Introduction to Machine Learning in R

Lab 2: Supervised Learning

Table of contents

Load Required Packages

```
library(here)
library(tidyverse)
library(tidymodels)
library(modelsummary)
library(skimr)
library(naniar)
library(xgboost)
library(vip)
library(kableExtra)
```

1. Preprocessing data: recipes & workflows

A short recipe example

- **Example below**: A recipe containing an outcome plus two numeric predictors that centers and scale ("normalize") the predictors

```
load(file = here("data/data_ess.Rdata"))
```

We start by loading the data and selecting a subset for illustration. We also inspect the data to identify any differences later.

	Unique	Missing Pct.	Mean	SD	Min	Median	Max	Histogram
respondent_id	1977	0	18 947.9	5193.1	10 005.0	18 927.0	27 908.0	
life_satisfaction	12	10	7.0	2.2	0.0	8.0	10.0	
age	76	0	49.5	18.7	16.0	50.0	90.0	
education	8	1	3.1	1.9	0.0	3.0	6.0	

		N	%
internet_use_frequency	Never	196	9.9
	Only occasionally	97	4.9
	A few times a week	78	3.9
	Most days	180	9.1
	Every day	1426	72.1
religion	Roman Catholic	751	38.0
	Protestant	41	2.1
	Eastern Orthodox	9	0.5
	Other Christian denomination	13	0.7
	Jewish	11	0.6
	Islam	150	7.6
	Eastern religions	5	0.3
	Other Non-Christian religions	9	0.5

```r
# Select a few variables for illustration
data <- data %>%
  select(respondent_id, life_satisfaction, age, education, internet_use_frequency, religion)
datasummary_skim(data, type = "numeric")
```

```r
datasummary_skim(data, type = "categorical")
```

Then we define our recipe (the single steps are concatenated with `%>%`). Make sure that you pick an order that makes sense, e.g., add polynomials to numeric variables before you convert categorical variables to numeric:

```r
recipe1 <- # Store recipe in object recipe 1
```

```r
# Step 1: Define formula and data
    # "." -> all predictors
  recipe(life_satisfaction ~ ., data = data) %>% # Define formula; use "." to select all pred

# Step 2: Define roles
  update_role(respondent_id, new_role = "ID") %>% # Define ID variable

# Step 3: Handle Missing Values
  step_naomit(all_predictors()) %>%

# Step 4: Feature Scaling (if applicable)
  # Assuming you have numerical features that need scaling
  step_normalize(all_numeric_predictors()) %>%

# Step 5: Add polynomials for all numeric predictors
  step_poly(all_numeric_predictors(), degree = 2,
            keep_original_cols = TRUE,
            options = list(raw = TRUE)) %>%

# Step 6: Encode Categorical Variables (AFTER TREATING OTHER NUMERIC VARIABLES)
  step_dummy(all_nominal_predictors(), one_hot = TRUE)
  # see also step_ordinalscore() to convert to numeric

# Inspect the recipe
  recipe1
```

```
-- Recipe ----------------------------------------------------------------------

-- Inputs

Number of variables by role

outcome:   1
predictor: 4
ID:        1
```

```
-- Operations

* Removing rows with NA values in: all_predictors()

* Centering and scaling for: all_numeric_predictors()

* Orthogonal polynomials on: all_numeric_predictors()

* Dummy variables from: all_nominal_predictors()
```

Now we can apply the recipe to some data and explore how the data changes:

```r
# Now you can apply the recipe to your data
data_preprocessed <- prep(recipe1, data)

# Access and inspect the preprocessed data with $
# View(data_preprocessed$template)
skim(data_preprocessed$template)
```

Table 1: Data summary

Name	data_preprocessed$templat...
Number of rows	985
Number of columns	21
Column type frequency:	
numeric	21
Group variables	None

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
respondent_id	0	1.0	18920.43	5165.06	10005.00	14473.00	18962.00	23428.00	27887.00	
age	0	1.0	0.00	1.00	-1.91	-0.76	0.02	0.81	1.96	
education	0	1.0	0.00	1.00	-1.56	-0.53	-0.01	0.50	1.53	

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
life_satisfaction	101	0.9	7.12	2.07	0.00	6.00	8.00	8.00	10.00	
age_poly_1	0	1.0	0.00	1.00	-1.91	-0.76	0.02	0.81	1.96	
age_poly_2	0	1.0	1.00	1.06	0.00	0.15	0.58	1.51	3.84	
education_poly_1	0	1.0	0.00	1.00	-1.56	-0.53	-0.01	0.50	1.53	
education_poly_2	0	1.0	1.00	0.98	0.00	0.25	0.28	2.35	2.43	
internet_use_frequency_1	0	1.0	0.12	0.33	0.00	0.00	0.00	0.00	1.00	
internet_use_frequency_2	0	1.0	0.06	0.23	0.00	0.00	0.00	0.00	1.00	
internet_use_frequency_3	0	1.0	0.04	0.20	0.00	0.00	0.00	0.00	1.00	
internet_use_frequency_4	0	1.0	0.10	0.30	0.00	0.00	0.00	0.00	1.00	
internet_use_frequency_5	0	1.0	0.68	0.47	0.00	0.00	1.00	1.00	1.00	
religion_1	0	1.0	0.76	0.43	0.00	1.00	1.00	1.00	1.00	
religion_2	0	1.0	0.04	0.20	0.00	0.00	0.00	0.00	1.00	
religion_3	0	1.0	0.01	0.10	0.00	0.00	0.00	0.00	1.00	
religion_4	0	1.0	0.01	0.11	0.00	0.00	0.00	0.00	1.00	
religion_5	0	1.0	0.01	0.11	0.00	0.00	0.00	0.00	1.00	
religion_6	0	1.0	0.15	0.36	0.00	0.00	0.00	0.00	1.00	
religion_7	0	1.0	0.01	0.07	0.00	0.00	0.00	0.00	1.00	
religion_8	0	1.0	0.01	0.10	0.00	0.00	0.00	0.00	1.00	

Usually the recipe is simply part of the workflow. Hence, we do not need to use the `prep()` and `bake()` function. Below we'll see an example. Besides, recipes can include a wide variety of preprocessing steps including functions such as `themis::step_downsample()`, `step_corr()` and `step_rm()` to downsample outcome data and omit highly correlated predictors.

