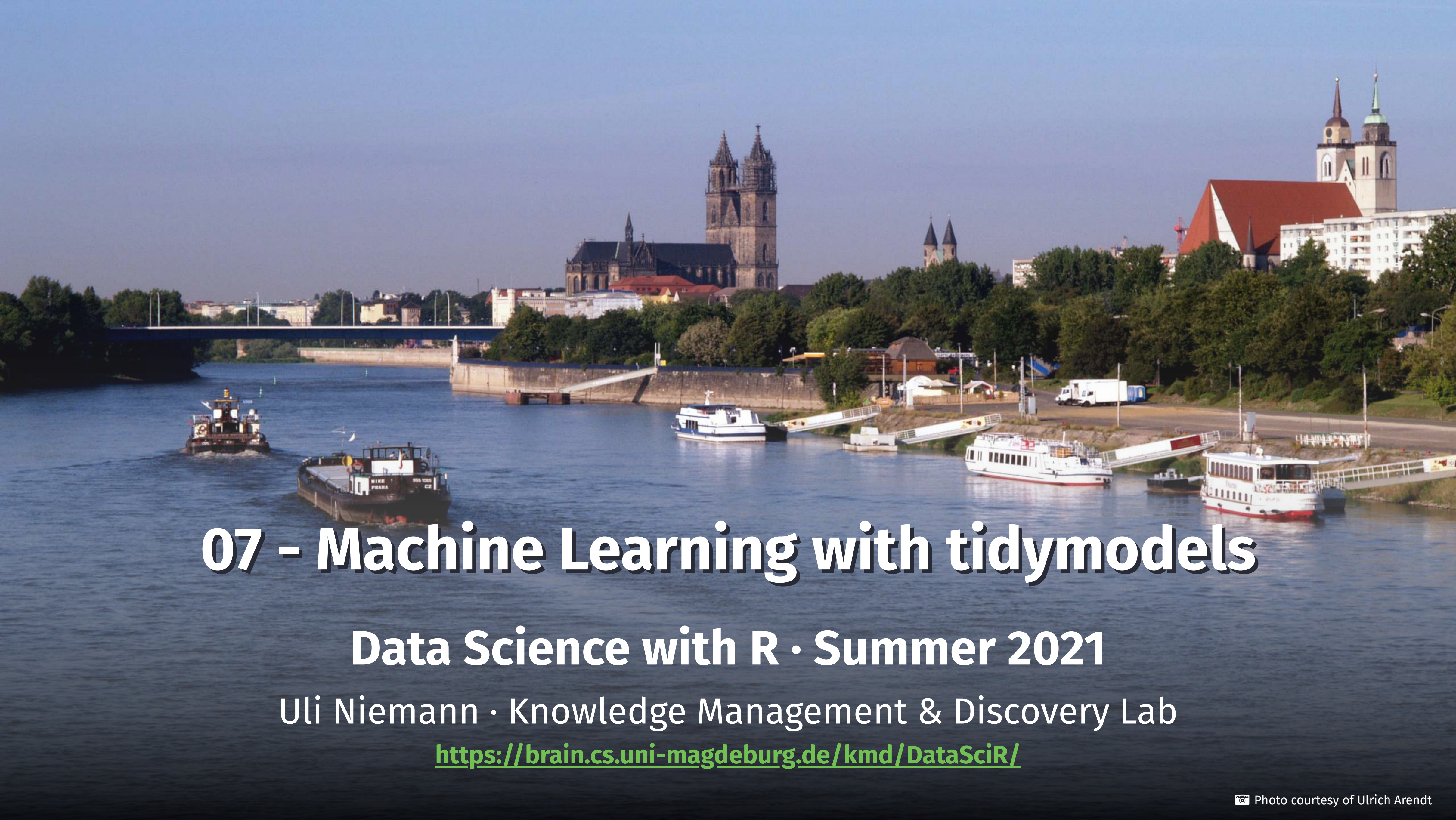


07 – Machine Learning with tidymodels

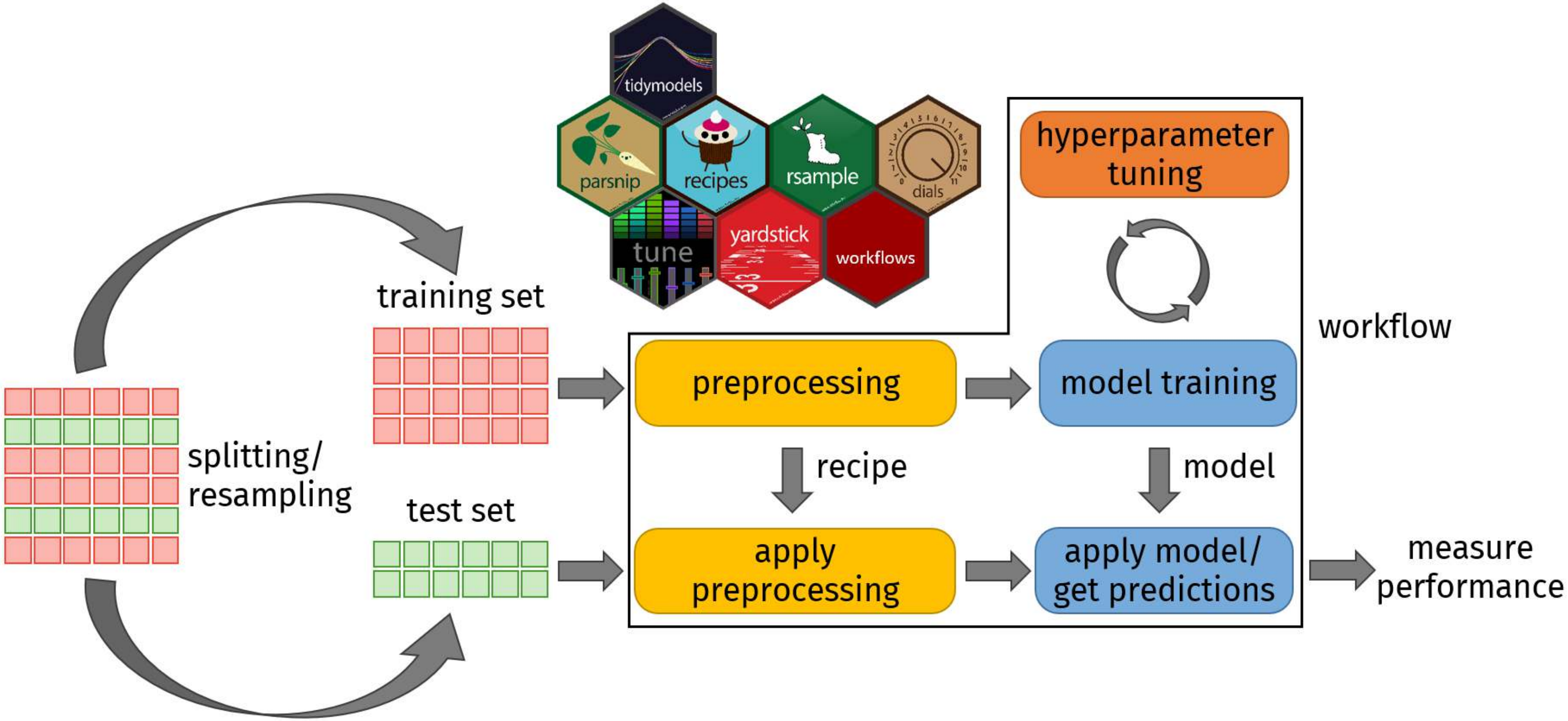
Data Science with R · Summer 2021

Uli Niemann · Knowledge Management & Discovery Lab

https://brain.cs.uni-magdeburg.de/kmd/DataSciR/

Photo courtesy of Ulrich Arendt

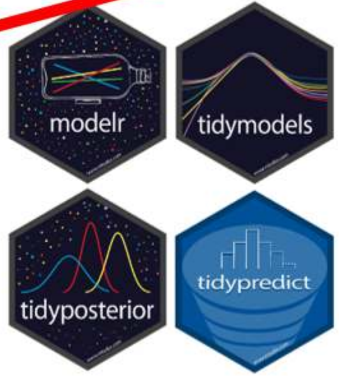
tidymodels


tidymodels
parsnip
recipes
rsample
dials
tune
yardstick
workflows
hyperparameter tuning
training set
splitting/ resampling
test set
preprocessing
model training
workflow
recipe
model
apply preprocessing
apply model/ get predictions
measure performance

Classification in `R`: the `caret` package

`caret`, short for **c**lassification **a**nd **re**gression **t**raining, is a **meta-package** for **predictive modeling** tasks. It builds upon many modeling packages available in `R` and provides a **unified interface** to them, making it easy to experiment with different classification algorithms without worrying about the partially inconsistent syntax across different packages. Furthermore, `caret` provides various additional functions for **data splitting, pre-processing, feature selection, model tuning and feature importance** estimation.

`caret` and the tidyverse



`caret` was developed before the tidyverse, so it doesn't support many of its features, like piping with `%>%`. RStudio currently spends much effort in extending the tidyverse with packages for predictive modeling.

The extensive `caret` documentation, which contains helpful code examples and a list of supported modeling algorithms, is available as free e-book.

