# Designing an end-to-end machine learning use case

END-TO-END MACHINE LEARNING

**Joshua Stapleton**
Machine Learning Engineer

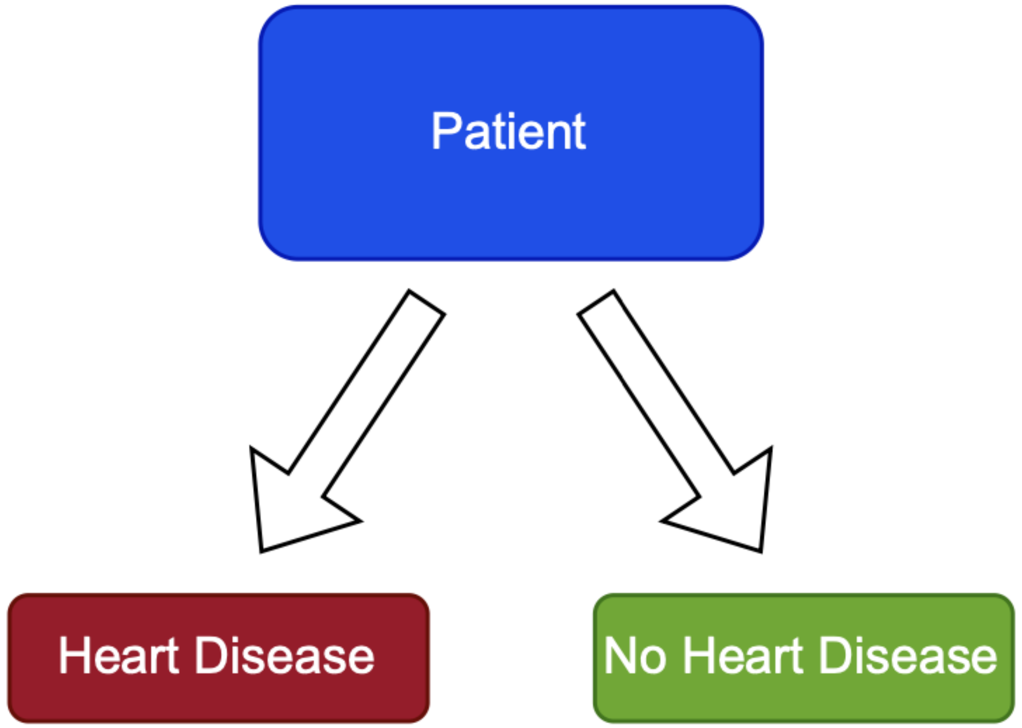
# The case study

- Predicting heart disease
- Goal: inform decision-making of cardiologists