A short workflow example

```r
# Below we simply use one dataset (and do not split)
  dim(data)
```

```
[1] 1977    6
```

```r
  names(data)
```

```
[1] "respondent_id"          "life_satisfaction"      "age"
[4] "education"              "internet_use_frequency" "religion"
```

```r
# Define a recipe for preprocessing (taken from above)
  recipe1 <-
    recipe(life_satisfaction ~ ., data = data) %>% # Define formula;
    update_role(respondent_id, new_role = "ID") %>% # Define ID variable
    step_unknown(religion) %>% # Change missings to unknown
    step_naomit(all_predictors()) %>%
    step_normalize(all_numeric_predictors()) %>%
    step_poly(all_numeric_predictors(), degree = 2,
            keep_original_cols = TRUE,
            options = list(raw = TRUE)) %>%
    step_dummy(all_nominal_predictors(), one_hot = TRUE)

# Define a model
  model1 <- linear_reg() %>% # linear model
    set_engine("lm") %>% # lm engine
    set_mode("regression") # regression problem


# Define a workflow
  workflow1 <- workflow() %>% # create empty workflow
    add_recipe(recipe1) %>% # add recipe
    add_model(model1) # add model

  workflow1 # Inspect
```

```
== Workflow ====================================================================
Preprocessor: Recipe
Model: linear_reg()

-- Preprocessor ----------------------------------------------------------------
5 Recipe Steps

* step_unknown()
* step_naomit()
* step_normalize()
* step_poly()
* step_dummy()

-- Model -----------------------------------------------------------------------
Linear Regression Model Specification (regression)

Computational engine: lm
```

```r
# Train the model (on all the data.. no split here..)
  fit1 <- fit(workflow1, data = data)

# Print summary of the trained model
  fit1
```

```
== Workflow [trained] ==========================================================
Preprocessor: Recipe
Model: linear_reg()

-- Preprocessor ----------------------------------------------------------------
5 Recipe Steps

* step_unknown()
* step_naomit()
* step_normalize()
* step_poly()
* step_dummy()

-- Model -----------------------------------------------------------------------

Call:
stats::lm(formula = ..y ~ ., data = data)

Coefficients:
             (Intercept)                       age                 education
                 6.71233                  -0.07823                   0.33949
              age_poly_1                age_poly_2          education_poly_1
                      NA                   0.25963                        NA
        education_poly_2  internet_use_frequency_1  internet_use_frequency_2
                 0.07600                  -0.67215                  -0.32091
internet_use_frequency_3  internet_use_frequency_4  internet_use_frequency_5
                -0.56426                  -0.15079                        NA
              religion_1                religion_2                religion_3
                 0.35125                  -0.34083                  -0.25586
              religion_4                religion_5                religion_6
                -0.33058                  -0.69344                  -0.22800
              religion_7                religion_8                religion_9
                -0.46725                  -0.66307                        NA
```

Exercise: Using a workflow to built a linear predictive model

1. See earlier description of the data (Slides day 1)
2. Below you find code that uses a workflow (recipe + model) to built a predictive model.
3. Start be running the code (loading the packages and data beforehand). How accurate is the model in the training and testdataset?
4. Then rerun the code after the marker "AFTER SPLIT" and change the preprocessing steps, e.g., increase the `degree` in `step_poly()` (and more if you like). How does the accuracy in training and test data change after you edit the preprocessing steps? (Important: Normally we would run such tests with a validation dataset. Once happy we would tes the model on the final dataset.)

We first import the data into R:

```
load(file = here("data/data_ess.Rdata"))

# Subset variables
data <- data %>%
  select(respondent_id, life_satisfaction, age, education,
         internet_use_frequency, religion, trust_people)


# Extract data with missing outcome
  data_missing_outcome <- data %>% filter(is.na(life_satisfaction))
  dim(data_missing_outcome)

# Omit individuals with missing outcome from data
  data <- data %>% drop_na(life_satisfaction) # ?drop_na
  dim(data)

# Split the data into training and test data
  set.seed(1234)
  data_split <- initial_split(data, prop = 0.80)
  data_split # Inspect

# Extract the two datasets
  data_train <- training(data_split)
  data_test <- testing(data_split) # Do not touch until the end!



# AFTER SPLIT
```

```r
# Define a recipe for preprocessing (taken from above)
  recipe1 <- recipe(life_satisfaction ~ ., data = data_train) %>% # Define formula;
    update_role(respondent_id, new_role = "ID") %>% # Define ID variable
    step_impute_mean(all_numeric_predictors()) %>%
    step_naomit(all_predictors()) %>%
    step_normalize(all_numeric_predictors()) %>%
    step_poly(all_numeric_predictors(), degree = 1,
            keep_original_cols = TRUE,
            options = list(raw = TRUE)) %>%
    step_dummy(all_nominal_predictors(), one_hot = TRUE)

  recipe1

# Define a model
  model1 <- linear_reg() %>% # linear model
    set_engine("lm") %>% # lm engine
    set_mode("regression") # regression problem

# Define a workflow
  workflow1 <- workflow() %>% # create empty workflow
    add_recipe(recipe1) %>% # add recipe
    add_model(model1) # add model

# Fit the workflow (including recipe and model)
  fit1 <- workflow1 %>% fit(data = data_train)


# Training data: Add predictions & calculate metrics
  augment(fit1, data_train) %>%
    metrics(truth = life_satisfaction, estimate = .pred)

# Test data: Add predictions & calculate metrics
  augment(fit1, data_test) %>%
    metrics(truth = life_satisfaction, estimate = .pred)
```