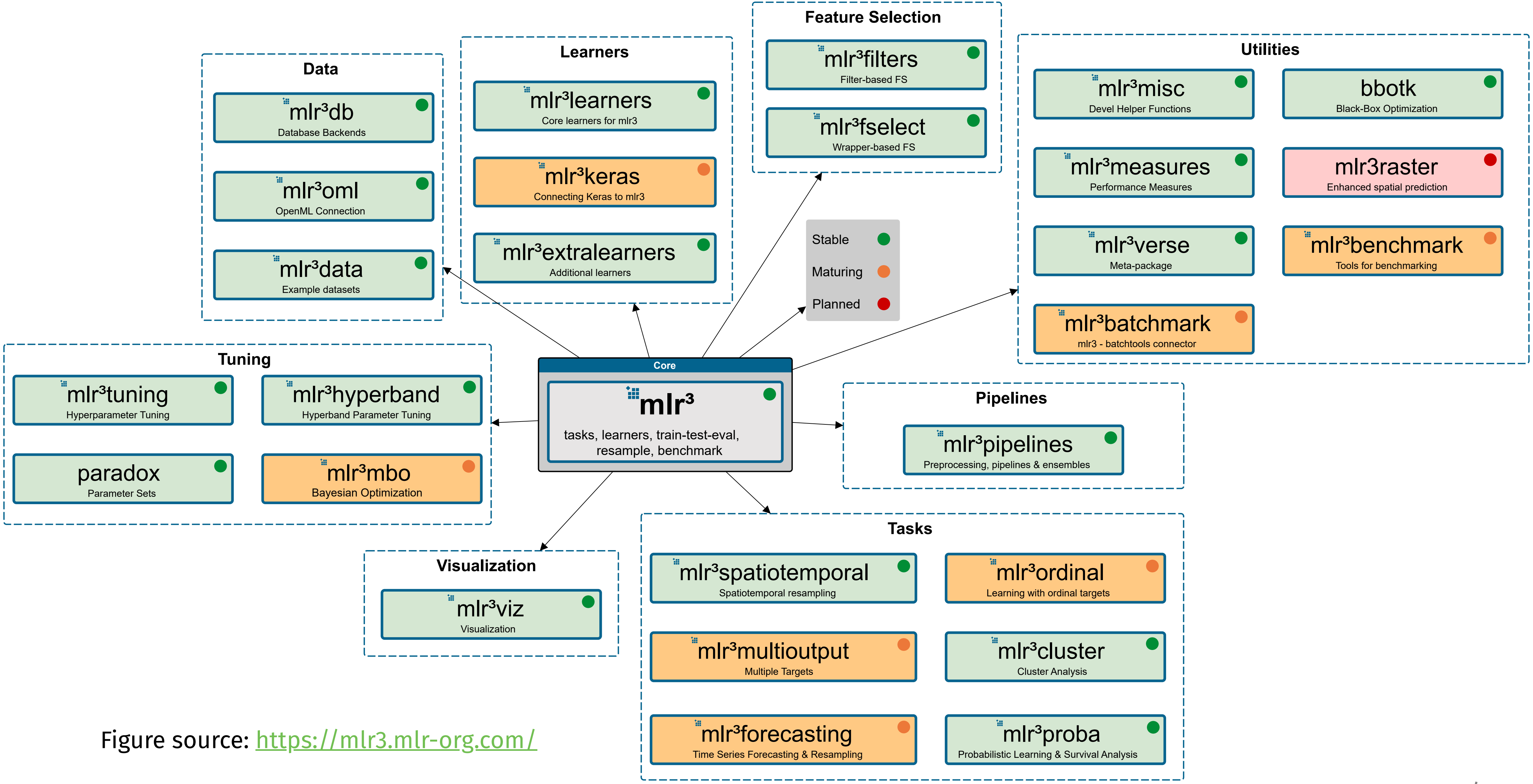


Figure source: https://mlr3.mlr-org.com/

This tutorial is a condensed version of the 2-day workshop "Introduction to Machine Learning with the Tidyverse" held by Dr. Alison Hill at the rstudio::conf 2020.

Setup

```r
library(tidyverse)
library(tidymodels)
```

```
## -- Attaching packages ------------------------------------------------ tidymodels 0.1.2 --

## v broom     0.7.6      v recipes   0.1.15
## v dials     0.0.9      v rsample   0.0.9
## v infer     0.5.4      v tune      0.1.2
## v modeldata 0.1.0      v workflows 0.2.2
## v parsnip   0.1.5      v yardstick 0.0.7

## -- Conflicts --------------------------------------------------- tidymodels_conflicts() --
## x scales::discard()        masks purrr::discard()
## x dplyr::filter()          masks stats::filter()
## x recipes::fixed()         masks stringr::fixed()
## x kableExtra::group_rows() masks dplyr::group_rows()
## x dplyr::lag()             masks stats::lag()
## x yardstick::spec()        masks readr::spec()
## x recipes::step()          masks stats::step()
```

Ames Iowa Housing Dataset

> *"Data set contains information from the Ames Assessor's Office used in computing assessed values for individual residential properties sold in Ames, IA from 2006 to 2010." — [Dataset documentation]()*

De Cock, Dean. "Ames, Iowa: Alternative to the Boston housing data as an end of semester regression project." Journal of Statistics Education 19.3 (2011). [URL]()

```r
library(AmesHousing)
(ames <- make_ames() %>% select(-matches("Qu")))
```

```
## # A tibble: 2,930 x 74
##    MS_SubClass  MS_Zoning Lot_Frontage Lot_Area Street Alley Lot_Shape
##    <fct>        <fct>            <dbl>    <int> <fct>  <fct> <fct>
##  1 One_Story_1~ Resident~          141    31770 Pave   No_A~ Slightly~
##  2 One_Story_1~ Resident~           80    11622 Pave   No_A~ Regular
##  3 One_Story_1~ Resident~           81    14267 Pave   No_A~ Slightly~
##  4 One_Story_1~ Resident~           93    11160 Pave   No_A~ Regular
##  5 Two_Story_1~ Resident~           74    13830 Pave   No_A~ Slightly~
##  6 Two_Story_1~ Resident~           78     9978 Pave   No_A~ Slightly~
##  7 One_Story_P~ Resident~           41     4920 Pave   No_A~ Regular
##  8 One_Story_P~ Resident~           43     5005 Pave   No_A~ Slightly~
##  9 One_Story_P~ Resident~           39     5389 Pave   No_A~ Slightly~
## 10 Two_Story_1~ Resident~           60     7500 Pave   No_A~ Regular
## # ... with 2,920 more rows, and 67 more variables:
## #   Land_Contour <fct>, Utilities <fct>, Lot_Config <fct>,
## #   Land_Slope <fct>, Neighborhood <fct>, Condition_1 <fct>,
## #   Condition_2 <fct>, Bldg_Type <fct>, House_Style <fct>,
## #   Overall_Cond <fct>, Year_Built <int>, Year_Remod_Add <int>,
## #   Roof_Style <fct>, Roof_Matl <fct>, Exterior_1st <fct>,
## #   Exterior_2nd <fct>, Mas_Vnr_Type <fct>, Mas_Vnr_Area <dbl>,
## #   Exter_Cond <fct>, Foundation <fct>, Bsmt_Cond <fct>,
## #   Bsmt_Exposure <fct>, BsmtFin_Type_1 <fct>, BsmtFin_SF_1 <dbl>,
## #   BsmtFin_Type_2 <fct>, BsmtFin_SF_2 <dbl>, Bsmt_Unf_SF <dbl>,
## #   Total_Bsmt_SF <dbl>, Heating <fct>, Heating_QC <fct>,
## #   Central_Air <fct>, Electrical <fct>, First_Flr_SF <int>,
## #   Second_Flr_SF <int>, Gr_Liv_Area <int>, Bsmt_Full_Bath <dbl>,
```

Specify a model with parsnip

Specify a model with `parsnip`

1. Pick a **model**
2. Set the **engine**
3. Set the **mode** (if needed)

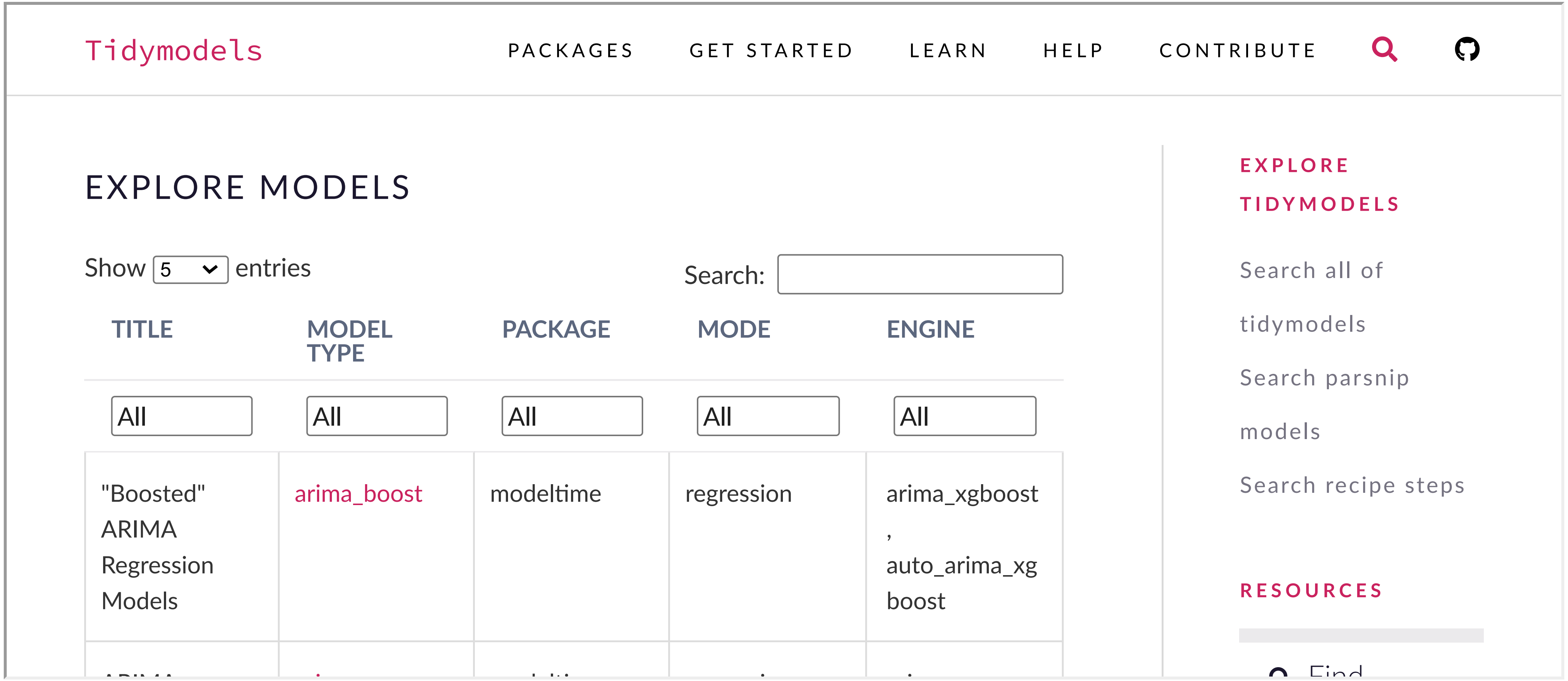
```r
decision_tree() %>% # model
  set_engine("rpart") %>% # engine
  set_mode("classification") # mode
```

```
## Decision Tree Model Specification (classification)
##
## Computational engine: rpart
```

```r
nearest_neighbor() %>%
  set_engine("kknn") %>%
  set_mode("regression")
```

```
## K-Nearest Neighbor Model Specification (regression)
##
## Computational engine: kknn
```

All available models are listed at https://www.tidymodels.org/find/parsnip/#models.