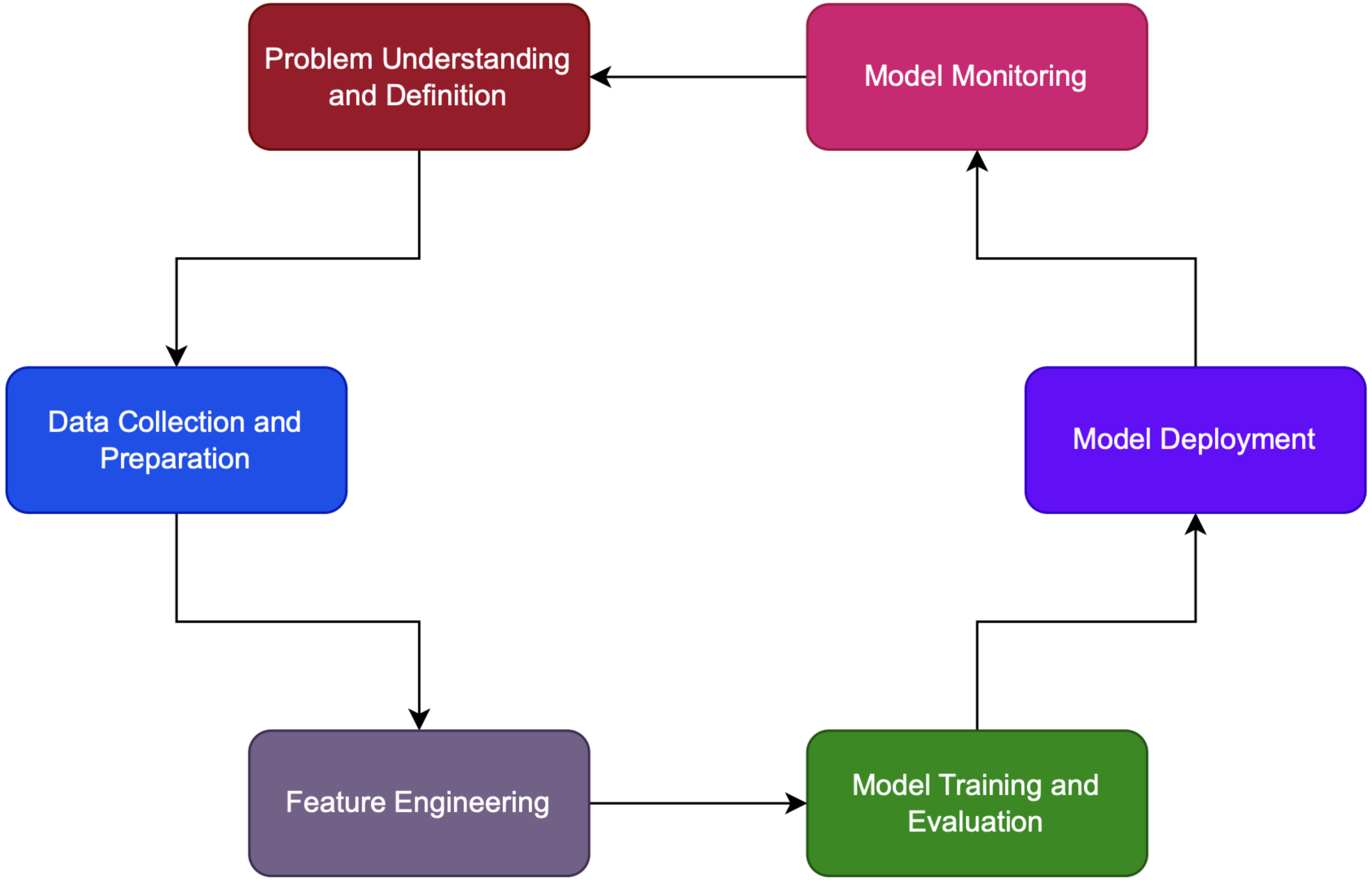

[1] Image source: https://www.flaticon.com/free-icons/doctor

# The model's role

- Models can **inform**, but should not **make** decisions
- Especially important in healthcare



