Welcome to the course!

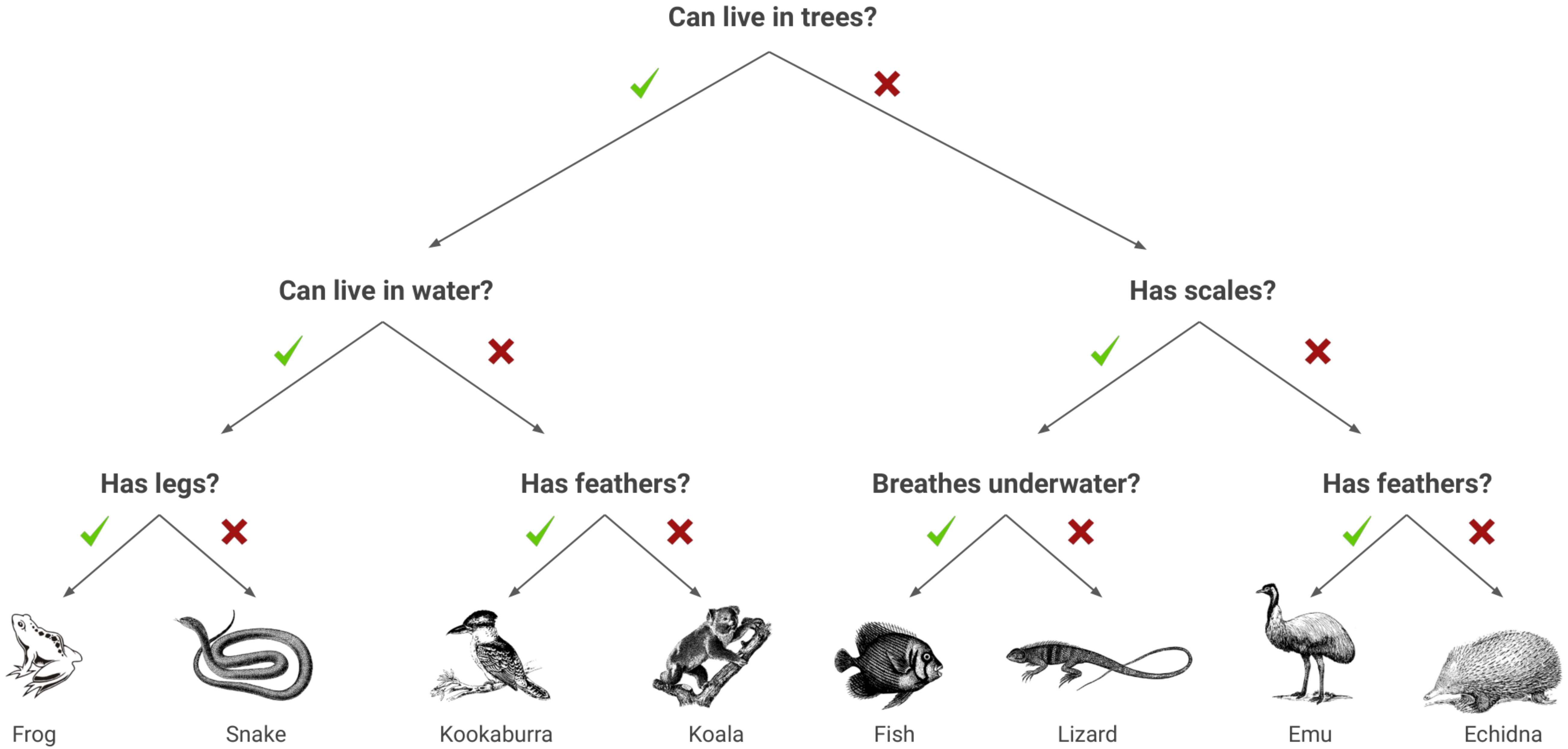
MACHINE LEARNING WITH TREE-BASED MODELS IN R

Sandro Raabe
Data Scientist

Course overview

- **Chapter 1:** Classification trees
- **Chapter 2:** Regression trees, cross-validation, bias-variance tradeoff
- **Chapter 3:** Hyperparameter tuning, bagging, random forests
- **Chapter 4:** Boosted trees

Decision trees are flowcharts



Can live in trees?

Can live in water?

Has scales?

Has legs?

Has feathers?

Breathes underwater?

Has feathers?