1. Pick a **model**

2. Set the **engine**

3. Set the **mode**

`linear_reg()`

Specify a model that uses linear regression:

```
linear_reg(
  mode = "regression", # type of model (only "regression" here)
  penalty = NULL, # amount of regularization
  mixture = NULL # proportion of L1 regularization
)
```

1. Pick a **model**

2. Set the **engine**

3. Set the **mode**

`set_engine()`

Add an engine to power or implement the model:

```
linear_reg() %>%
  set_engine(engine = "lm", ...)
```

Available engines for `linear_reg()`:

- R: "lm" (the default) or "glmnet"
- Stan: "stan"
- Spark: "spark"
- keras: "keras"

1. Pick a **model**

2. Set the **engine**

3. Set the **mode**

`set_mode()`

Set the model type, either `"regression"` or `"classification"`. Not necessary if mode is set in Step 1.

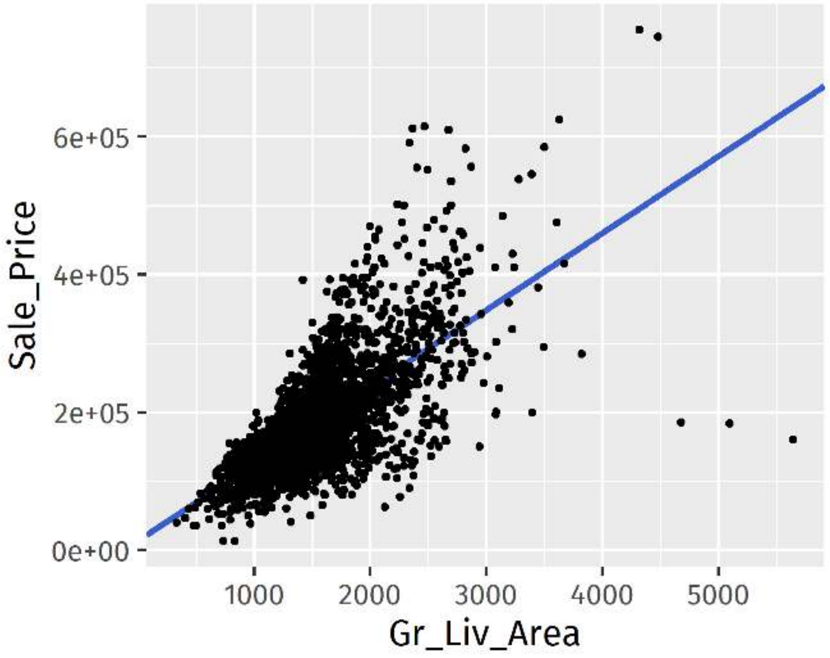
```
linear_reg() %>%
  set_engine(engine = "lm") %>%
  set_mode(mode = "regression")
```

`fit()`

`fit()`: fit a simple linear regression model to predict *sale price* based on *above ground living area*.

```
lm_spec <- linear_reg() %>%
  set_engine(engine = "lm") %>%
  set_mode(mode = "regression")
m <- fit(
  lm_spec, # parsnip model spec
  Sale_Price ~ Gr_Liv_Area, # formula
  ames # data frame
)
m
```

```
## parsnip model object
##
## Fit time:  10ms
##
## Call:
## stats::lm(formula = Sale_Price ~ Gr_Liv_Area, data = data)
##
## Coefficients:
## (Intercept)  Gr_Liv_Area
##     13289.6        111.7
```

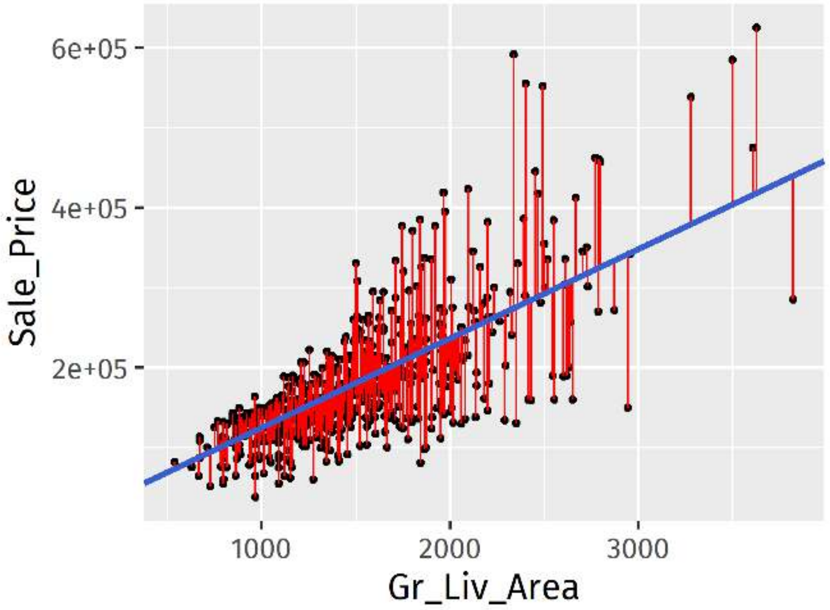


`predict()`

`predict()`: use a fitted model to predict new response values from data. Returns a tibble.

```
p <- predict(m, new_data = ames)
p
```

```
## # A tibble: 2,930 x 1
##      .pred
##      <dbl>
##  1 198255.
##  2 113367.
##  3 161731.
##  4 248964.
##  5 195239.
##  6 192447.
##  7 162736.
##  8 156258.
##  9 193787.
## 10 214786.
## # ... with 2,920 more rows
```



(Only a random sample of 20% is shown.)

Measure model performance with yardstick

Measure the model performance with `yardstick::rmse()`

- **Residuals.** The difference between observed and predicted values: $\hat{y}_i - y_i$.
- **Mean Absolute Error.** $\frac{1}{n}\sum_{i=1}^{n}|\hat{y}_i - y_i|$.
- **Root Mean Squared Error.** $\sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}$.

Calculate the RMSE based on two columns in a data frame:

- truth y_i
- predicted estimate $\hat{y}$

```
lm_spec <- linear_reg() %>%
  set_engine(engine = "lm") %>%
  set_mode(mode = "regression")
lm_fit <- fit(object = lm_spec, formula = Sale_Price ~ Gr_Liv_Area, data = ames)
price_pred <- lm_fit %>%
  predict(new_data = ames) %>%
  mutate(truth = ames$Sale_Price)

rmse(price_pred, truth = truth, estimate = .pred)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard      56505.
```

Available metrics in yardstick

https://yardstick.tidymodels.org/articles/metric-types.html#metrics

Metrics

Below is a table of all of the metrics available in `yardstick`, grouped by type.

type	metric
class	`accuracy()`
class	`bal_accuracy()`
class	`detection_prevalence()`
class	`f_meas()`
class	`j_index()`
class	`kap()`

Contents

Perform resampling with rsample

`initial_split()`

`initial_split()`: partition data randomly into a single training and a single test set.

```
set.seed(123)
(ames_split <- initial_split(ames, prop = 3/4)) # prop = proportion of training instances
```

```
## <Analysis/Assess/Total>
## <2198/732/2930>
```

`training()` and `testing()`

Extract training and testing sets from an `rsplit` object:

```r
training(ames_split)
```

```
## # A tibble: 2,198 x 74
##    MS_SubClass      MS_Zoning    Lot_Frontage
##    <fct>            <fct>               <dbl>
##  1 One_Story_1946_~ Residential~          141
##  2 One_Story_1946_~ Residential~           80
##  3 One_Story_1946_~ Residential~           81
##  4 One_Story_1946_~ Residential~           93
##  5 Two_Story_1946_~ Residential~           74
##  6 Two_Story_1946_~ Residential~           78
##  7 One_Story_PUD_1~ Residential~           41
##  8 Two_Story_1946_~ Residential~           75
##  9 One_Story_1946_~ Residential~            0
## 10 One_Story_1946_~ Residential~           85
## # ... with 2,188 more rows, and 71 more
## #   variables: Lot_Area <int>, Street <fct>,
## #   Alley <fct>, Lot_Shape <fct>,
## #   Land_Contour <fct>, Utilities <fct>,
## #   Lot_Config <fct>, Land_Slope <fct>,
## #   Neighborhood <fct>, Condition_1 <fct>,
## #   Condition_2 <fct>, Bldg_Type <fct>,
## #   House_Style <fct>, Overall_Cond <fct>,
## #   Year_Built <int>, Year_Remod_Add <int>,
## #   Roof_Style <fct>, Roof_Matl <fct>,
## #   Exterior_1st <fct>, Exterior_2nd <fct>,
## #   Mas_Vnr_Type <fct>, Mas_Vnr_Area <dbl>,
```

```r
testing(ames_split)
```

```
## # A tibble: 732 x 74
##    MS_SubClass      MS_Zoning    Lot_Frontage
##    <fct>            <fct>               <dbl>
##  1 One_Story_PUD_1~ Residential~           43
##  2 One_Story_PUD_1~ Residential~           39
##  3 Two_Story_1946_~ Residential~           60
##  4 Two_Story_1946_~ Residential~           63
##  5 Two_Story_1946_~ Residential~           47
##  6 One_Story_1946_~ Residential~           88
##  7 One_Story_1946_~ Residential~            0
##  8 Two_Story_PUD_1~ Residential~           21
##  9 One_Story_1946_~ Residential~           95
## 10 One_Story_1946_~ Residential~           70
## # ... with 722 more rows, and 71 more
## #   variables: Lot_Area <int>, Street <fct>,
## #   Alley <fct>, Lot_Shape <fct>,
## #   Land_Contour <fct>, Utilities <fct>,
## #   Lot_Config <fct>, Land_Slope <fct>,
## #   Neighborhood <fct>, Condition_1 <fct>,
## #   Condition_2 <fct>, Bldg_Type <fct>,
## #   House_Style <fct>, Overall_Cond <fct>,
## #   Year_Built <int>, Year_Remod_Add <int>,
## #   Roof_Style <fct>, Roof_Matl <fct>,
## #   Exterior_1st <fct>, Exterior_2nd <fct>,
## #   Mas_Vnr_Type <fct>, Mas_Vnr_Area <dbl>,
```