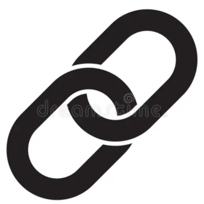
# The machine learning lifecycle



# Understanding end user requirements

## Accuracy

## Reliability

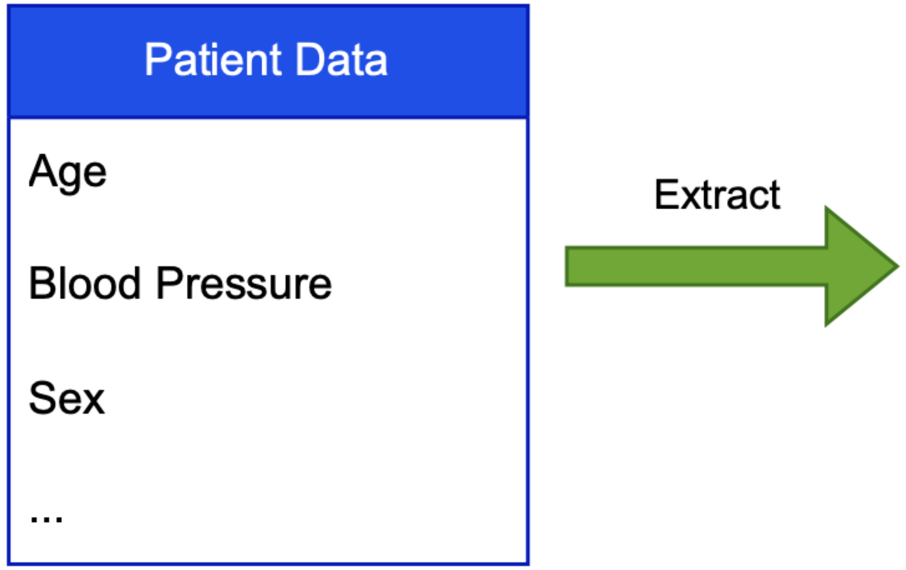


## Security

## Interpretability

# Data collection

- Collect relevant data
  - Private dataset from company
  - Public dataset
- Understand data and context
  - Representation and measurement
  - Potential bias



# Let's practice!

END-TO-END MACHINE LEARNING

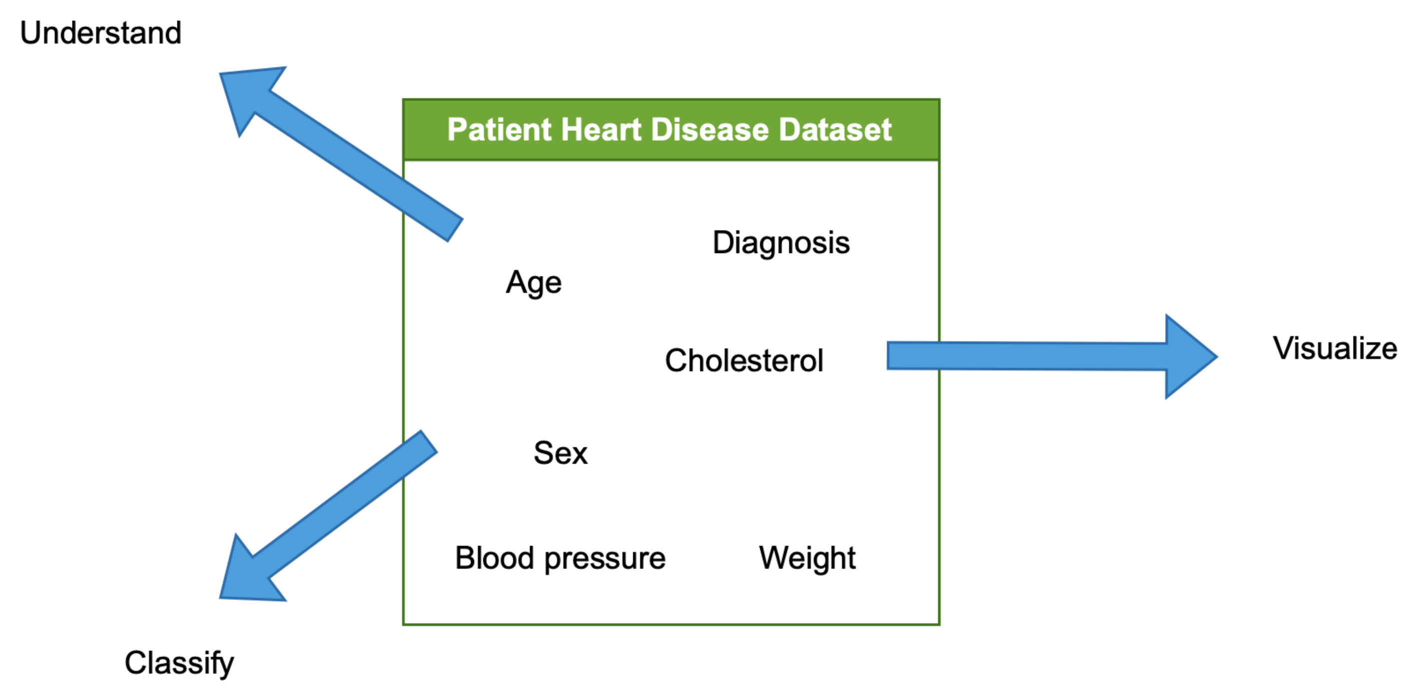
# Exploratory Data Analysis

END-TO-END MACHINE LEARNING

**Joshua Stapleton**
Machine Learning Engineer

# The EDA process

- Examine and analyse the dataset
- Understand the dataset
- Visualize the dataset
- Characterize / classify the dataset