Appendix: Other preprocessing&/preparation steps

Dropping data

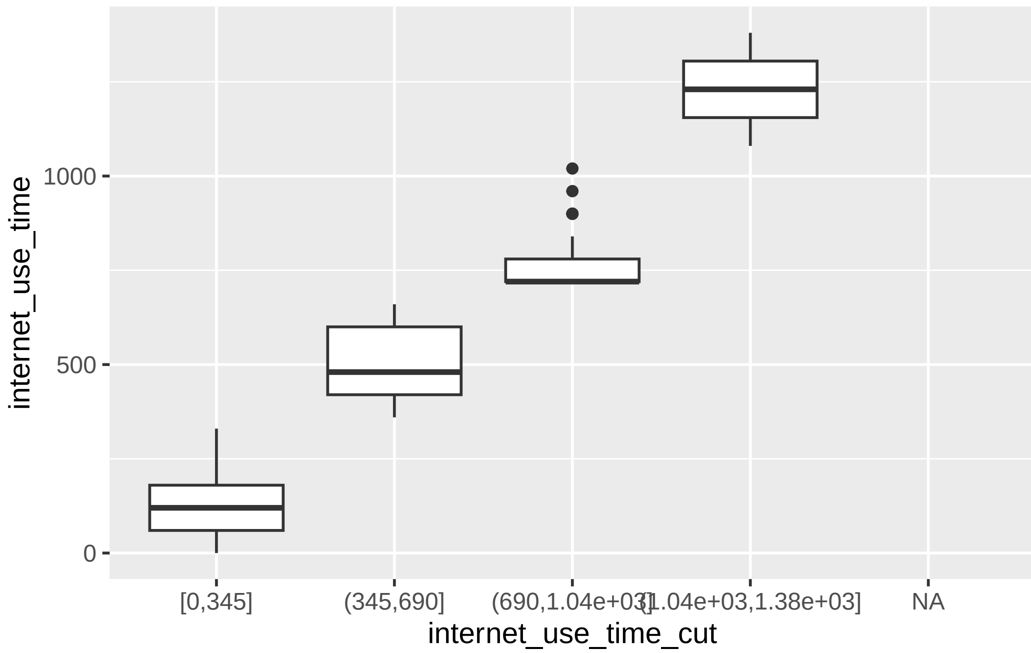
```r
# Dropping data
data <- data %>%
  drop_na(life_satisfaction) %>% # Drop missing on outcome
  select(where(~mean(is.na(.)) < 0.5)) # select features with less than X % missing

# data_listwise <- data %>% na.omit() # Listwise deletion (Be careful!)
```

Discretizing data

Sometimes it makes sense to discretize data, i.e., convert it to less categories. Below we create a new variable based on `internet_use_time` with four categories:

```r
load(file = here("data/data_ess.Rdata"))

# Dropping data
data <- data %>%
  mutate(internet_use_time_cut = cut_interval(internet_use_time, 4))

ggplot(data = data,
       aes(x = internet_use_time_cut,
           y = internet_use_time)) +
  geom_boxplot()
```

```
Warning: Removed 375 rows containing non-finite outside the scale range
(`stat_boxplot()`).
```



Deleting variables with too many categories

Below we use a loop to explore whether our data contains factor variables that have many values. Subsequently, we could delete them.

```
# Identify factors with too many levels

  # Identify factors with too many levels
    for(i in names(data)){

      if(!is.factor(data %>% pull(i))) next # Skip non-factors

      if(length(levels(data %>% pull(i)))<9) next # Skip if levels < X

      cat("\n\n\n",i,"\n") # Print variable
      print(levels(data %>% pull(i))) # Print levels

      Sys.sleep(0) # Increase if many variables

    }
```

```
country
 [1] "BE" "BG" "CH" "CZ" "EE" "FI" "FR" "GB" "GR" "HR" "HU" "IE" "IS" "IT" "LT"
[16] "ME" "MK" "NL" "NO" "PT" "SI" "SK"
```

```
rlgdme
 [1] "Roman Catholic"
 [2] "Protestant"
 [3] "Serbian Orthodox"
 [4] "Montenegrin Orthodox"
 [5] "Other Eastern Orthodox, denomination not specified"
 [6] "Other Christian denominations"
 [7] "Jewish"
 [8] "Islam"
 [9] "Eastern religions"
[10] "Other Non-Christian religions"
```

```
household_net_income
 [1] "J - 1st decile"  "R - 2nd decile"  "C - 3rd decile"  "M - 4th decile"
 [5] "F - 5th decile"  "S - 6th decile"  "K - 7th decile"  "P - 8th decile"
 [9] "D - 9th decile"  "H - 10th decile"
```

```
father_occupation_at_14
[1] "Professional and technical occupations"
[2] "Higher administrator occupations"
[3] "Clerical occupations"
[4] "Sales occupations"
[5] "Service occupations"
[6] "Skilled worker"
[7] "Semi-skilled worker"
[8] "Unskilled worker"
[9] "Farm worker"
```

```
mother_occupation_at_14
```

```
[1] "Professional and technical occupations"
[2] "Higher administrator occupations"
[3] "Clerical occupations"
[4] "Sales occupations"
[5] "Service occupations"
[6] "Skilled worker"
[7] "Semi-skilled worker"
[8] "Unskilled worker"
[9] "Farm worker"
```

Afterwards we could decide to drop those variables or to recode them in some fashion.

2. Resampling & cross-validation

Evaluate a classification model using training, validation and test dataset

```
load(file = here("data/data_compas.Rdata"))
```

Steps:

1. Split data into training, validation and test data.
2. Define the recipe & model
3. Bundle the model/formula into a workflow
4. Fit the workflow on the training data and evaluate on validation data -> if not happy change workflow/recipe/model
5. If happy with the accuracy estimate model on complete training dataset (analysis + assessment) and evaluate accuracy in test data (holdout dataset)

```
# Extract data with missing outcome
  data_missing_outcome <- data %>% filter(is.na(is_recid))
  dim(data_missing_outcome)
```

```
[1] 614  14
```

```
# Omit individuals with missing outcome from data
  data <- data %>% drop_na(is_recid) # ?drop_na
  dim(data)
```

```
[1] 6600   14
```

```r
# 1.
# Split the data into training, validation and test data
  set.seed(1234)
  data_split <- initial_validation_split(data, prop = c(0.6, 0.2))
  data_split # Inspect
```

```
<Training/Validation/Testing/Total>
<3960/1320/1320/6600>
```

```r
# Extract the datasets
  data_train <- training(data_split)
  data_validation <- validation(data_split)
  data_test <- testing(data_split) # Do not touch until the end!
  dim(data_train)
```

```
[1] 3960   14
```

```r
  dim(data_validation)
```

```
[1] 1320   14
```

```r
  dim(data_test)
```

```
[1] 1320   14
```

```r
# 2.
# Define the recipe & model
  recipe1 <- recipe(is_recid_factor ~ age + priors_count +
                      sex + race, data = data_train)

  model1 <- logistic_reg() %>%
    set_engine("glm") %>% # lm engine
    set_mode("classification") # regression problem

# 3.
# Create workflow
  workflow1 <-
  workflow() %>%
  add_model(model1) %>%
```

```
  add_recipe(recipe1)

# 4. Fit the workflow on training set & accuracy
  fit_train <- workflow1 %>% fit(data = data_train)

  data_train <- augment(fit_train, data_train,
                        type.predict = "response")

  data_train %>%
      metrics(truth = is_recid_factor, estimate = .pred_class)
```

```
# A tibble: 2 x 3
  .metric  .estimator .estimate
  <chr>    <chr>          <dbl>
1 accuracy binary         0.673
2 kap      binary         0.344
```

```
  # Predict & assess metrics in validation set
  augment(fit_train, data_validation, # Make sure to use training fit!
          type.predict = "response") %>%
      metrics(truth = is_recid_factor, estimate = .pred_class)
```

```
# A tibble: 2 x 3
  .metric  .estimator .estimate
  <chr>    <chr>          <dbl>
1 accuracy binary         0.670
2 kap      binary         0.336
```

```
# 5. If happy fit workflow on full
  # training set (data_train + data validation)
  # and predict values on test set
  data_training_full <- bind_rows(data_train, data_validation)
  fit_train_full <- workflow1 %>% fit(data = data_training_full)

  # Predict and assess accuracy
  augment(fit_train_full, data_test,
          type.predict = "response") %>%
      metrics(truth = is_recid_factor, estimate = .pred_class)
```

```
# A tibble: 2 x 3
```

```
  .metric  .estimator .estimate
  <chr>    <chr>          <dbl>
1 accuracy binary         0.671
2 kap      binary         0.341
```