Stratified sampling

```
initial_split(ames, strata = Sale_Price, breaks = 6)
```

Cross-validation with `vfold_cv()`

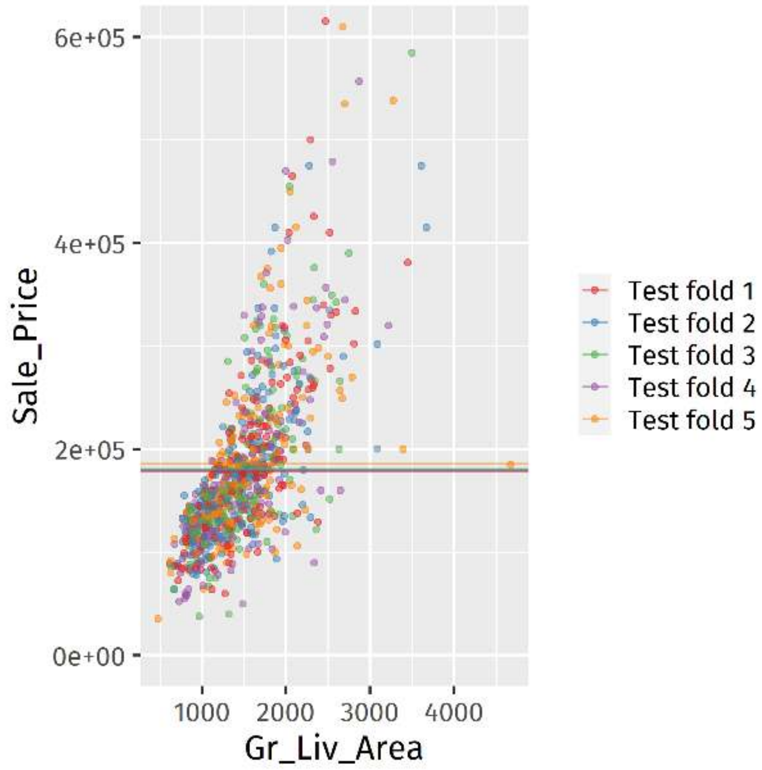
General syntax:

```
vfold_cv(data, v = 10, repeats = 1, strata = NULL, breaks = 4, ...)
```

Example: 10-fold CV on ames data:

```
set.seed(123)
(folds <- vfold_cv(ames, v = 5))
```

```
## #  5-fold cross-validation
## # A tibble: 5 x 2
##   splits             id
##   <list>             <chr>
## 1 <split [2344/586]> Fold1
## 2 <split [2344/586]> Fold2
## 3 <split [2344/586]> Fold3
## 4 <split [2344/586]> Fold4
## 5 <split [2344/586]> Fold5
```



Check whether mean y is approx. equal in each training fold:

```
map_dbl(folds$splits, ~mean(.x$data$Sale_Price[.x$in_id]))
```

```
## [1] 181310.8 180991.0 180840.0 181268.6
## [5] 179569.9
```

Calculate the model performance on multiple resamples with `fit_resamples()`

```
res <- fit_resamples(lm_spec, Sale_Price ~ Gr_Liv_Area, resamples = folds)
res
```

```
## # Resampling results
## # 5-fold cross-validation
## # A tibble: 5 x 4
##   splits             id    .metrics             .notes
##   <list>             <chr> <list>               <list>
## 1 <split [2344/586]> Fold1 <tibble[,4] [2 x 4]> <tibble[,1] [0 x 1]>
## 2 <split [2344/586]> Fold2 <tibble[,4] [2 x 4]> <tibble[,1] [0 x 1]>
## 3 <split [2344/586]> Fold3 <tibble[,4] [2 x 4]> <tibble[,1] [0 x 1]>
## 4 <split [2344/586]> Fold4 <tibble[,4] [2 x 4]> <tibble[,1] [0 x 1]>
## 5 <split [2344/586]> Fold5 <tibble[,4] [2 x 4]> <tibble[,1] [0 x 1]>
```

Collapse performance results across resamples with `collect_metrics()`

```r
res %>% collect_metrics()
```

```
## # A tibble: 2 x 6
##   .metric .estimator      mean     n   std_err .config
##   <chr>   <chr>          <dbl> <int>     <dbl> <chr>
## 1 rmse    standard   56486.        5 1866.     Preprocessor1_Model1
## 2 rsq     standard       0.504     5    0.0193 Preprocessor1_Model1
```

```r
res %>% collect_metrics(summarize = FALSE)
```

```
## # A tibble: 10 x 5
##    id    .metric .estimator .estimate .config
##    <chr> <chr>   <chr>          <dbl> <chr>
##  1 Fold1 rmse    standard   51064.    Preprocessor1_Model1
##  2 Fold1 rsq     standard       0.542 Preprocessor1_Model1
##  3 Fold2 rmse    standard   57206.    Preprocessor1_Model1
##  4 Fold2 rsq     standard       0.464 Preprocessor1_Model1
##  5 Fold3 rmse    standard   53526.    Preprocessor1_Model1
##  6 Fold3 rsq     standard       0.557 Preprocessor1_Model1
##  7 Fold4 rmse    standard   61210.    Preprocessor1_Model1
##  8 Fold4 rsq     standard       0.468 Preprocessor1_Model1
##  9 Fold5 rmse    standard   59422.    Preprocessor1_Model1
## 10 Fold5 rsq     standard       0.488 Preprocessor1_Model1
```

`metric_set()`

`metric_set()`: a helper function for selecting yardstick metric functions.

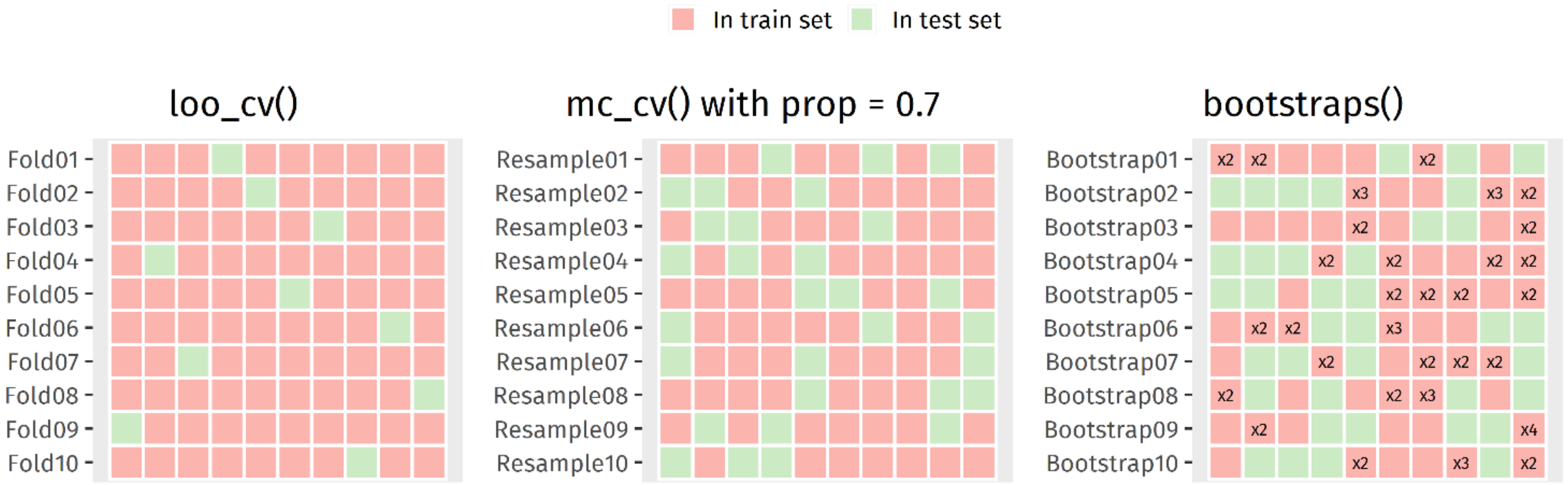
```
fit_resamples(
  object,
  resamples,
  ...,
  metrics = metric_set(rmse, rsq),
  control = control_resamples()
)
```

If `metrics = NULL`:

- regression: `metric_set(rmse, rsq)`
- classification: `metric_set(accuracy, roc_auc)`

Other resampling methods

- `loo_cv()`: leave-one-out CV
- `mc_cv()`: repeated holdout / Monte Carlo (random) CV: test sets sampled without replacement
- `bootstraps()`: test sets sampled with replacement



A classification example

```r
stackoverflow <- read_rds(here::here("data/stackoverflow.rds"))
glimpse(stackoverflow)
```

```
## Rows: 1,150
## Columns: 21
## $ country                              <fct> United States, United States, United Kingdo~
## $ salary                               <dbl> 63750.00, 93000.00, 40625.00, 45000.00, 100~
## $ years_coded_job                      <int> 4, 9, 8, 3, 8, 12, 20, 17, 20, 4, 3, 13, 16~
## $ open_source                          <dbl> 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1~
## $ hobby                                <dbl> 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1~
## $ company_size_number                  <dbl> 20, 1000, 10000, 1, 10, 100, 20, 500, 1, 20~
## $ remote                               <fct> Remote, Remote, Remote, Remote, Remote, Rem~
## $ career_satisfaction                  <int> 8, 8, 5, 10, 8, 10, 9, 7, 8, 7, 9, 8, 8, 7,~
## $ data_scientist                       <dbl> 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ database_administrator               <dbl> 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0~
## $ desktop_applications_developer       <dbl> 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0~
## $ developer_with_stats_math_background <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0~
## $ dev_ops                              <dbl> 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0~
## $ embedded_developer                   <dbl> 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0~
## $ graphic_designer                     <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ graphics_programming                 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ machine_learning_specialist          <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ mobile_developer                     <dbl> 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1~
## $ quality_assurance_engineer           <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ systems_administrator                <dbl> 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ web_developer                        <dbl> 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1~
```