# Understanding our data

`df.head()`

- Shows first rows of the dataset
- Provides snapshot of data's structure

```python
# Print the first 5 columns
print(heart_disease_df.head())
```

| | age | sex | cp | trestbps | chol | fbs | restecg | thalach | exang | oldpeak | slope | ca | thal | target |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 52 | 1 | 0.0 | 125 | 212.0 | 0 | 1.0 | 168 | 0 | NaN | 2 | 2 | 3 | 0 |
| **1** | 53 | 1 | 0.0 | 140 | 203.0 | 1 | 0.0 | 155 | 1 | NaN | 0 | 0 | 3 | 0 |
| **2** | 70 | 1 | 0.0 | 145 | 174.0 | 0 | 1.0 | 125 | 1 | NaN | 0 | 0 | 3 | 0 |
| **3** | 61 | 1 | 0.0 | 148 | 203.0 | 0 | 1.0 | 161 | 0 | NaN | 2 | 1 | 3 | 0 |
| **4** | 62 | 0 | 0.0 | 138 | 294.0 | 1 | 1.0 | 106 | 0 | NaN | 1 | 3 | 2 | 0 |

`df.info()`

- Summarizes features
- Shows non-null entries and feature types

```python
# Print out details
print(heart_disease_df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1076 entries, 0 to 1075
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       1076 non-null   int64
 1   sex       1076 non-null   int64
 2   cp        1023 non-null   float64
 3   trestbps  1076 non-null   int64
 4   chol      1021 non-null   float64
 5   fbs       1076 non-null   int64
 6   restecg   1028 non-null   float64
 7   thalach   1076 non-null   int64
 8   exang     1076 non-null   int64
 9   oldpeak   0 non-null      float64
 10  slope     1076 non-null   int64
 11  ca        1076 non-null   int64
 12  thal      1076 non-null   int64
 13  target    1076 non-null   int64
dtypes: float64(4), int64(10)
memory usage: 117.8 KB
```

# Class (im)balance

`df.value_counts()`

- Counts number of unique occurrences of each class
- Class: binary presence of heart disease (1/0)
- Important for modeling

```
# print the class balance
print(heart_disease_df['target'].value_counts(normalize=True))
```

```
1     551
0     525
Name: target, dtype: int64
```

# Missing values

- Can lead to errors
- Unrepresentative, biased results

Use `df.isnull()`

- Checks for null/empty/missing values
- Applied to column or collection of columns