Evaluate a classification model with resampling

```
load(file = here("data/data_compas.Rdata"))
```

Steps:

1. Initial first split of the data
2. Create resampled partitions/folds with `vfold_cv()`
3. Define the recipe & model
4. Bundle the model/formula into a workflow
5. Fit the workflow on the resamples/folds & extract accuracy metrics
6. If happy with the accuracy estimate model on complete training dataset and evaluate accuracy in test data (holdout dataset)

```
# Extract data with missing outcome
  data_missing_outcome <- data %>%
                  filter(is.na(is_recid_factor))
  dim(data_missing_outcome)
```

```
[1] 614  14
```

```
# Omit individuals with missing outcome from data
  data <- data %>% drop_na(is_recid_factor) # ?drop_na
  dim(data)
```

```
[1] 6600   14
```

```
# 1.
# Split the data into training and test data
  data_split <- initial_split(data, prop = 0.8)
  data_split # Inspect
```

```
<Training/Testing/Total>
<5280/1320/6600>
```

```r
# Extract the two datasets
  data_train <- training(data_split)
  data_test <- testing(data_split) # Do not touch until the end!

# 2.
# Create resampled partitions of training data
  set.seed(345)
  data_folds <- vfold_cv(data_train, v = 10) # V-fold/k-fold cross-validation
  data_folds # data_folds now contains several resamples of our training data
```

```
#  10-fold cross-validation
# A tibble: 10 x 2
   splits             id
   <list>             <chr>
 1 <split [4752/528]> Fold01
 2 <split [4752/528]> Fold02
 3 <split [4752/528]> Fold03
 4 <split [4752/528]> Fold04
 5 <split [4752/528]> Fold05
 6 <split [4752/528]> Fold06
 7 <split [4752/528]> Fold07
 8 <split [4752/528]> Fold08
 9 <split [4752/528]> Fold09
10 <split [4752/528]> Fold10
```

```r
  # v = 10 -> n/10 gives number of validation set observation in each fold



# 3.
# Define the recipe & model
  recipe1 <- recipe(is_recid_factor ~ age + priors_count +
                      sex + race, data = data_train)

  model1 <- logistic_reg() %>%
    set_engine("glm") %>% # lm engine
    set_mode("classification") # regression problem

# 4.
# Create workflow
  workflow1 <-
```

```r
  workflow() %>%
  add_model(model1) %>%
  add_recipe(recipe1)

# 5. Fit the workflow on the folds/resamples
  # There is no need in specifying
  # data_analysis/data_assessment as
  # the functions understand the corresponding parts
  fit1 <-
  workflow1 %>%
  fit_resamples(resamples = data_folds)
  # add argument "control" to keep predictions
    # control = control_resamples(save_pred = TRUE, extract = function (x) extract_fit_parsni

  fit1
```

```
# Resampling results
# 10-fold cross-validation
# A tibble: 10 x 4
   splits             id     .metrics         .notes
   <list>             <chr>  <list>           <list>
 1 <split [4752/528]> Fold01 <tibble [3 x 4]> <tibble [0 x 3]>
 2 <split [4752/528]> Fold02 <tibble [3 x 4]> <tibble [0 x 3]>
 3 <split [4752/528]> Fold03 <tibble [3 x 4]> <tibble [0 x 3]>
 4 <split [4752/528]> Fold04 <tibble [3 x 4]> <tibble [0 x 3]>
 5 <split [4752/528]> Fold05 <tibble [3 x 4]> <tibble [0 x 3]>
 6 <split [4752/528]> Fold06 <tibble [3 x 4]> <tibble [0 x 3]>
 7 <split [4752/528]> Fold07 <tibble [3 x 4]> <tibble [0 x 3]>
 8 <split [4752/528]> Fold08 <tibble [3 x 4]> <tibble [0 x 3]>
 9 <split [4752/528]> Fold09 <tibble [3 x 4]> <tibble [0 x 3]>
10 <split [4752/528]> Fold10 <tibble [3 x 4]> <tibble [0 x 3]>
```

```r
# Extract single components from folds
  fit1$.metrics[1:2] # get first two rows of .metrics
```

```
[[1]]
# A tibble: 3 x 4
  .metric     .estimator .estimate .config
  <chr>       <chr>          <dbl> <chr>
1 accuracy    binary         0.661 Preprocessor1_Model1
2 roc_auc     binary         0.707 Preprocessor1_Model1
```

```
3 brier_class binary         0.219 Preprocessor1_Model1

[[2]]
# A tibble: 3 x 4
  .metric     .estimator .estimate .config
  <chr>       <chr>          <dbl> <chr>
1 accuracy    binary         0.693 Preprocessor1_Model1
2 roc_auc     binary         0.749 Preprocessor1_Model1
3 brier_class binary         0.206 Preprocessor1_Model1
```

```r
# 6.
# Collect metrics across resamples
  collect_metrics(fit1)
```

```
# A tibble: 3 x 6
  .metric     .estimator  mean     n std_err .config
  <chr>       <chr>      <dbl> <int>   <dbl> <chr>
1 accuracy    binary     0.678    10 0.00503 Preprocessor1_Model1
2 brier_class binary     0.211    10 0.00208 Preprocessor1_Model1
3 roc_auc     binary     0.732    10 0.00582 Preprocessor1_Model1
```