Data source: [Stack Overflow Annual Developer Survey]

Specify a classification model

1. Pick a **model**
2. Set the **engine**
3. Set the **mode**

Specify a decision tree model with default parameter settings:

```
vanilla_tree_spec <- decision_tree() %>%
  set_engine("rpart") %>%
  set_mode("classification")
```

Measure the performance of a vanilla decision tree model using 5-fold CV:

```
set.seed(100)
so_cv <- vfold_cv(stackoverflow, v = 5)
(fit_van_res <- fit_resamples(vanilla_tree_spec, remote ~ ., resamples = so_cv) %>%
  collect_metrics())
```

```
## # A tibble: 2 x 6
##   .metric  .estimator  mean     n std_err .config
##   <chr>    <chr>      <dbl> <int>   <dbl> <chr>
## 1 accuracy binary     0.639     5 0.00870 Preprocessor1_Model1
## 2 roc_auc  binary     0.663     5 0.0155  Preprocessor1_Model1
```

🤔 *"Can we improve the performance by tuning the algorithm parameters?"*

🤔 *"Which parameters can we tune?"*

args()

`args()` prints the arguments for a parsnip model specification:

```
args(decision_tree)
```

```
## function (mode = "unknown", cost_complexity = NULL, tree_depth = NULL, 
##     min_n = NULL) 
## NULL
```

Arguments of `decision_tree()`:

- `cost_complexity`: minimum fit improvement of a split (0 < `cost_complexity` $\leq$ 1)
- `tree_depth`: maximum number of levels in the tree
- `min_n`: minimum number of observations in a node in order for a split to be attempted

```r
decision_tree(
  cost_complexity = 0.01,  # min. fit improvement of a split (0 < cp <=1)
  tree_depth = 30, # max. number of levels in the tree
  min_n = 20 # min. number of observations in a node in order for a split to be attempted
)
```

```
## Decision Tree Model Specification (unknown)
##
## Main Arguments:
##   cost_complexity = 0.01
##   tree_depth = 30
##   min_n = 20
```

If the arguments are left to their defaults (`NULL`), the arguments will use the engine's underlying model functions default value.

For example, `rpart` is used as default engine. The default parameters are:

```r
args(rpart::rpart.control) # cost_complexity -> cp; tree_depth -> maxdepth; min_n -> minsplit
```

```
## function (minsplit = 20L, minbucket = round(minsplit/3), cp = 0.01,
##     maxcompete = 4L, maxsurrogate = 5L, usesurrogate = 2L, xval = 10L,
##     surrogatestyle = 0L, maxdepth = 30L, ...)
## NULL
```

`set_args()`

`set_args()`: **change** the arguments for a parsnip model specification:

```
dt_spec <- decision_tree()

dt_spec %>% set_args(tree_depth = 3)
```

```
## Decision Tree Model Specification (unknown)
##
## Main Arguments:
##   tree_depth = 3
```

... which is equivalent to:

```
dt_spec <- decision_tree(tree_depth = 3)
dt_spec
```

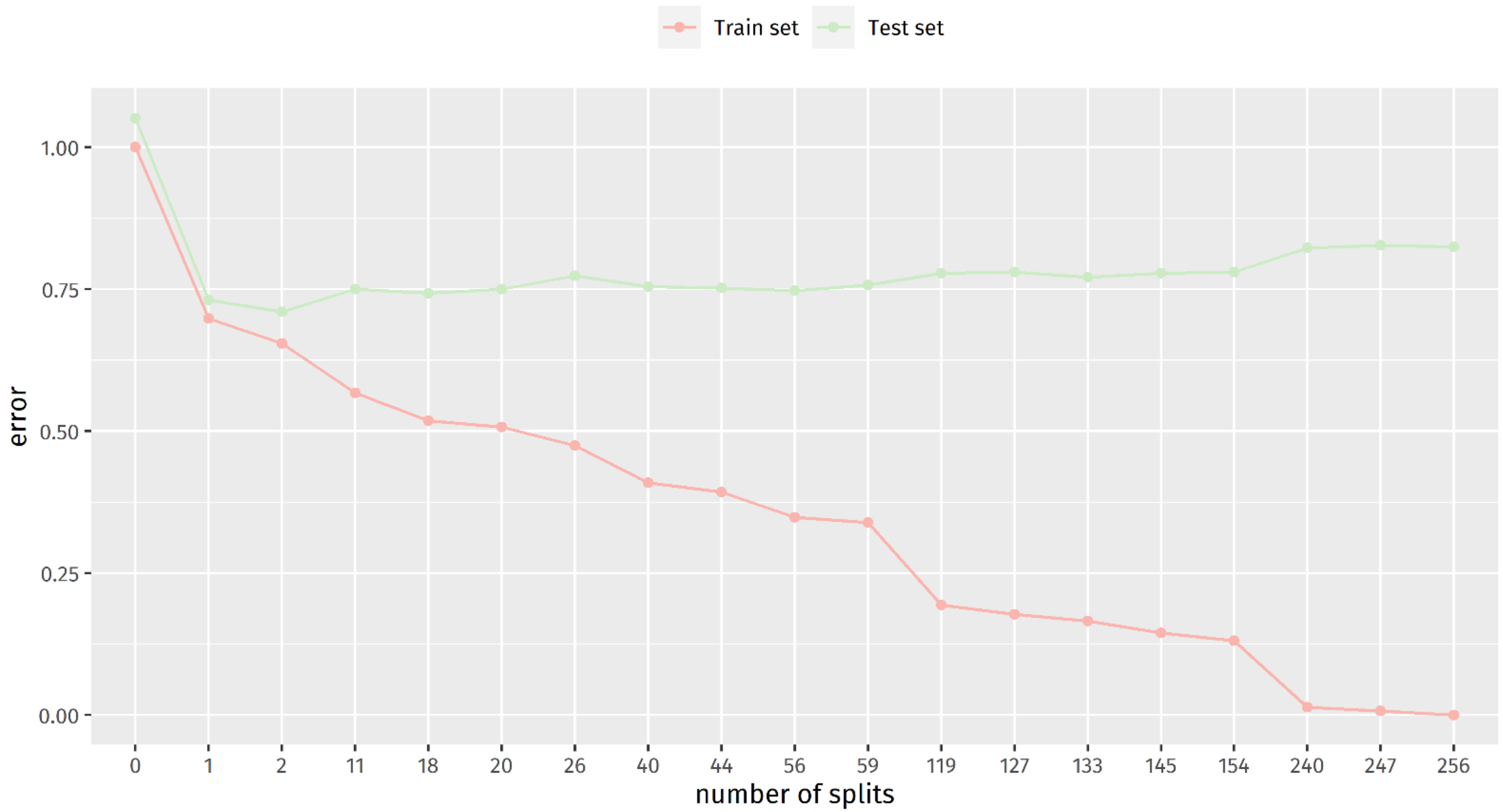
```
## Decision Tree Model Specification (unknown)
##
## Main Arguments:
##   tree_depth = 3
```

An example spec of model, engine, mode and tree depth:

```
decision_tree() %>%
  set_engine("rpart") %>%
  set_mode("classification") %>%
  set_args(tree_depth = 3)
```

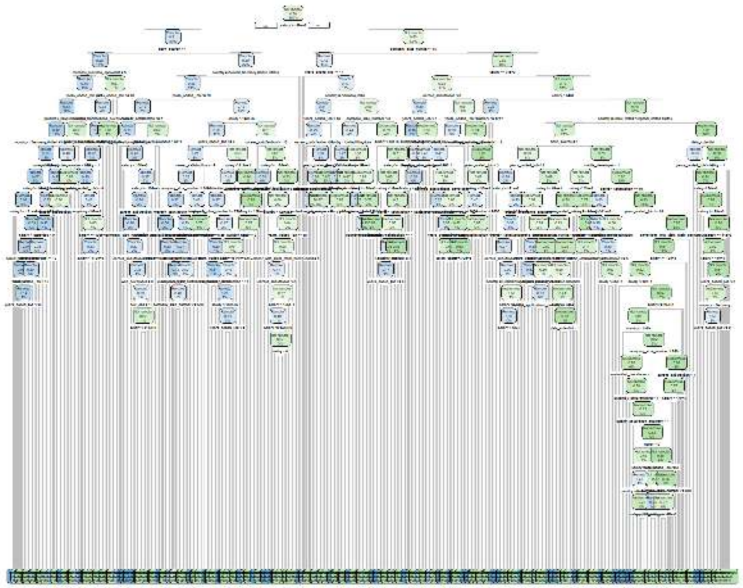
```
## Decision Tree Model Specification (classification)
##
## Main Arguments:
##   tree_depth = 3
##
## Computational engine: rpart
```


Train set
Test set
1.00
0.75
0.50
0.25
0.00
error
0.3014 0.0444 0.0093 0.007 0.0058 0.0051 0.0047 0.0039 0.0035 0.0031 0.0023 0.002 0.0019 0.0018 0.0016 0.0012 0.001 8e-04 0
cost complexity

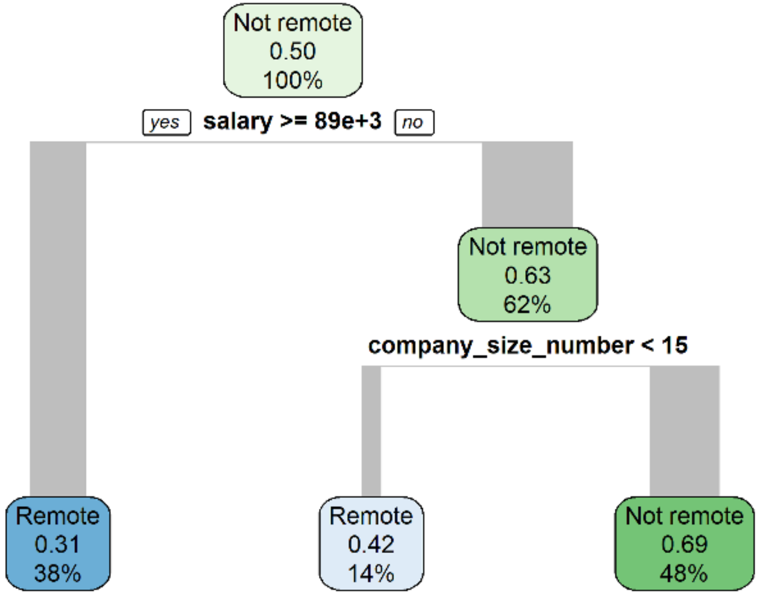


Train set
Test set
1.00
0.75
0.50
0.25
0.00
error
0
1
2
11
18
20
26
40
44
56
59
119
127
133
145
154
240
247
256
number of splits

Overfitted tree (`cost_complexity`=0.0008):



Optimal tree (`cost_complexity`=0.0093):



`workflow()`

Create a workflow with `workflow()`.