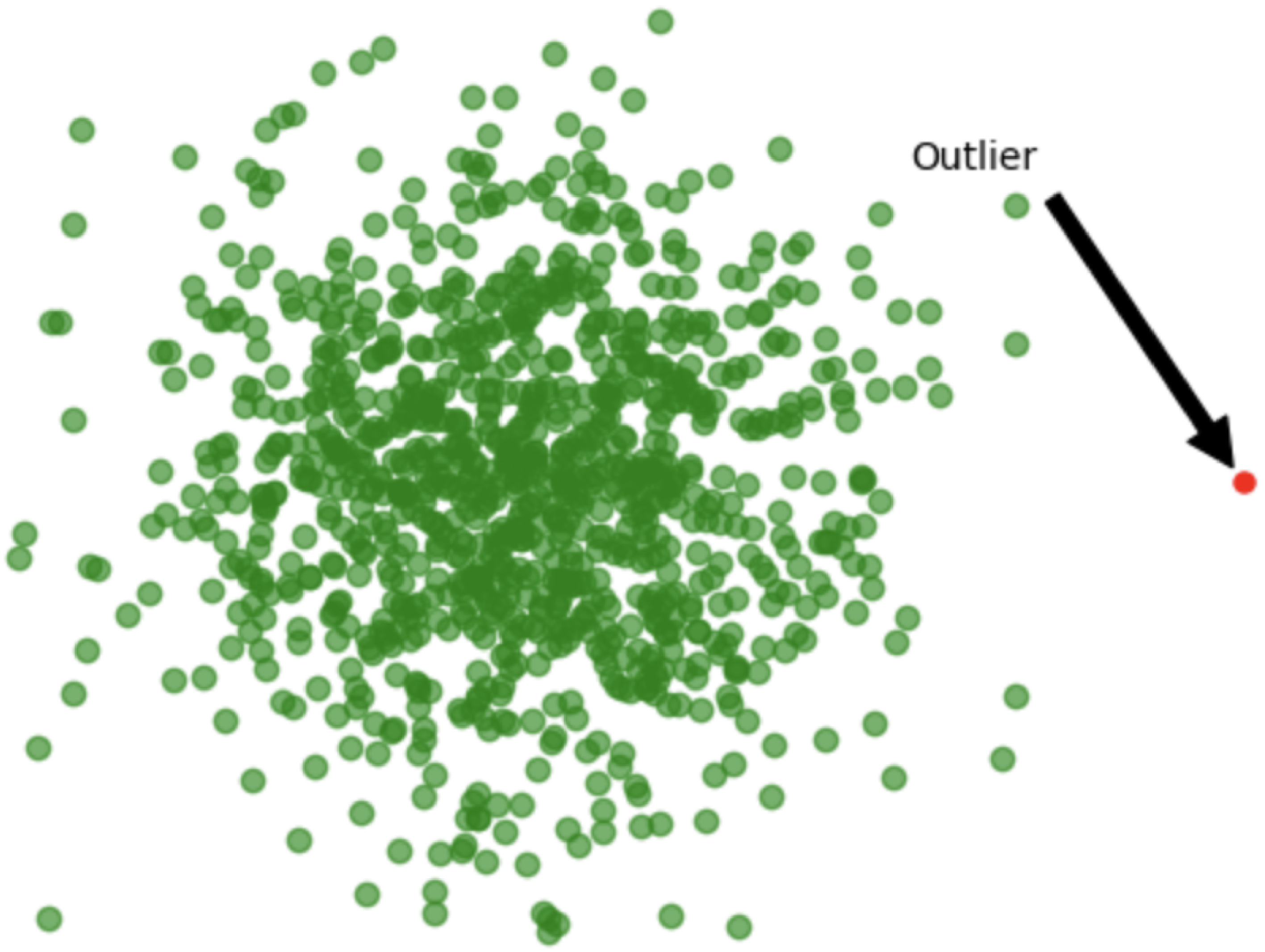
**Usage**

```
# check whether all values in a column are null
print(heart_disease_df['oldpeak'].isnull().all())
```

```
True
```

# Outliers

- Anomalous values
  - Measurement errors
  - Data entry errors
  - Rare events
- Can skew model performance
  - Model learns based on extreme values
  - Doesn't capture general data trend
- Sometimes can be useful:
  - Rare values
  - Detection: use boxplot, or IQR