If we are happy with the average accuracy of our model of 0.68 across resamples we would then use the same model defined above, fit it on the whole (non-splitted) training dataset and create a new fitted model `fit2`. We can evaluate the accuracy of that new model in the training set and then move to the test data further below.

```r
# 7.
  # Fit on training dataset (fit2!!!)
  fit2 <- workflow1 %>% fit(data = data_train)


# Training data: Add predictions
  data_train <- augment(fit2, data_train, type.predict = "response")

  head(data_train %>%
      select(is_recid_factor, age, .pred_class,
             .pred_no, .pred_yes))
```

```
# A tibble: 6 x 5
  is_recid_factor   age .pred_class .pred_no .pred_yes
  <fct>           <dbl> <fct>          <dbl>     <dbl>
```

```
1 yes                  30 yes           0.293    0.707
2 no                   35 no            0.583    0.417
3 yes                  28 no            0.631    0.369
4 no                   37 no            0.644    0.356
5 yes                  37 no            0.527    0.473
6 no                   21 yes           0.413    0.587
```

```r
# Training data: Metrics
  data_train %>%
      metrics(truth = is_recid_factor, estimate = .pred_class)
```

```
# A tibble: 2 x 3
  .metric  .estimator .estimate
  <chr>    <chr>          <dbl>
1 accuracy binary         0.679
2 kap      binary         0.356
```

And finally evaluate the model using the test data (holdout set).

```r
# 8.
# Test data: Add predictions
  data_test <- augment(fit2, data_test, type.predict = "response")

  head(data_test %>%
      select(is_recid_factor, age, .pred_class, .pred_no, .pred_yes))
```

```
# A tibble: 6 x 5
  is_recid_factor   age .pred_class .pred_no .pred_yes
  <fct>           <dbl> <fct>          <dbl>     <dbl>
1 yes                24 yes            0.334     0.666
2 no                 39 no             0.757     0.243
3 yes                64 yes            0.453     0.547
4 no                 21 no             0.564     0.436
5 yes                26 yes            0.320     0.680
6 yes                33 no             0.598     0.402
```

```r
# Test data: Metrics
  data_test %>%
      metrics(truth = is_recid_factor, estimate = .pred_class)
```

```
# A tibble: 2 x 3
  .metric  .estimator .estimate
  <chr>    <chr>          <dbl>
1 accuracy binary         0.645
2 kap      binary         0.287
```

```
# More accuracy metrics
  metrics_combined <- metric_set(accuracy, precision, recall, f_meas)

  # The returned function has arguments:
  data_test %>%
  metrics_combined(truth = is_recid_factor, estimate = .pred_class)
```

```
# A tibble: 4 x 3
  .metric   .estimator .estimate
  <chr>     <chr>          <dbl>
1 accuracy  binary         0.645
2 precision binary         0.647
3 recall    binary         0.686
4 f_meas    binary         0.666
```

```
# Cross-classification table
  conf_mat(data = data_test,
           truth = is_recid_factor, estimate = .pred_class)
```

```
          Truth
Prediction  no yes
       no  468 255
       yes 214 383
```

Visual assessment of accuracy

Figure 1 displays the ROC curve. The corresponding values can be obtained using the `roc_curve()` function.

- Important (`?roc_curve`): "There is no common convention on which factor level should automatically be considered the"event" or "positive" result when computing binary classification metrics. In yardstick, the default is to use the first level. To alter this, change the argument event_level to "second" to consider the last level of the factor the level of interest."

– Here we pick the 1s, i.e., recidivating as the “event” or “positive” result.

```
data_test %>%
  roc_curve(truth = is_recid_factor,
            .pred_yes,
            event_level = "second") %>%
  autoplot() +
  xlab("1 - specificity (FPR = FP/N, false positives rate)") +
  ylab("sensitivity (TPR = TP/P, true positives rate)")
```

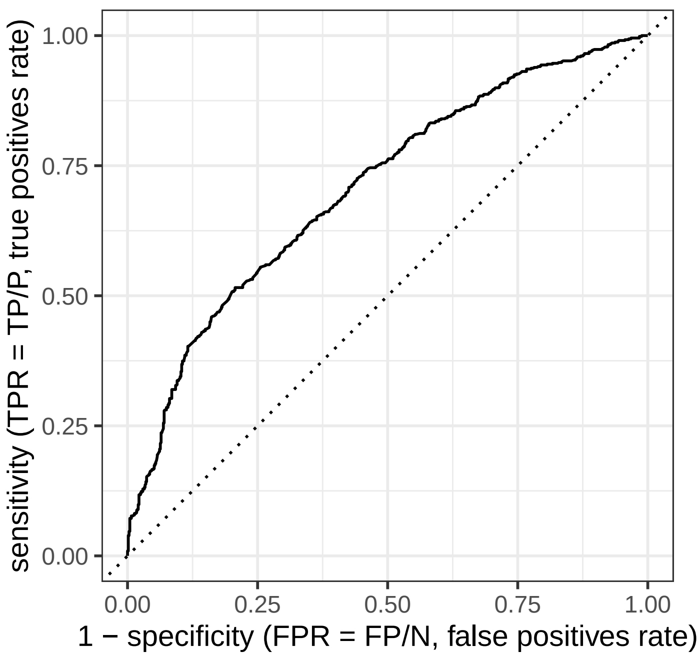


Figure 1: Precision, recall and threshold values

And we can also calculate the area under the ROC curve (the higher the better with 1 being the maximum):

```
# Compute area under ROC curve
data_test %>%
    roc_auc(truth = is_recid_factor,
            .pred_yes,
              event_level = "second")
```

```
# A tibble: 1 x 3
```

```
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 roc_auc binary         0.706
```