Frog

Snake

Kookaburra

Koala

Fish

Lizard

Emu

Echidna

---

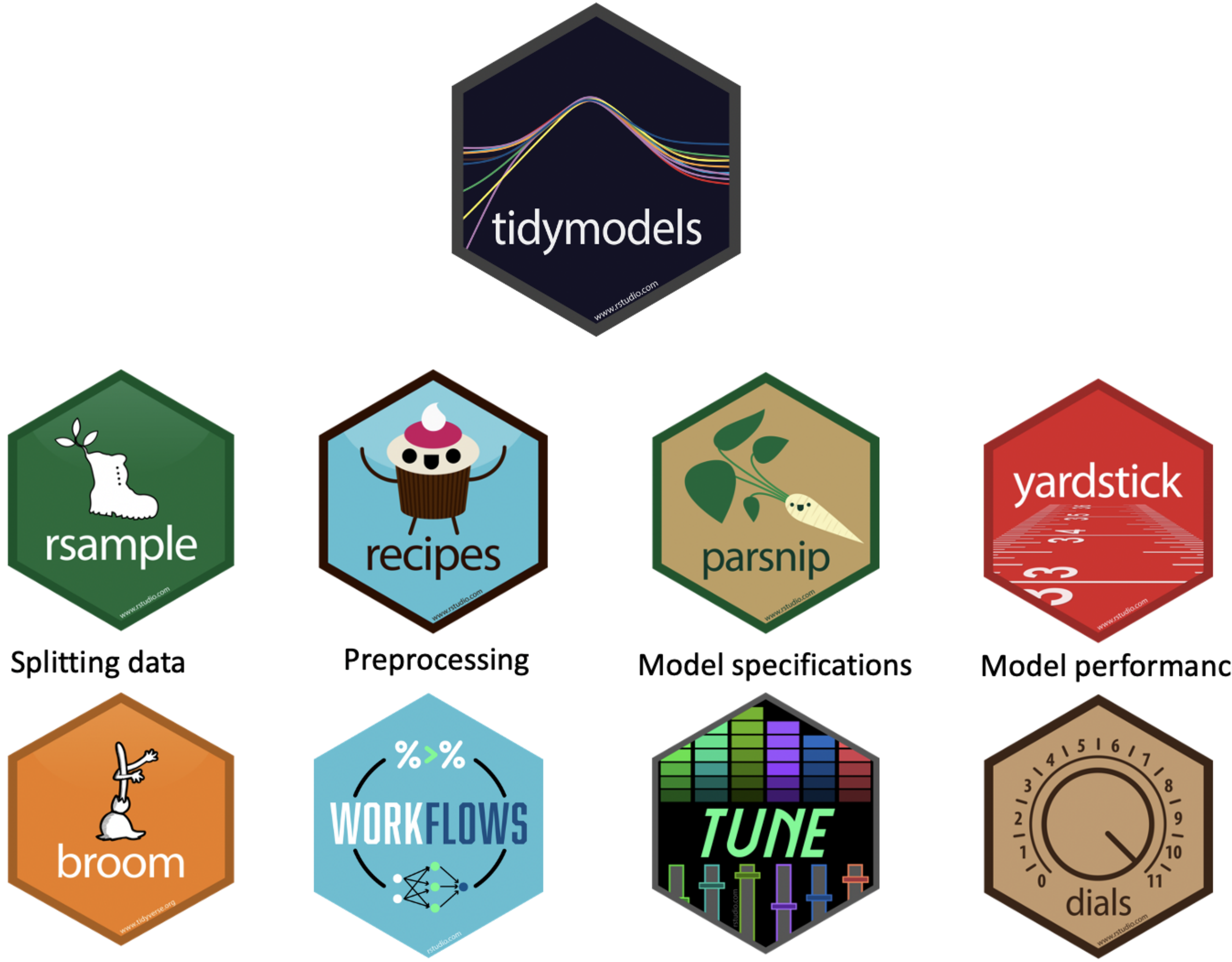
[1] https://aca.edu.au/resources/decision-trees-classifying-animals/decision-trees.pdf

Advantages of tree-based models

- Easy to explain and understand
- Possible to capture non-linear relationships
- Require no normalization or standardization of numeric features
- No need to create dummy indicator variables
- Robust to outliers
- Fast for large datasets

Disadvantages of tree-based models

- Hard to interpret if large, deep, or ensembled
- High variance, complex trees are prone to overfitting



Splitting data

Preprocessing

Model specifications

Model performance

View models and metrics in a tidy way

Make modeling workflow

Tune hyperparameters and get performance metrics

Tune hyperparameters

The tidymodels package

```
library(tidymodels)
```

```
-- Attaching packages -------------------- tidymodels 0.1.4 --
v parsnip   0.2.1      v rsample   0.1.1
v dplyr     1.0.9      v tibble    3.1.7
v yardstick 0.0.9      v tune      0.1.6
```

Create a decision tree

Specification / functional design

1. Pick a model class

```
library(tidymodels)

decision_tree()
```

```
Decision Tree Model Specification (unknown)
```

Create a decision tree

2. Set the engine that powers your model

```r
library(tidymodels)

decision_tree() %>%
    set_engine("rpart")
```

```
Decision Tree Model Specification (unknown)

Computational engine: rpart
```

Create a decision tree

3. Set the mode (classification or regression)

```r
library(tidymodels)

decision_tree() %>%
     set_engine("rpart") %>%
     set_mode("classification")
```

```
Decision Tree Model Specification (classification)

Computational engine: rpart
```

From a model specification to a real model

Specification is a skeleton and needs data to be trained with

```
library(tidymodels)
tree_spec <- decision_tree() %>%
               set_engine("rpart") %>%
               set_mode("classification")
```

```
# A model specification is fit using a formula to training data
tree_spec %>%
  fit(formula = outcome ~ age + bmi,
      data = diabetes)
```

```
parsnip model object
Fit time: 19 ms
n = 652
```

Let's build a model!