# Visualizing our data

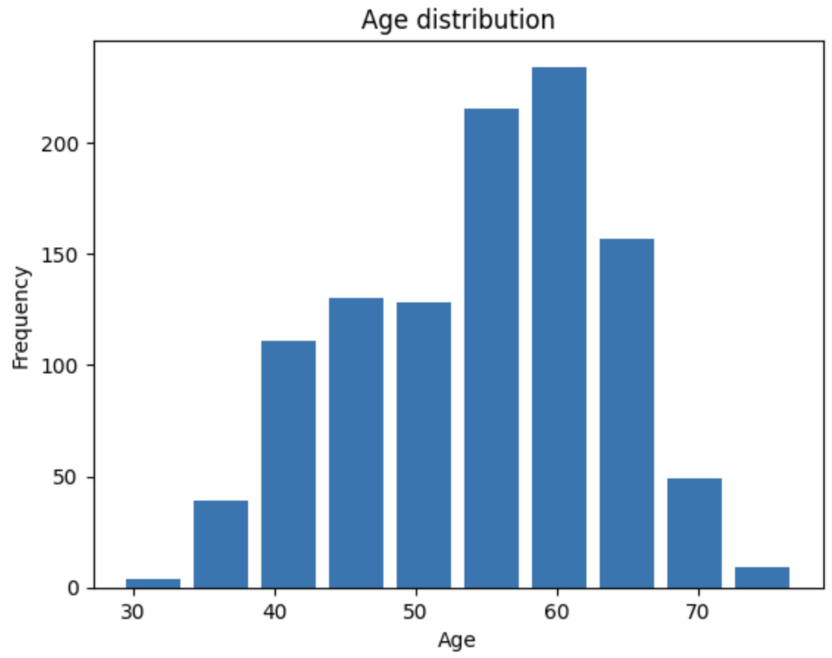
Visualizations show:

- General trends
- Missing values and outliers

Other types of visualizations:

- Kernel density estimation
- Empirical cumulative distributions
- Bivariate distributions

```
df['age'].plot(kind='hist')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.show()
```




[1] https://seaborn.pydata.org/tutorial/distributions.html, https://app.datacamp.com/learn/courses/intermediate-data-visualization-with-seaborn

# Goals of EDA

**Understand the data**

- Are there any patterns?
- Eg: do men have higher rate of heart disease?

**Detect outliers**

- Does any data fall outside what is acceptable?
- Are there incorrect or missing values?

**Formulate hypotheses**

- What should we expect from the data?

**Check assumptions**

- Does what we expect line up with reality?

# Let's practice!

END-TO-END MACHINE LEARNING

# Data preparation

END-TO-END MACHINE LEARNING

**Joshua Stapleton**
Machine Learning Engineer

# Data preparation steps

Dataset has:

- Missing values
- Outliers
- Imbalances
- Empty columns
- Duplicates

Data preparation:

- Based on insights from EDA
- Critical for model performance downstream

# Null / empty values

- Drop missing or sparse rows/columns
- Null values can break model
- Use `df.drop()` for columns
- Use `df.dropna(how='all')` for rows

```
# count missing values
print(df['oldpeak'].isnull().sum())

# Drop empty column(s) and row(s)
columns_dropped = heart_disease_df.drop(['oldpeak'], axis='columns')
rows_and_columns_dropped = columns_dropped.dropna(how='all')
```

# Dealing with null / empty values

- Data cleaning / dropping values depends on EDA findings

- If given column has too many missing values:
  - Drop column

- If target column has missing values:
  - Drop rows with missing targets
  - Or treat as separate category

# Imputation

What to do when there are only a few missing values?

- Imputation:
  - Fill missing values with substitutes
- Strategies
  - Fill with mean or median
  - Use constant or previous value

```
# Calculate the mean cholestrol value
mean_value = heart_disease_df['chol'].mean()

# Fill missing cholestrol values with the mean
heart_disease_df['chol'].fillna(mean_value, inplace=True)
```

# Advanced imputation

Advanced techniques:

- K-nearest neighbors
- SMOTE (synthetic minority oversampling technique)

```
from sklearn.impute import KNNImputer

# Initialize KNNImputer
imputer = KNNImputer(n_neighbors=2, weights="uniform")

# Perform the imputation on your DataFrame
df_imputed['oldpeak'] = imputer.fit_transform(df['oldpeak'])
```

# Dropping duplicates

- Data must be clean, concise, and rich
- Redundancies are unhelpful
- Duplicates can bias or confuse model
- Look at unique identifiers as a criteria for dropping records / rows.

```
# Drop duplicate rows
heart_disease_duplicates_dropped = heart_disease_column_dropped.drop_duplicates()
```

# Let's practice!

END-TO-END MACHINE LEARNING