`add_formula()`

Add a formula to a workflow

```
workflow() %>% add_formula(Sale_Price ~ Year)
```

`add_model()`

Add a parsnip model spec to a workflow:

```
workflow() %>% add_model(lm_spec)
```

Example workflow

```
wf <- workflow() %>%
  add_formula(remote ~ .) %>%
  add_model(decision_tree() %>% set_engine("rpart") %>% set_mode("classification"))

wf %>% fit_resamples(so_cv)
```

```
## # Resampling results
## # 5-fold cross-validation
## # A tibble: 5 x 4
##   splits            id    .metrics             .notes
##   <list>            <chr> <list>               <list>
## 1 <split [920/230]> Fold1 <tibble[,4] [2 x 4]> <tibble[,1] [0 x 1]>
## 2 <split [920/230]> Fold2 <tibble[,4] [2 x 4]> <tibble[,1] [0 x 1]>
## 3 <split [920/230]> Fold3 <tibble[,4] [2 x 4]> <tibble[,1] [0 x 1]>
## 4 <split [920/230]> Fold4 <tibble[,4] [2 x 4]> <tibble[,1] [0 x 1]>
## 5 <split [920/230]> Fold5 <tibble[,4] [2 x 4]> <tibble[,1] [0 x 1]>
```

`update_formula()`

Replace a workflow formula with a new one:

```
workflow() %>%
  add_formula(remote ~ .) %>%
  update_formula(remote ~ salary + open_source)
```

```
## == Workflow ====================================================================
## Preprocessor: Formula
## Model: None
##
## -- Preprocessor ----------------------------------------------------------------
## remote ~ salary + open_source
```

`update_model()`

Replaces a workflow model spec with a new one:

```
workflow() %>%
  add_model(nearest_neighbor()) %>%
  update_model(decision_tree())
```

```
## == Workflow ==============================================================================
## Preprocessor: None
## Model: decision_tree()
##
## -- Model ---------------------------------------------------------------------------------
## Decision Tree Model Specification (unknown)
```

Tune model hyperparameters with tune

`tune()`

`tune()` is a placeholder for hyperparameters that are to be tuned:

```
decision_tree(cost_complexity = tune())
```

```
## Decision Tree Model Specification (unknown)
##
## Main Arguments:
##   cost_complexity = tune()
```

`tune_grid()`

A version of `fit_resamples()` that performs a grid search for the best combination of tuned hyper-parameters.

```
tune_grid(
  object, # a model workflow, R formula or recipe object.
  resamples, # a resampling object, e.g. the output of vfold_cv()
  ...,
  grid = 10, # the number of tuning iterations or a data frame of tuning operations (tuning grid)
  metrics = NULL, # yardstick::metric_set() or NULL
  control = control_grid() # An object used to modify the tuning process
)
```

`expand_grid()`

`tidyr::expand_grid()`: takes one or more vectors, and returns a data frame holding all combinations of their values.

```
expand_grid(cost_complexity = 10^(0:-5), min_n = seq(4,20,4))
```

```
## # A tibble: 30 x 2
##    cost_complexity min_n
##              <dbl> <dbl>
##  1             1       4
##  2             1       8
##  3             1      12
##  4             1      16
##  5             1      20
##  6             0.1     4
##  7             0.1     8
##  8             0.1    12
##  9             0.1    16
## 10             0.1    20
## # ... with 20 more rows
```

`expand_grid()` is a re-implementation of the base `expand.grid()`.

```r
dt_spec <- decision_tree(
  cost_complexity = tune(),
  tree_depth = tune()
) %>%
  set_engine("rpart") %>%
  set_mode("classification")

dt_wf <- workflow() %>%
  add_model(dt_spec) %>%
  add_formula(remote ~ .)

dt_res <- dt_wf %>%
    tune_grid(resamples = so_cv,
              grid = expand_grid(cost_complexity = 10^-(1:5), tree_depth = 1:6)
    )
dt_res
```

```
## # Tuning results
## # 5-fold cross-validation
## # A tibble: 5 x 4
##   splits            id    .metrics              .notes
##   <list>            <chr> <list>                <list>
## 1 <split [920/230]> Fold1 <tibble[,6] [60 x 6]> <tibble[,1] [0 x 1]>
## 2 <split [920/230]> Fold2 <tibble[,6] [60 x 6]> <tibble[,1] [0 x 1]>
## 3 <split [920/230]> Fold3 <tibble[,6] [60 x 6]> <tibble[,1] [0 x 1]>
## 4 <split [920/230]> Fold4 <tibble[,6] [60 x 6]> <tibble[,1] [0 x 1]>
## 5 <split [920/230]> Fold5 <tibble[,6] [60 x 6]> <tibble[,1] [0 x 1]>
```

```r
dt_res %>%
  collect_metrics() %>%
  filter(.metric == "accuracy") %>%
  arrange(desc(mean))
```

```
## # A tibble: 30 x 8
##    cost_complexity tree_depth .metric  .estimator  mean     n std_err .config
##              <dbl>      <int> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
##  1         0.001            2 accuracy binary     0.66      5 0.0158  Preprocessor1_Model~
##  2         0.0001           2 accuracy binary     0.66      5 0.0158  Preprocessor1_Model~
##  3         0.00001          2 accuracy binary     0.66      5 0.0158  Preprocessor1_Model~
##  4         0.01             2 accuracy binary     0.656     5 0.0142  Preprocessor1_Model~
##  5         0.01             3 accuracy binary     0.649     5 0.0142  Preprocessor1_Model~
##  6         0.001            5 accuracy binary     0.646     5 0.00488 Preprocessor1_Model~
##  7         0.001            6 accuracy binary     0.646     5 0.00918 Preprocessor1_Model~
##  8         0.0001           5 accuracy binary     0.646     5 0.00488 Preprocessor1_Model~
##  9         0.0001           6 accuracy binary     0.646     5 0.00918 Preprocessor1_Model~
## 10         0.00001          5 accuracy binary     0.646     5 0.00488 Preprocessor1_Model~
## # ... with 20 more rows
```

`show_best()`

`show_best()`: display the `n` best hyperparameters combinations according to a `metric`:

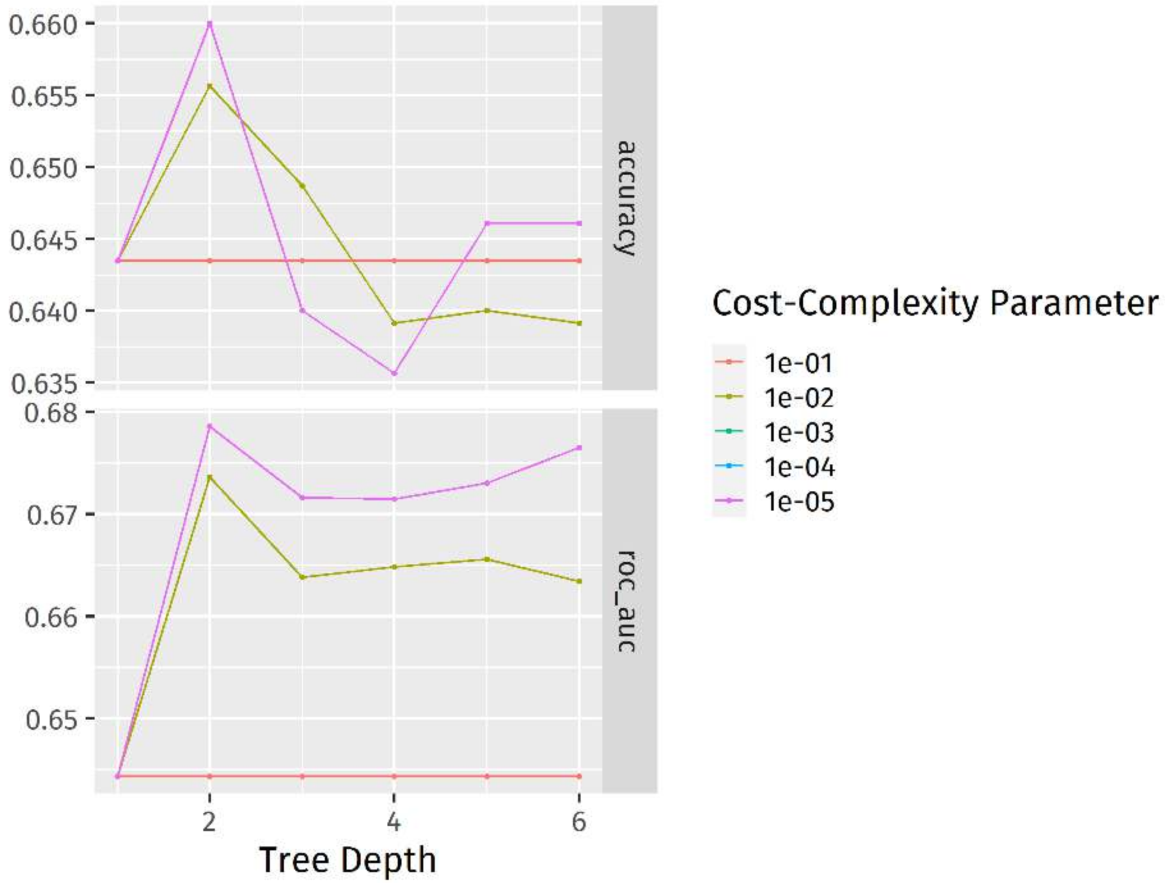
```
dt_res %>%
  show_best(metric = "accuracy", n = 5)
```

```
## # A tibble: 5 x 8
##   cost_complexity tree_depth .metric  .estimator  mean     n std_err .config
##             <dbl>      <int> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
## 1         0.001            2 accuracy binary     0.66      5  0.0158 Preprocessor1_Model14
## 2         0.0001           2 accuracy binary     0.66      5  0.0158 Preprocessor1_Model20
## 3         0.00001          2 accuracy binary     0.66      5  0.0158 Preprocessor1_Model26
## 4         0.01             2 accuracy binary     0.656     5  0.0142 Preprocessor1_Model08
## 5         0.01             3 accuracy binary     0.649     5  0.0142 Preprocessor1_Model09
```