MACHINE LEARNING WITH TREE-BASED MODELS IN R

How to grow your tree

MACHINE LEARNING WITH TREE-BASED MODELS IN R

Sandro Raabe
Data Scientist

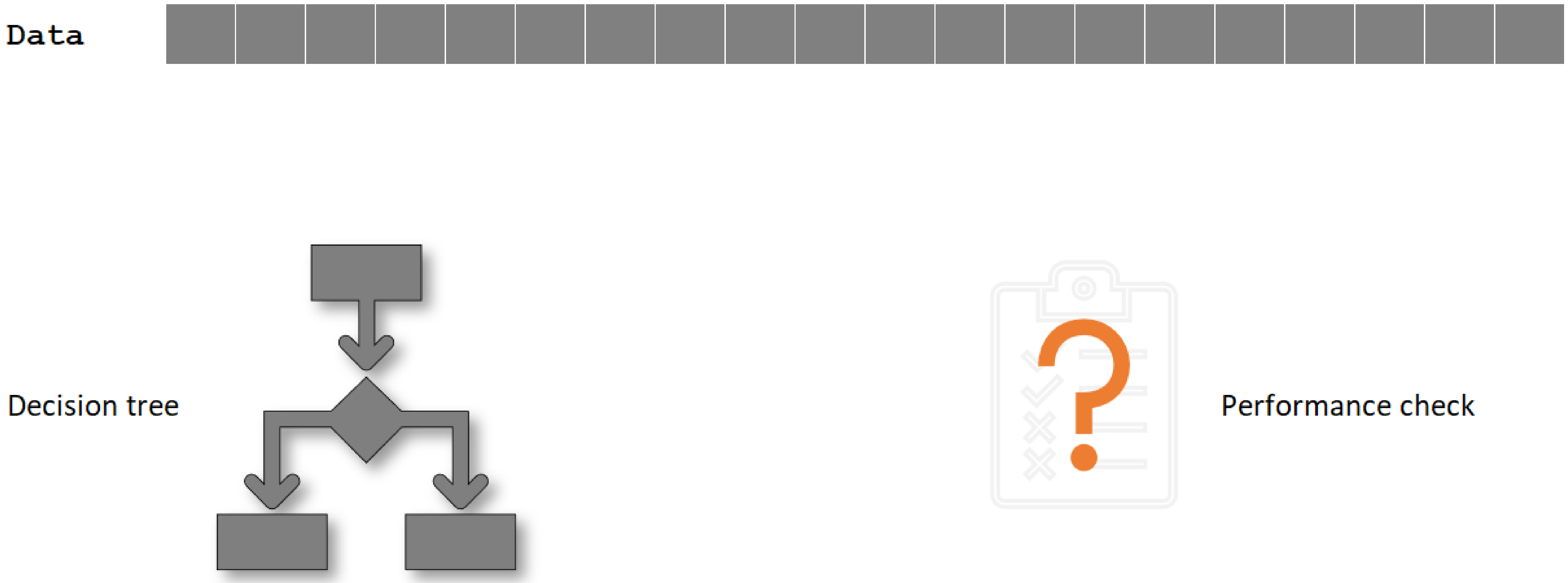
Diabetes dataset

```r
head(diabetes)
```

```
# A tibble: 6 x 9
  outcome pregnancies glucose blood_pressure skin_thickness insulin   bmi    age
  <fct>         <int>   <int>          <int>          <int>   <int> <dbl>  <int>
1 yes               6     148             72             35       0  33.6     50
2 no                1      85             66             29       0  26.6     31
3 yes               8     183             64              0       0  23.3     32
```