Importantly, the ROC curve does not provide information on how FPR and TPR change as a function of the threshold. In Figure 2 we visualize both precision and recall (TPR) as a function of the threshold. The `pr_curve()` function can be used to calculate the corresponding values and we could also change it to FPR vs. TPR.

```
library(ggplot2)
library(dplyr)

data_test %>%
  pr_curve(truth = is_recid_factor,
           .pred_yes,
           event_level = "second") %>%
  pivot_longer(cols = c("recall", "precision"),
               names_to = "measure",
               values_to = "value") %>%
  mutate(measure = recode(measure,
                          "recall" = "Recall (= True pos. rate = TP/P = sensitivity)",
                          "precision" = "Precision (= Pos. pred. value = TP/P*)")) %>%
  ggplot(aes(x = .threshold,
             y = value,
             color = measure)) +
  geom_line() +
  xlab("Threshold value") +
  ylab("Value of measure") +
  theme_bw()
```

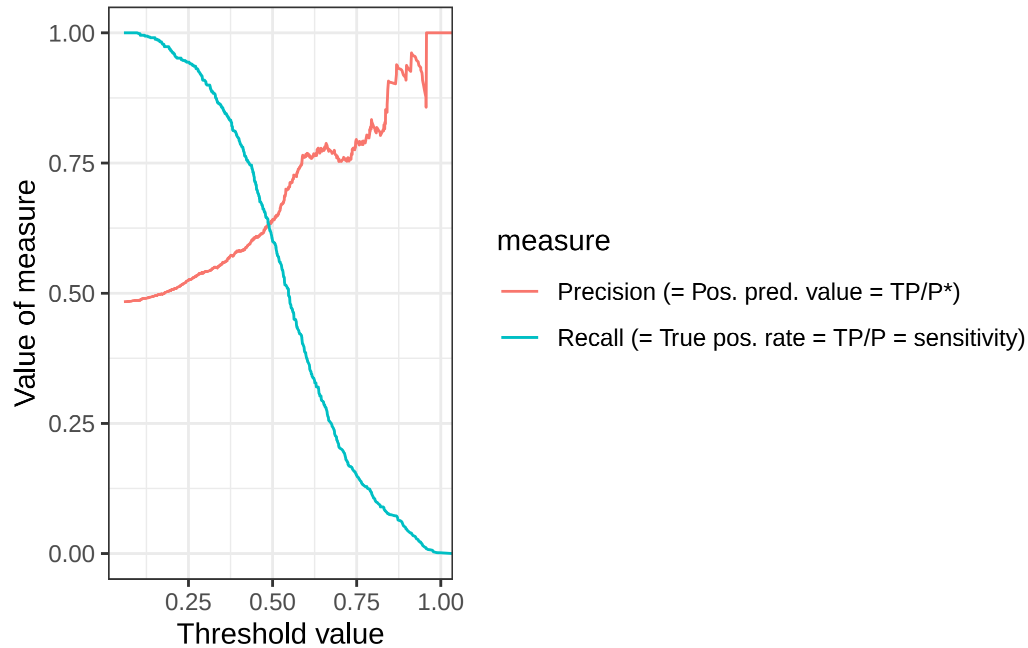


Figure 2: Precision, recall and threshold values

Exercise

1. Start by re-reading the code presented above (which you can find in the chunk below) to see whether everything is clear.
2. Above we re-evaluated the accuracy of our model using 10-fold cross validation. Please re-evaluate the model but now compare the setting where you use k-Fold Cross-Validation using 5 folds (k = 5) and 20 folds (k = 20). Do you find any differences?
3. Keep the same predictors but change the recipe and add polynomials for the numeric variables `age`, `priors_count` and use one_hot encoding for the `race` variable.[1] Does it change the accuracy?
4. Finally, shorty outline the advantages and disadvantages of the *validation set approach* (training/analyis vs. validation/assessment vs. test data), *Leave-one-out cross-validation (LOOCV)* and *k-Fold Cross-Validation* (e.g., discuss dataset sizes, representativeness, computational efficiency).

```
load(file = here("data/data_compas.Rdata"))

# 1.
# Split the data into training and test data
```

[1] `step_poly(all_numeric_predictors(), degree = 4, keep_original_cols = TRUE, options = list(raw = TRUE)) %>% step_dummy(race, one_hot = TRUE)`

```r
  set.seed(345)
  data_split <- initial_split(data, prop = 0.8)
  data_split # Inspect

# Extract the two datasets
  data_train <- training(data_split)
  data_test <- testing(data_split) # Do not touch until the end!

# 2.
# Create resampled partitions of training data
  data_folds <- vfold_cv(data_train, v = 10) # V-fold/k-fold cross-validation
  data_folds # data_folds now contains several resamples of our training data
  # You can also try loo_cv(data_train) instead



# 3.
# Define the recipe & model
  recipe1 <- recipe(is_recid_factor ~ age + priors_count +
                      sex + race, data = data_train) %>%
    step_poly(all_numeric_predictors(), degree = 4,
              keep_original_cols = FALSE,
              options = list(raw = TRUE)) %>%
    step_dummy(race, one_hot = TRUE)

  model1 <- logistic_reg() %>%
    set_engine("glm") %>% # lm engine
    set_mode("classification") # regression problem

# 4.
# Create workflow
  workflow1 <-
  workflow() %>%
  add_model(model1) %>%
  add_recipe(recipe1)

# 5. Fit the workflow on the folds/resamples
  # There is no need in specifying data_analysis/data_assessment as
  # the functions understand the corresponding parts
  fit1 <-
  workflow1 %>%
  fit_resamples(resamples = data_folds,
```

```
                  control = control_resamples(save_pred = TRUE,
                                              extract = function (x) extract_fit_parsnip(x)))
  fit1

# Extract single components from folds
  fit1$.metrics[1:2] # get first two rows of .metrics
  fit1$.extracts [1:2] # get first two rows of .extracts
  fit1$.extracts[[1]]$.extracts # Extract models



# 6.
# Collect metrics across resamples
  collect_metrics(fit1)

# 7.
  # Fit on training dataset (fit2!!!)
  fit2 <- workflow1 %>% fit(data = data_train)


# Training data: Add predictions & get metrics
  augment(fit2, data_train, type.predict = "response")  %>%
      metrics(truth = is_recid_factor, estimate = .pred_class)




# 8.
# Test data: Add predictions & get metrics
  augment(fit2, data_test, type.predict = "response") %>%
      metrics(truth = is_recid_factor, estimate = .pred_class)

# More accuracy metrics
  metrics_combined <- metric_set(accuracy, precision, recall, f_meas)

  augment(fit2, data_test, type.predict = "response") %>%
  metrics_combined(truth = is_recid_factor, estimate = .pred_class)

# Cross-classification table
  augment(fit2, data_test, type.predict = "response") %>%
      conf_mat(data = .,
               truth = is_recid_factor, estimate = .pred_class)
```