`autoplot()`

`autoplot()`: quickly visualize tuning results

```
dt_res %>% autoplot()
```



`select_best()`

`select_best()` returns the best combination of hyperparameters according to a metric:

```
so_best <- dt_res %>% select_best(metric = "roc_auc")
so_best
```

```
## # A tibble: 1 x 3
##   cost_complexity tree_depth .config
##             <dbl>      <int> <chr>
## 1           0.001          2 Preprocessor1_Model14
```

`finalize_workflow()`

`finalize_workflow()`: replaces `tune()` placeholders in a model/recipe/workflow with a set of hyper-parameter values.

```
dt_wf_final <- dt_wf %>% finalize_workflow(so_best)
dt_wf_final
```

```
## == Workflow ====================================================================
## Preprocessor: Formula
## Model: decision_tree()
##
## -- Preprocessor ----------------------------------------------------------------
## remote ~ .
##
## -- Model -----------------------------------------------------------------------
## Decision Tree Model Specification (classification)
##
## Main Arguments:
##   cost_complexity = 0.001
##   tree_depth = 2
##
## Computational engine: rpart
```

Preprocessing with recipes

1. Create a `recipe()`
2. Define the predictor and outcome variables
3. Add one or more preprocessing step *specifications*
4. Calculate statistics from the training set
5. Apply preprocessing to datasets

1. Create a `recipe()`

2. Define the predictor and outcome variables

3. Add one or more preprocessing step *specifications*

4. Calculate statistics from the training set

5. Apply preprocessing to datasets

recipe()

`recipe()`: create a recipe by specifying predictors, responses and reference (*template*) data frame.

```
recipe(Sale_Price ~ ., data = ames)
```

```
## Data Recipe
##
## Inputs:
##
##       role #variables
##    outcome          1
##  predictor         73
```

1. Create a `recipe()`

2. Define the predictor and outcome variables

3. Add one or more preprocessing step *specifications*

4. Calculate statistics from the training set

5. Apply preprocessing to datasets

step_*()

`step_*()`: add preprocessing step specifications in the order they will be performed.

```
recipe(Sale_Price ~ ., data = ames) %>%
  # step_novel(): assign a previously unseen factor level to
  # a new value
  step_novel(all_nominal()) %>%
  # step_zv(): zero variance filter: remove vars that contain
  # only a single value
  step_zv(all_predictors())
```

```
## Data Recipe
##
## Inputs:
##
##       role #variables
##    outcome          1
##  predictor         73
##
## Operations:
##
## Novel factor level assignment for all_nominal()
## Zero variance filter on all_predictors()
```

step_*()

Complete list at: https://recipes.tidymodels.org/reference/index.html

`add_role()` `update_role()` `remove_role()`	Manually Alter Roles

STEP FUNCTIONS - IMPUTATION

`step_impute_bag()` `step_bagimpute()` `imp_vars()` `tidy(<step_impute_bag>)`	Imputation via Bagged Trees
`step_impute_knn()` `step_knnimpute()` `tidy(<step_impute_knn>)`	Imputation via K-Nearest Neighbors
`step_impute_linear()` `tidy(<step_impute_linear>)`	Imputation of numeric variables via a linear model.

CONTENTS

Selectors

Selectors, e.g., `all_nominal()` and `all_predictors()` are helper functions for selecting sets of variables, which behave similar to the select helpers from `dplyr`.

```
rec %>%
  step_novel(all_nominal()) %>%
  step_zv(all_predictors())
```

selector	description
`all_predictors()`	Each x variable (right side of ~)
`all_outcomes()`	Each y variable (left side of ~)
`all_numeric()`	Each numeric variable
`all_nominal()`	Each categorical variable (e.g. factor, string)
`dplyr::select()` helpers	`starts_with('Lot_')`, etc.

1. Create a `recipe()`

2. Define the predictor and outcome variables

3. Add one or more preprocessing step *specifications*

4. Calculate statistics from the training set

5. Apply preprocessing to datasets

`prep()`

`prep()` "trains" a recipe, i.e., calculates statistics from the training data

```
recipe(Sale_Price ~ ., data = ames) %>%
  step_novel(all_nominal()) %>%
  step_zv(all_predictors()) %>%
  prep(training = training(ames_split))
```

```
## Data Recipe
##
## Inputs:
##
##       role #variables
##    outcome          1
##  predictor         73
##
## Training data contained 2198 data points and no missing data.
##
## Operations:
##
## Novel factor level assignment for MS_SubClass, MS_Zoning, Street, Alley, ... [tr
## Zero variance filter removed no terms [trained]
```