Using the whole dataset

- Used all your data for training - no data left to test the model



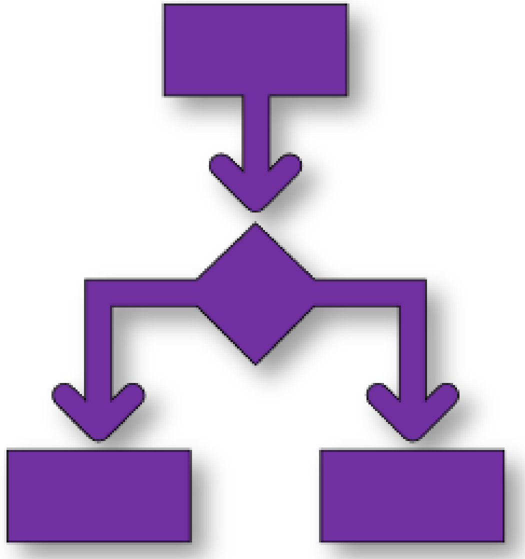
Data split

`Data`

`Training set`

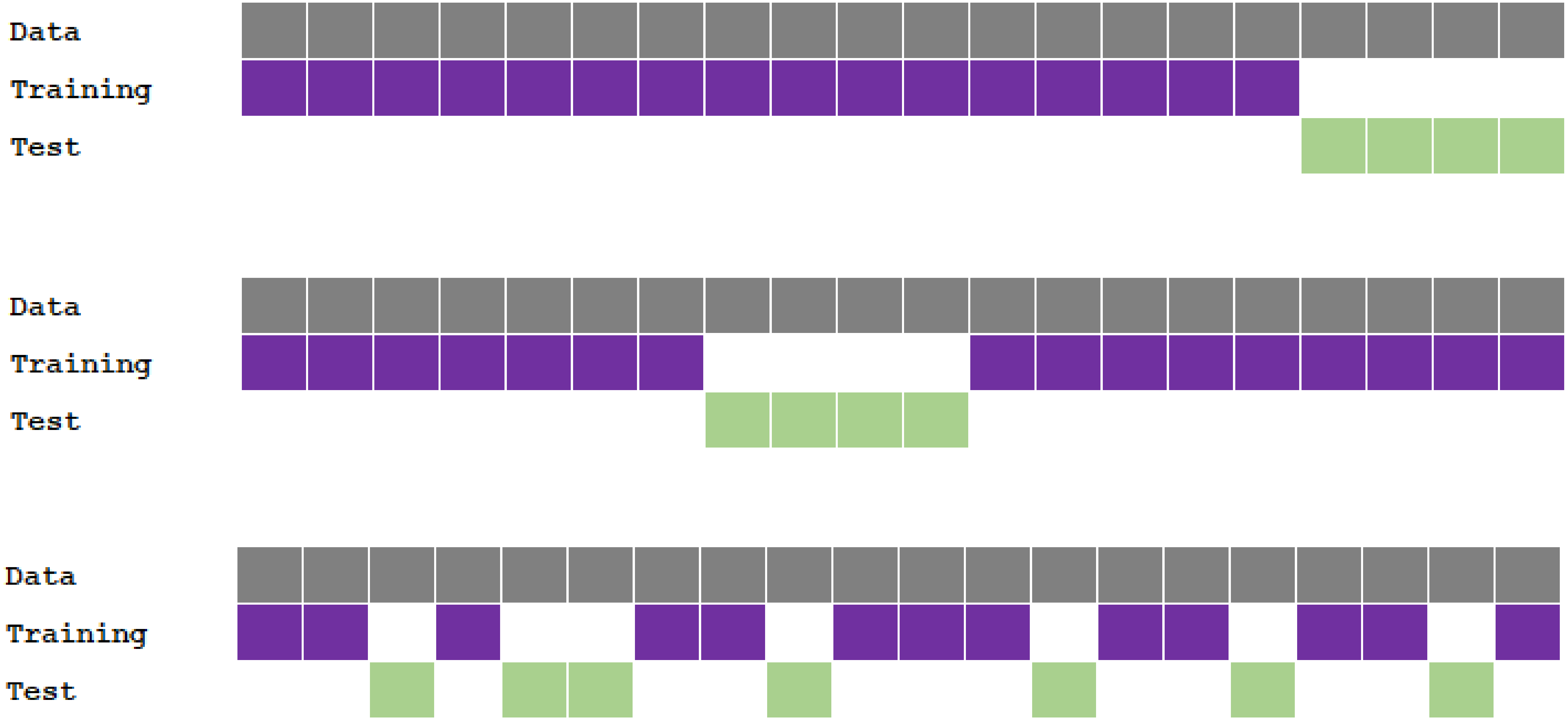
`Test set`

Decision tree



Performance check

Splitting methods



The initial_split() function

- Splits data randomly into single training and single test set

```r
# Split data proportionally (default: 0.75)
diabetes_split <- initial_split(diabetes, prop = 0.9)
diabetes_split
```

```
<Analysis/Assess/Total>
<692/76/768>
```

---

[1] from the rsample package

Functions training() and testing()

- Extract training and test sets from a data split

```
diabetes_train <- training(diabetes_split)
diabetes_test  <- testing(diabetes_split)
```

- Verification:

  ```
  nrow(diabetes_train)/nrow(diabetes)
  ```

```
[1] 0.9007812
```

---

[1] from rsample

Avoid class imbalances

```r
# Training count of 'yes' and 'no' outcomes
counts_train <- table(diabetes_train$outcome)
counts_train
```

```
 no yes
490 86
```

```r
# Test data count of 'yes' and 'no' outcomes
counts_test <- table(diabetes_test$outcome)
counts_test
```

```
no yes
28  48
```

```r
# Training proportion of 'yes' outcome
prop_yes_train <- counts_train["yes"]/
                  sum(counts_train)
prop_yes_train
```

```
0.15
```

```r
# Test data proportion of 'yes' outcome
prop_yes_test <- counts_test["yes"]/
                  sum(counts_test)
prop_yes_test
```

```
0.63
```

Solution - enforce similar distributions

```
initial_split(diabetes,
              prop = 0.9,
              strata = outcome)
```

- Ensures random split with similar distribution of `outcome` variable

Let's split!

MACHINE LEARNING WITH TREE-BASED MODELS IN R

Predict and evaluate

MACHINE LEARNING WITH TREE-BASED MODELS IN R

Sandro Raabe
Data Scientist

Predicting on new data

General call:

```
predict(model, new_data, type)
```

Arguments:

