 | 1 | 23 | 192 |
| 6 | 1 | 22.6 | 139 |
| 7 | 2 | 22 | 160 |
| 8 | 2 | 26.2 | 255 |

# Dealing with Missing Data

- Replacing with mean/median/mode

Diabetes Data

| | AGE | SEX | BMI | BP | S1 | S2 |
|---|---|---|---|---|---|---|
| 1 | 59 | 2 | 32.1 | 101 | 157 | 93.2 |
| 2 | 48 | 1 | 21.6 | 87 | 183 | 103.2 |
| 3 | 72 | 2 | 30.5 | missing | 156 | 93.6 |
| 4 | 24 | 1 | 25.3 | 84 | 198 | missing |
| 5 | 50 | 1 | 23 | missing | 192 | 125.4 |
| 6 | missing | 1 | 22.6 | 89 | 139 | 64.8 |
| 7 | 36 | 2 | 22 | 90 | 160 | 99.6 |
| 8 | 66 | 2 | 26.2 | 114 | 255 | 185 |

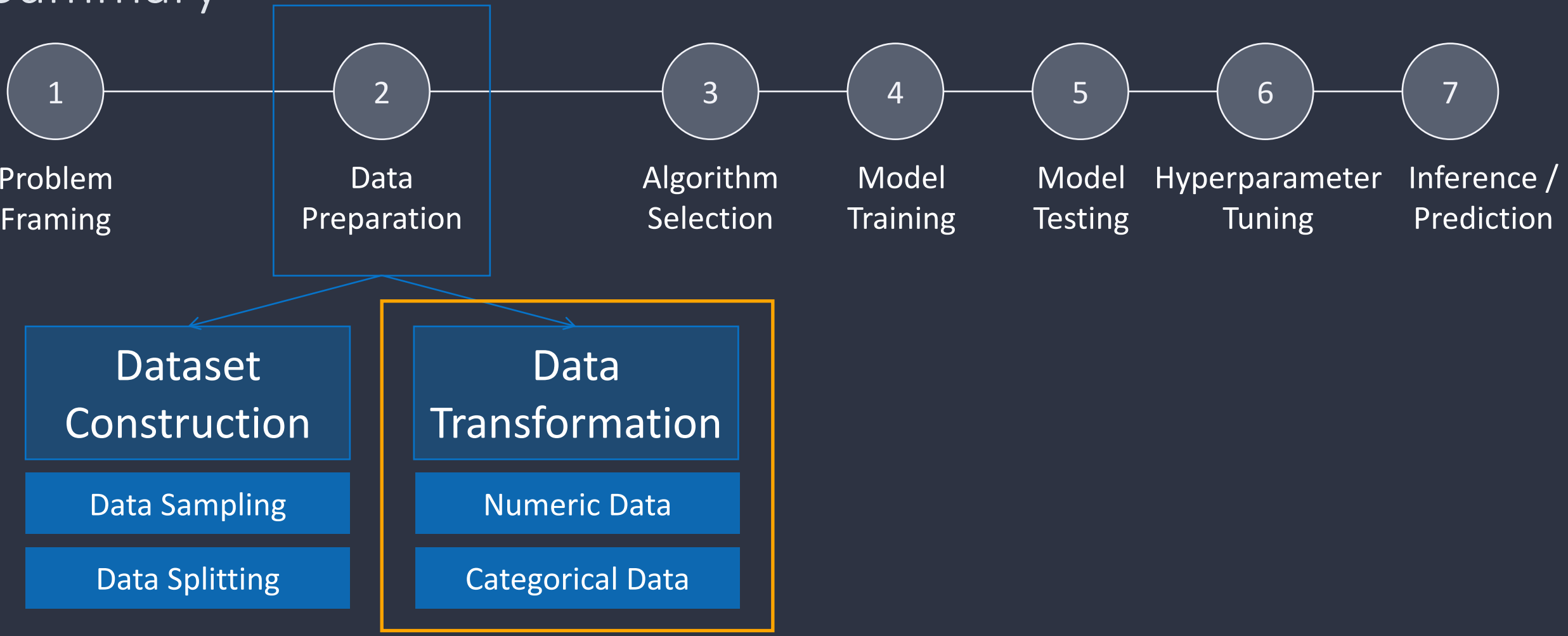
```
from sklearn.impute import SimpleImputer

imputer = SimpleImputer(missing_values=np.nan,
                                    strategy='mean’)
X = df.values
X = imputer.fit_transform(df.values)
X
```

# Dealing with Missing Data

- Predicting missing values, e.g. regression
- Using machine learning algorithms that support missing values
- …

# Summary

# Summary




## Next Lecture

Machine Learning Algorithms