1. Create a `recipe()`

2. Define the predictor and outcome variables

3. Add one or more preprocessing step *specifications*

4. Calculate statistics from the training set

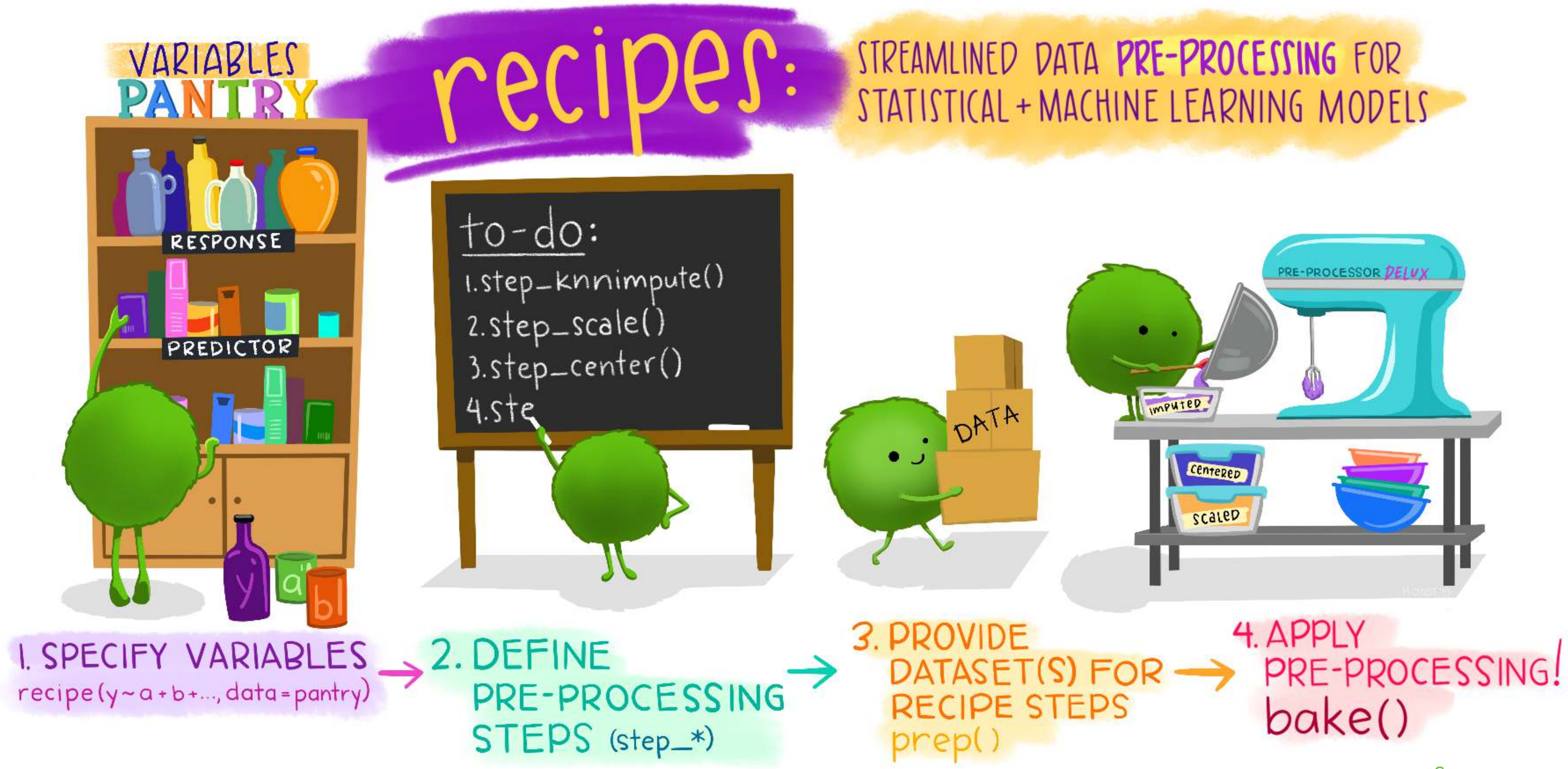
5. Apply preprocessing to datasets

`bake()`

`bake()` transforms data with the prepped recipe

```
recipe(Sale_Price ~ ., data = ames) %>%
  step_novel(all_nominal()) %>%
  step_zv(all_predictors()) %>%
  prep(training = training(ames_split)) %>%
  bake(new_data = testing(ames_split)) # or training(ames_split)
```

```
## # A tibble: 732 x 74
##    MS_SubClass     MS_Zoning   Lot_Frontage Lot_Area Street Alley  Lot_Shape  La
##    <fct>           <fct>              <dbl>    <int> <fct>  <fct>  <fct>      <f
##  1 One_Story_PUD_~ Residentia~           43     5005 Pave   No_Al~ Slightly_~ HL
##  2 One_Story_PUD_~ Residentia~           39     5389 Pave   No_Al~ Slightly_~ Lv
##  3 Two_Story_1946~ Residentia~           60     7500 Pave   No_Al~ Regular    Lv
##  4 Two_Story_1946~ Residentia~           63     8402 Pave   No_Al~ Slightly_~ Lv
##  5 Two_Story_1946~ Residentia~           47    53504 Pave   No_Al~ Moderatel~ HL
##  6 One_Story_1946~ Residentia~           88    11394 Pave   No_Al~ Regular    Lv
##  7 One_Story_1946~ Residentia~            0    11241 Pave   No_Al~ Slightly_~ Lv
##  8 Two_Story_PUD_~ Residentia~           21     1680 Pave   No_Al~ Regular    Lv
##  9 One_Story_1946~ Residentia~           95    12182 Pave   No_Al~ Regular    Lv
## 10 One_Story_1946~ Residentia~           70    10171 Pave   No_Al~ Slightly_~ Lv
## # ... with 722 more rows, and 66 more variables: Utilities <fct>, Lot_Config <fc
## #   Land_Slope <fct>, Neighborhood <fct>, Condition_1 <fct>, Condition_2 <fct>,
## #   Bldg_Type <fct>, House_Style <fct>, Overall_Cond <fct>, Year_Built <int>,
## #   Year_Remod_Add <int>, Roof_Style <fct>, Roof_Matl <fct>, Exterior_1st <fct>,
## #   Exterior_2nd <fct>, Mas_Vnr_Type <fct>, Mas_Vnr_Area <dbl>, Exter_Cond <fct>
## #   Foundation <fct>, Bsmt_Cond <fct>, Bsmt_Exposure <fct>, BsmtFin_Type_1 <fct>
## #   BsmtFin_SF_1 <dbl>, BsmtFin_Type_2 <fct>, BsmtFin_SF_2 <dbl>, Bsmt_Unf_SF <d
```


VARIABLES PANTRY
recipes:
STREAMLINED DATA PRE-PROCESSING FOR STATISTICAL + MACHINE LEARNING MODELS
RESPONSE
PREDICTOR
to-do:
1.step_knnimpute()
2.step_scale()
3.step_center()
4.ste
DATA
PRE-PROCESSOR DELUX
imputed
centered
scaled
y
a
bl
1. SPECIFY VARIABLES
recipe(y ~ a + b + ..., data = pantry)
2. DEFINE PRE-PROCESSING STEPS (step_*)
3. PROVIDE DATASET(S) FOR RECIPE STEPS
prep()
4. APPLY PRE-PROCESSING!
bake()

Source

`juice()`

`juice()` returns the preprocessed training data back from a prepped recipe, without having to rerun the preprocessing steps on the training data.

```
rec <- recipe(Sale_Price ~ ., data = ames) %>%
    step_center(all_numeric()) %>%
    step_scale(all_numeric())
rec %>%
  prep(training = training(ames_split),
       retain = TRUE
  ) %>%
  juice()
```

```
## # A tibble: 2,198 x 74
##    MS_SubClass     MS_Zoning   Lot_Frontage Lot_Area Street Alley  Lot_Shape   Land_Contour
##    <fct>           <fct>              <dbl>    <dbl> <fct>  <fct>  <fct>       <fct>
##  1 One_Story_1946~ Residentia~        2.46     2.64   Pave   No_Al~ Slightly_~ Lvl
##  2 One_Story_1946~ Residentia~        0.658    0.185  Pave   No_Al~ Regular    Lvl
##  3 One_Story_1946~ Residentia~        0.687    0.507  Pave   No_Al~ Slightly_~ Lvl
##  4 One_Story_1946~ Residentia~        1.04     0.128  Pave   No_Al~ Regular    Lvl
##  5 Two_Story_1946~ Residentia~        0.480    0.454  Pave   No_Al~ Slightly_~ Lvl
##  6 Two_Story_1946~ Residentia~        0.598 -0.0156   Pave   No_Al~ Slightly_~ Lvl
##  7 One_Story_PUD_~ Residentia~       -0.496 -0.632    Pave   No_Al~ Regular    Lvl
##  8 Two_Story_1946~ Residentia~        0.510 -0.0129   Pave   No_Al~ Slightly_~ Lvl
##  9 One_Story_1946~ Residentia~       -1.71  -0.259    Pave   No_Al~ Slightly_~ Lvl
## 10 One_Story_1946~ Residentia~        0.805  0.00851  Pave   No_Al~ Regular    Lvl
## # ... with 2,188 more rows, and 66 more variables: Utilities <fct>, Lot_Config <fct>,
## #   Land_Slope <fct>, Neighborhood <fct>, Condition_1 <fct>, Condition_2 <fct>,
## #   Bldg_Type <fct>, House_Style <fct>, Overall_Cond <fct>, Year_Built <dbl>,
## #   Year_Remod_Add <dbl>, Roof_Style <fct>, Roof_Matl <fct>, Exterior_1st <fct>,
```

A full workflow

```r
set.seed(123)
so_cv <- vfold_cv(stackoverflow, v = 5)
so_rec <- recipe(remote ~ ., data = stackoverflow) %>%
  step_dummy(all_nominal(), -all_outcomes()) %>%
  step_corr(all_predictors(), threshold = 0.5)

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

so_wf <- workflow() %>%
  add_model(tree_spec) %>%
  add_recipe(so_rec)

fit_resamples(so_wf, # note: workflow object instead of model spec
              resamples = so_cv,
              metrics = metric_set(accuracy, sens, spec),
              control = control_resamples(save_pred = TRUE)) %>%
  # collect_metrics() %>%
  collect_predictions() %>%
  conf_mat(remote, .pred_class)
```

```
##             Truth
## Prediction   Remote Not remote
##   Remote        381        224
##   Not remote    194        351
```

You can tune models **and** recipes!

```
pca_tuner <- recipe(Sale_Price ~ ., data = ames) %>%
    step_novel(all_nominal()) %>%
    step_dummy(all_nominal()) %>%
    step_zv(all_predictors()) %>%
    step_center(all_predictors()) %>%
    step_scale(all_predictors()) %>%
    step_pca(all_predictors(), num_comp = tune())
pca_twf <- workflow() %>%
    add_recipe(pca_tuner) %>%
    add_model(nearest_neighbor(neighbors = tune()) %>%
                set_engine("kknn") %>% set_mode("regression"))
tg <- expand_grid(num_comp = 2:10, neighbors = seq(1, 15, 4))
set.seed(100)
cv_folds <- vfold_cv(ames, v = 5, strata = Sale_Price, breaks = 4)
set.seed(100)
pca_results <- pca_twf %>%
    tune_grid(resamples = cv_folds, grid = tg)
pca_results %>% show_best(metric = "rmse")
```

```
## # A tibble: 5 x 8
##   neighbors num_comp .metric .estimator   mean     n std_err .config
##       <dbl>    <int> <chr>   <chr>       <dbl> <int>   <dbl> <chr>
## 1         9        7 rmse    standard   31793.     5    968. Preprocessor6_Model3
## 2        13        7 rmse    standard   31961.     5   1157. Preprocessor6_Model4
## 3         9        8 rmse    standard   31963.     5   1099. Preprocessor7_Model3
## 4         9        5 rmse    standard   32141.     5    951. Preprocessor4_Model3
## 5        13        8 rmse    standard   32180.     5   1234. Preprocessor7_Model4
```

Session info

```
## setting  value
## version  R version 4.0.5 (2021-03-31)
## os       Windows 10 x64
## system   x86_64, mingw32
## ui       RTerm
## language (EN)
## collate  English_United States.1252
## ctype    English_United States.1252
## tz       Europe/Berlin
## date     2021-05-10
```

package	version	date	source
AmesHousing	0.0.4	2020-06-23	CRAN (R 4.0.3)
broom	0.7.6	2021-04-05	CRAN (R 4.0.5)
dials	0.0.9	2020-09-16	CRAN (R 4.0.3)
dplyr	1.0.5	2021-03-05	CRAN (R 4.0.4)
forcats	0.5.1	2021-01-27	CRAN (R 4.0.3)
ggplot2	3.3.3	2020-12-30	CRAN (R 4.0.3)
infer	0.5.4	2021-01-13	CRAN (R 4.0.3)
kableExtra	1.3.4	2021-02-20	CRAN (R 4.0.3)
kknn	1.3.1	2016-03-26	CRAN (R 4.0.4)
knitr	1.31	2021-01-27	CRAN (R 4.0.3)
modeldata	0.1.0	2020-10-22	CRAN (R 4.0.5)
parsnip	0.1.5	2021-01-19	CRAN (R 4.0.3)
patchwork	1.1.1	2020-12-17	CRAN (R 4.0.3)
purrr	0.3.4	2020-04-17	CRAN (R 4.0.2)
readr	1.4.0	2020-10-05	CRAN (R 4.0.3)

package	version	date	source
recipes	0.1.15	2020-11-11	CRAN (R 4.0.3)
rlang	0.4.11	2021-04-30	CRAN (R 4.0.5)
rpart	4.1.15	2019-04-12	CRAN (R 4.0.5)
rpart.plot	3.0.9	2020-09-17	CRAN (R 4.0.2)
rsample	0.0.9	2021-02-17	CRAN (R 4.0.4)
scales	1.1.1	2020-05-11	CRAN (R 4.0.2)
stringr	1.4.0	2019-02-10	CRAN (R 4.0.2)
tibble	3.1.1	2021-04-18	CRAN (R 4.0.5)
tidymodels	0.1.2	2020-11-22	CRAN (R 4.0.3)
tidyr	1.1.3	2021-03-03	CRAN (R 4.0.4)
tidyverse	1.3.0	2019-11-21	CRAN (R 4.0.2)
tune	0.1.2	2020-11-17	CRAN (R 4.0.3)
vctrs	0.3.8	2021-04-29	CRAN (R 4.0.5)
workflows	0.2.2	2021-03-10	CRAN (R 4.0.4)
yardstick	0.0.7	2020-07-13	CRAN (R 4.0.3)

Thank you! Questions?

Photo courtesy of Stefan Berger