1. Trained model
2. Dataset to predict on
3. Prediction type: labels or probabilities

Predicting on new data

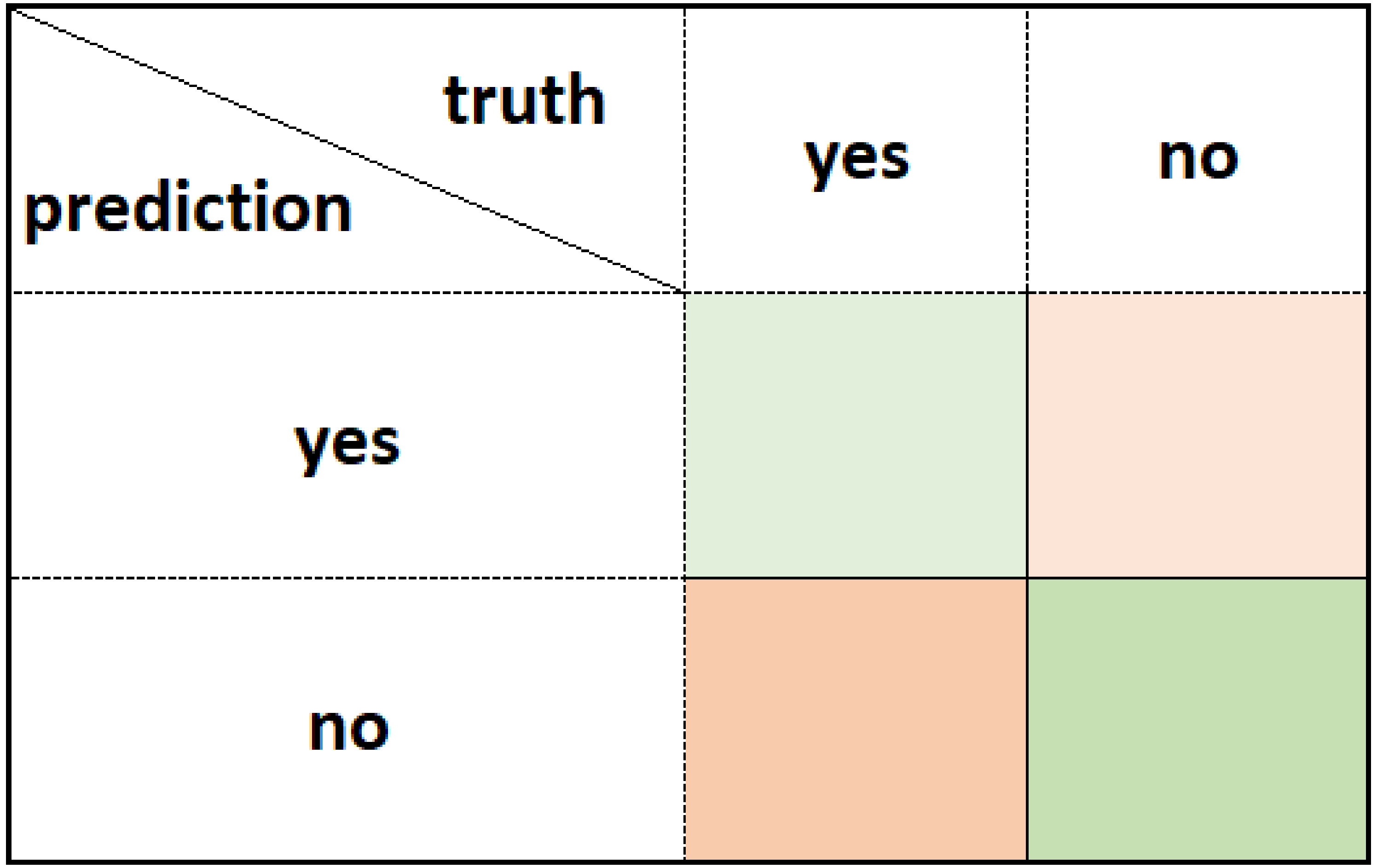
```
predict(model, new_data = test_data,
               type = "class")
```

```
  .pred_class
  <fct>
1 no
2 no
3 yes
4 no
```

```
predict(model, new_data = test_data,
               type = "prob")
```

```
     .pred_no  .pred_yes
     <dbl>     <dbl>
1    0.866     0.134
2    0.956     0.044
3    0.672     0.328
4    0.877     0.123
```