3. Feature selection & regularization

Ridge Regression

We first import the data into R:

```
load(file = here("data/data_ess.Rdata"))
```

Below we split the data and create resampled partitions with `vfold_cv()` stored in an object called `data_folds`. Hence, we have the original `data_train` and `data_test` but also already resamples of `data_train` in `data_folds`.

```
# Extract data with missing outcome
  data_missing_outcome <- data %>% filter(is.na(life_satisfaction))
  dim(data_missing_outcome)
```

```
[1] 205 346
```

```
# Omit individuals with missing outcome from data
  data <- data %>% drop_na(life_satisfaction) # ?drop_na
  dim(data)
```

```
[1] 1772  346
```

```
# Split the data into training and test data
  set.seed(345)
  data_split <- initial_split(data, prop = 0.80)
  data_split # Inspect
```

```
<Training/Testing/Total>
<1417/355/1772>
```

```
# Extract the two datasets
  data_train <- training(data_split)
  data_test <- testing(data_split) # Do not touch until the end!

# Create resampled partitions
  data_folds <- vfold_cv(data_train, v = 5) # V-fold/k-fold cross-validation
  data_folds # data_folds now contains several resamples of our training data
```

```
#  5-fold cross-validation
# A tibble: 5 x 2
  splits             id
  <list>             <chr>
1 <split [1133/284]> Fold1
2 <split [1133/284]> Fold2
3 <split [1134/283]> Fold3
4 <split [1134/283]> Fold4
5 <split [1134/283]> Fold5
```

```
  # You can also try loo_cv(data_train) instead
```

The training data has 1417 rows, the test data has 355. The training data is further split into 10 folds.

Next, we provide a quick example of a ridge regression (beware: below in the recipe we standardize predictors).