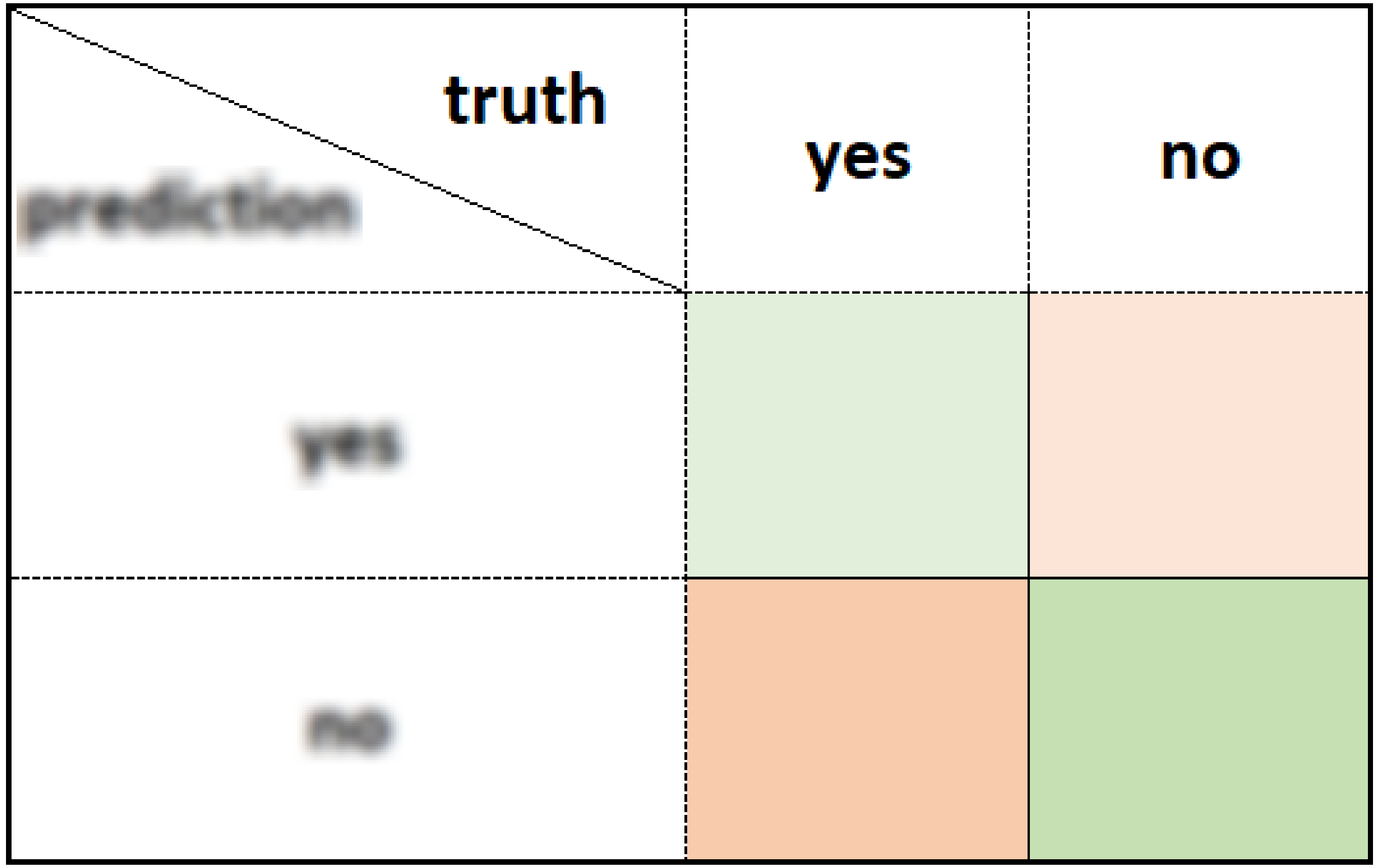
Confusion matrix



truth prediction	yes	no
yes	🟩	🟧
no	🟧	🟩

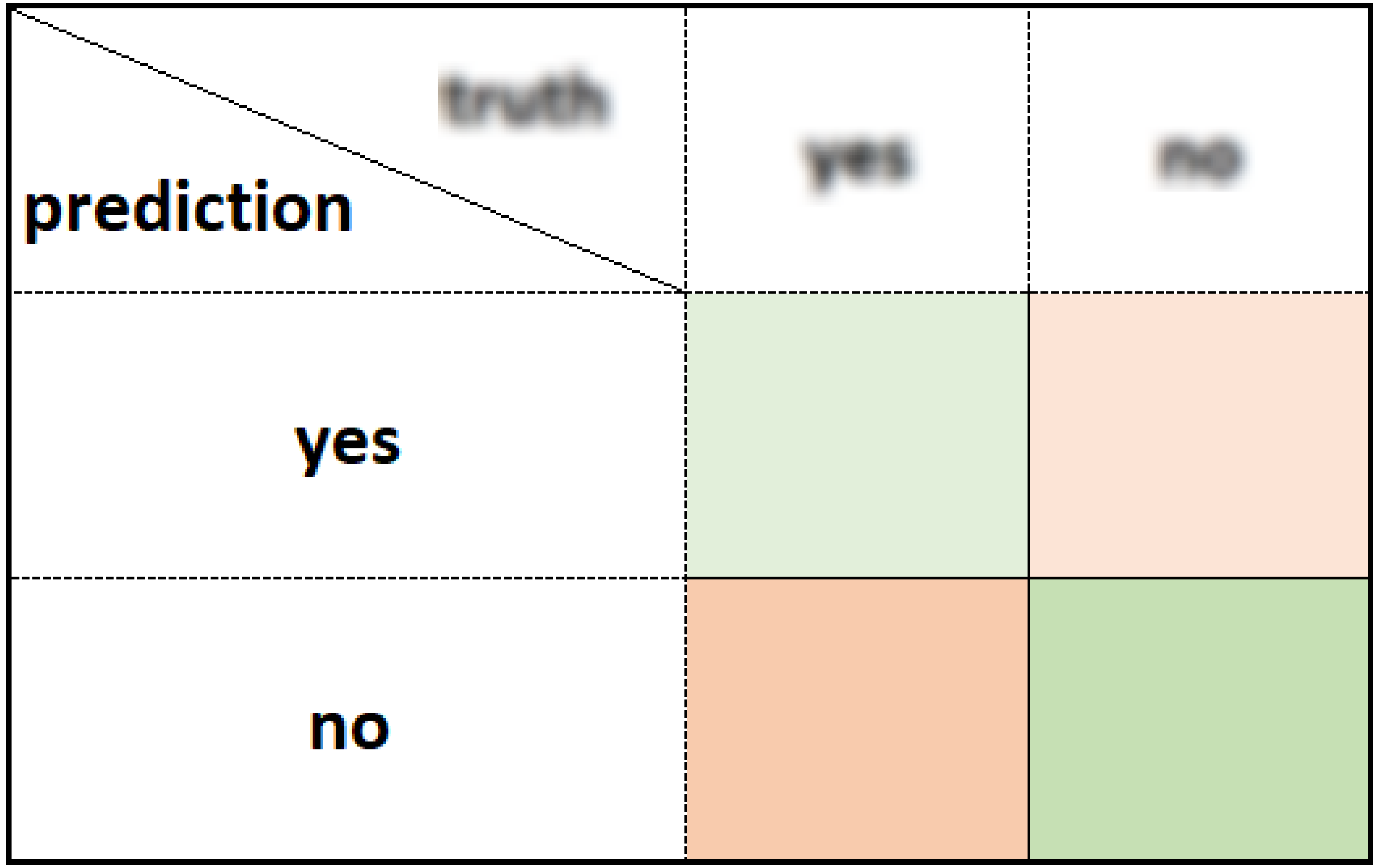
- Reveals how confused a model is

Confusion matrix



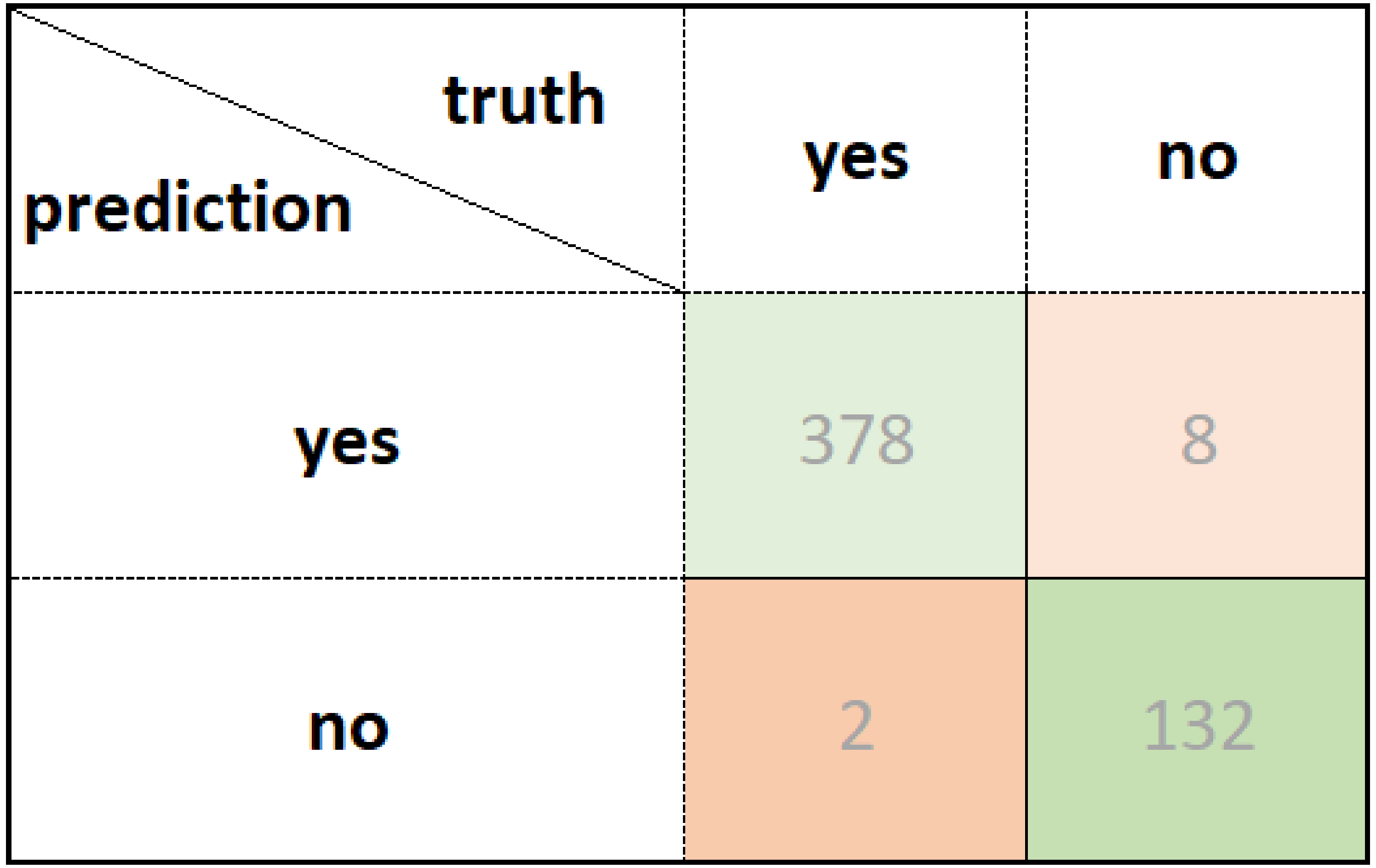
truth prediction	yes	no
yes	🟩	🟧
no	🟧	🟩

Confusion matrix



prediction \ truth	yes	no
yes	🟩	🟧
no	🟧	🟩

Confusion matrix

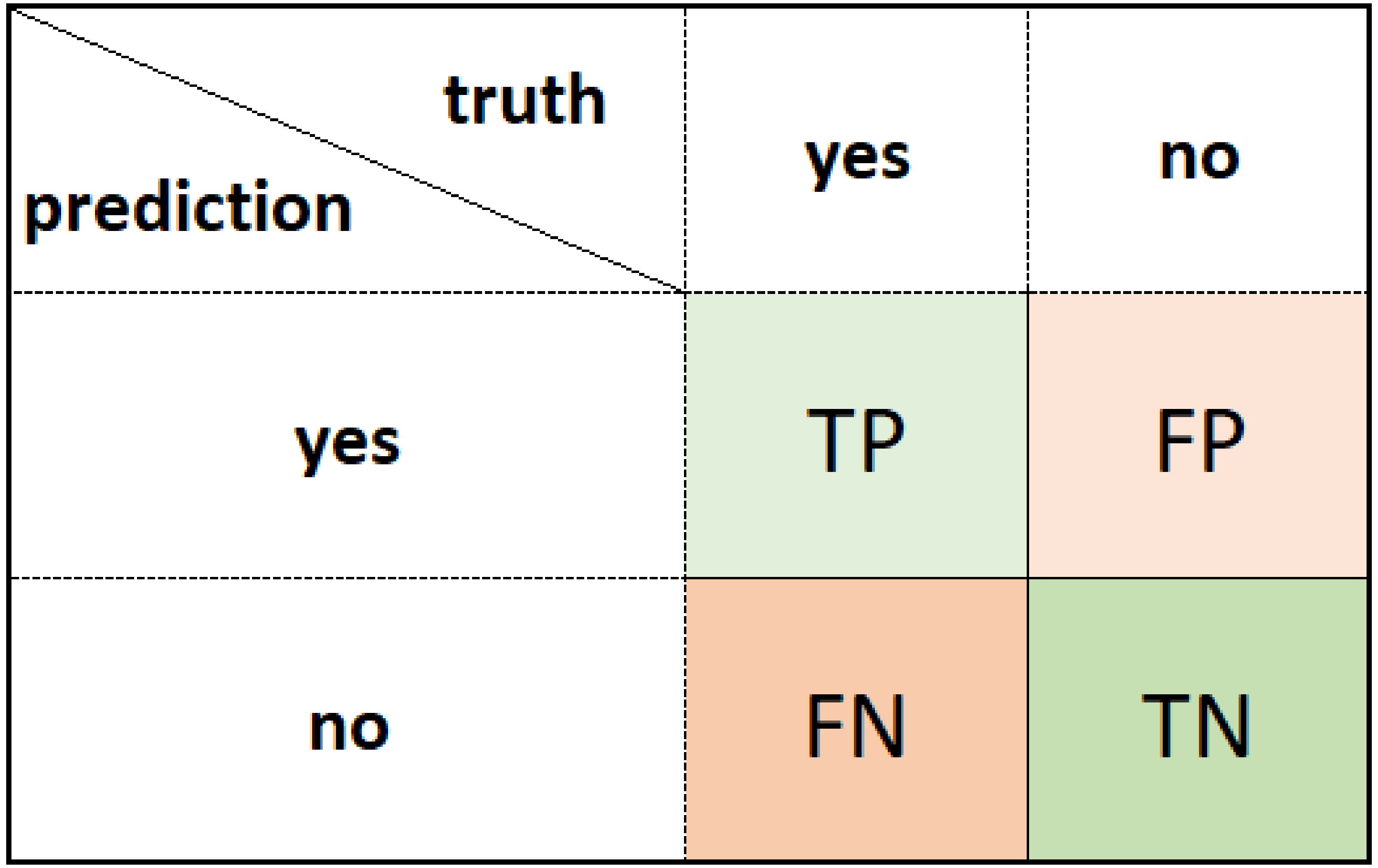


truth prediction	yes	no
yes	378	8
no	2	132

- Diagonal: correct predictions
- Off-diagonal: incorrect predictions

Confusion matrix

- **TP**: prediction is **yes**, truth is **yes**
- **TN**: prediction is **no**, truth is **no**
- **FP**: prediction is **yes**, truth is **no**
- **FN**: prediction is **no**, truth is **yes**



truth prediction	yes	no
yes	TP	FP
no	FN	TN

Create the confusion matrix

```r
# Combine predictions and truth values
pred_combined <- predictions %>%
   mutate(true_class = test_data$outcome)

pred_combined
```

```
  .pred_class  true_class
  <fct>        <fct>
1 no           no
2 no           yes
3 no           no
4 yes          yes
```

```r
# Calculate the confusion matrix
conf_mat(data = pred_combined,
         estimate = .pred_class,
         truth = true_class)
```

```
            Truth
Prediction   no   yes
        no  116    31
       yes   12    40
```

Accuracy

$$\text{accuracy} = \frac{\text{n of correct predictions}}{\text{n of total predictions}}$$

- Function name: `accuracy()`
- Same arguments as `conf_mat()`
    - `data`, `estimate` and `truth`
    - Common structure in `yardstick`

```
accuracy(pred_combined,
         estimate = .pred_class,
         truth = true_class)
```

```
  .metric      .estimate
  <chr>            <dbl>
1 accuracy         0.708
```

Let's evaluate!

MACHINE LEARNING WITH TREE-BASED MODELS IN R