- Arguments of `glmnet` `linear_reg()`
  - `mixture = 0`: to specify a ridge model
    - `mixture = 0` specifies only ridge regularization
    - `mixture = 1` specifies only lasso regularization
    - Setting mixture to a value between 0 and 1 lets us use both
  - `penalty = 0`: Penality we need to set when using `glmnet` engine (for now 0)

```
# Define a recipe for preprocessing
  recipe1 <- recipe(life_satisfaction ~ ., data = data_train) %>%
    update_role(respondent_id, new_role = "ID") %>% # define as ID variable
    step_filter_missing(all_predictors(), threshold = 0.01) %>%
    step_naomit(all_predictors()) %>%
    step_zv(all_numeric_predictors()) %>% # remove predictors with zero variance
    step_normalize(all_numeric_predictors()) %>% # normalize predictors
    step_dummy(all_nominal_predictors())

  # Extract and preview data + recipe (direclty with $)
  data_preprocessed <- prep(recipe1, data_train)$template
  dim(data_preprocessed)
```

```
[1] 1258  208
```

```r
# Define a model
  model1 <- linear_reg(mixture = 0, penalty = 2) %>% # ridge regression model
    set_engine("lm") %>% # lm engine
    set_mode("regression") # regression problem

# Define a workflow
  workflow1 <- workflow() %>% # create empty workflow
    add_recipe(recipe1) %>% # add recipe
    add_model(model1) # add model

# Fit the workflow (including recipe and model)
  fit1 <- workflow1 %>%
    fit_resamples(resamples = data_folds, # specify cv-datasets
    metrics = metric_set(rmse, rsq, mae))
```

```
> A | warning: ! There are new levels in `political_interest`: NA.
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
                 `step_dummy()` to handle missing values., ! There are new levels in `childre
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
                 `step_dummy()` to handle missing values., ! There are new levels in `discuss
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
                 `step_dummy()` to handle missing values., ! There are new levels in `social_
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
                 `step_dummy()` to handle missing values., ! There are new levels in `religio
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
                 `step_dummy()` to handle missing values., ! There are new levels in `climate
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
                 `step_dummy()` to handle missing values., ! There are new levels in `climate
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
                 `step_dummy()` to handle missing values., ! There are new levels in `mnactic
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
                 `step_dummy()` to handle missing values., ! There are new levels in `trade_u
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
                 `step_dummy()` to handle missing values., ! There are new levels in `feeling
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
                 `step_dummy()` to handle missing values., ! There are new levels in `parents
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
                 `step_dummy()` to handle missing values., ! There are new levels in `value_w
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
                 `step_dummy()` to handle missing values., ! There are new levels in `value_s
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
                 `step_dummy()` to handle missing values., prediction from a rank-deficient f
```

```
There were issues with some computations   A: x1

> B | warning: ! There are new levels in `children_learns_obedience`: NA.
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
                 `step_dummy()` to handle missing values., ! There are new levels in `social_
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
                 `step_dummy()` to handle missing values., ! There are new levels in `discuss
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
                 `step_dummy()` to handle missing values., ! There are new levels in `subject
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
                 `step_dummy()` to handle missing values., ! There are new levels in `daily_a
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
                 `step_dummy()` to handle missing values., ! There are new levels in `religio
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
                 `step_dummy()` to handle missing values., ! There are new levels in `climate
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
                 `step_dummy()` to handle missing values., ! There are new levels in `climate
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
                 `step_dummy()` to handle missing values., ! There are new levels in `climate
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
                 `step_dummy()` to handle missing values., ! There are new levels in `marital
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
                 `step_dummy()` to handle missing values., ! There are new levels in `mnactio
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
                 `step_dummy()` to handle missing values., ! There are new levels in `trade_u
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
                 `step_dummy()` to handle missing values., ! There are new levels in `househo
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
                 `step_dummy()` to handle missing values., ! There are new levels in `parents
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
                 `step_dummy()` to handle missing values., ! There are new levels in `value_s
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
                 `step_dummy()` to handle missing values., ! There are new levels in `value_t
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
                 `step_dummy()` to handle missing values., ! There are new levels in `value_u
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
                 `step_dummy()` to handle missing values., ! There are new levels in `value_l
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
                 `step_dummy()` to handle missing values., prediction from a rank-deficient f

There were issues with some computations   A: x1
> C | warning: ! There are new levels in `children_learns_obedience`: NA.
```

```
            i Consider using step_unknown() (`?recipes::step_unknown()`) before
              `step_dummy()` to handle missing values., ! There are new levels in `discuss
            i Consider using step_unknown() (`?recipes::step_unknown()`) before
              `step_dummy()` to handle missing values., ! There are new levels in `social_
            i Consider using step_unknown() (`?recipes::step_unknown()`) before
              `step_dummy()` to handle missing values., ! There are new levels in `daily_a
            i Consider using step_unknown() (`?recipes::step_unknown()`) before
              `step_dummy()` to handle missing values., ! There are new levels in `religio
            i Consider using step_unknown() (`?recipes::step_unknown()`) before
              `step_dummy()` to handle missing values., ! There are new levels in `prayer_
            i Consider using step_unknown() (`?recipes::step_unknown()`) before
              `step_dummy()` to handle missing values., ! There are new levels in `climate
            i Consider using step_unknown() (`?recipes::step_unknown()`) before
              `step_dummy()` to handle missing values., ! There are new levels in `marital
            i Consider using step_unknown() (`?recipes::step_unknown()`) before
              `step_dummy()` to handle missing values., ! There are new levels in `mnactic
            i Consider using step_unknown() (`?recipes::step_unknown()`) before
              `step_dummy()` to handle missing values., ! There are new levels in `trade_u
            i Consider using step_unknown() (`?recipes::step_unknown()`) before
              `step_dummy()` to handle missing values., ! There are new levels in `feeling
            i Consider using step_unknown() (`?recipes::step_unknown()`) before
              `step_dummy()` to handle missing values., ! There are new levels in `parents
            i Consider using step_unknown() (`?recipes::step_unknown()`) before
              `step_dummy()` to handle missing values., ! There are new levels in `value_w
            i Consider using step_unknown() (`?recipes::step_unknown()`) before
              `step_dummy()` to handle missing values., ! There are new levels in `value_e
            i Consider using step_unknown() (`?recipes::step_unknown()`) before
              `step_dummy()` to handle missing values., ! There are new levels in `value_s
            i Consider using step_unknown() (`?recipes::step_unknown()`) before
              `step_dummy()` to handle missing values., ! There are new levels in `value_t
            i Consider using step_unknown() (`?recipes::step_unknown()`) before
              `step_dummy()` to handle missing values., ! There are new levels in `value_u
            i Consider using step_unknown() (`?recipes::step_unknown()`) before
              `step_dummy()` to handle missing values., ! There are new levels in `value_l
            i Consider using step_unknown() (`?recipes::step_unknown()`) before
              `step_dummy()` to handle missing values., ! There are new levels in `value_i
            i Consider using step_unknown() (`?recipes::step_unknown()`) before
              `step_dummy()` to handle missing values., prediction from a rank-deficient f
There were issues with some computations   A: x1
There were issues with some computations   A: x1   B: x1   C: x1
> D | warning: ! There are new levels in `family_member_gay_shame`: NA.
            i Consider using step_unknown() (`?recipes::step_unknown()`) before
              `step_dummy()` to handle missing values., ! There are new levels in `childre
```

```
                i Consider using step_unknown() (`?recipes::step_unknown()`) before
                  `step_dummy()` to handle missing values., ! There are new levels in `social_
                i Consider using step_unknown() (`?recipes::step_unknown()`) before
                  `step_dummy()` to handle missing values., ! There are new levels in `climate
                i Consider using step_unknown() (`?recipes::step_unknown()`) before
                  `step_dummy()` to handle missing values., ! There are new levels in `climate
                i Consider using step_unknown() (`?recipes::step_unknown()`) before
                  `step_dummy()` to handle missing values., ! There are new levels in `mnacti
                i Consider using step_unknown() (`?recipes::step_unknown()`) before
                  `step_dummy()` to handle missing values., ! There are new levels in `trade_
                i Consider using step_unknown() (`?recipes::step_unknown()`) before
                  `step_dummy()` to handle missing values., ! There are new levels in `parent
                i Consider using step_unknown() (`?recipes::step_unknown()`) before
                  `step_dummy()` to handle missing values., ! There are new levels in `value_
                i Consider using step_unknown() (`?recipes::step_unknown()`) before
                  `step_dummy()` to handle missing values., ! There are new levels in `value_
                i Consider using step_unknown() (`?recipes::step_unknown()`) before
                  `step_dummy()` to handle missing values., ! There are new levels in `value_
                i Consider using step_unknown() (`?recipes::step_unknown()`) before
                  `step_dummy()` to handle missing values., ! There are new levels in `value_
                i Consider using step_unknown() (`?recipes::step_unknown()`) before
                  `step_dummy()` to handle missing values., ! There are new levels in `value_
                i Consider using step_unknown() (`?recipes::step_unknown()`) before
                  `step_dummy()` to handle missing values., ! There are new levels in `value_
                i Consider using step_unknown() (`?recipes::step_unknown()`) before
                  `step_dummy()` to handle missing values., ! There are new levels in `value_
                i Consider using step_unknown() (`?recipes::step_unknown()`) before
                  `step_dummy()` to handle missing values., ! There are new levels in `value_
                i Consider using step_unknown() (`?recipes::step_unknown()`) before
                  `step_dummy()` to handle missing values., ! There are new levels in `value_
                i Consider using step_unknown() (`?recipes::step_unknown()`) before
                  `step_dummy()` to handle missing values., prediction from a rank-deficient
There were issues with some computations   A: x1   B: x1   C: x1
> E | warning: ! There are new levels in `political_interest`: NA.
                i Consider using step_unknown() (`?recipes::step_unknown()`) before
                  `step_dummy()` to handle missing values., ! There are new levels in `active_
                i Consider using step_unknown() (`?recipes::step_unknown()`) before
                  `step_dummy()` to handle missing values., ! There are new levels in `social_
                i Consider using step_unknown() (`?recipes::step_unknown()`) before
                  `step_dummy()` to handle missing values., ! There are new levels in `discus
                i Consider using step_unknown() (`?recipes::step_unknown()`) before
                  `step_dummy()` to handle missing values., ! There are new levels in `social_
                i Consider using step_unknown() (`?recipes::step_unknown()`) before
```

```
                 `step_dummy()` to handle missing values., ! There are new levels in `religio
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
                 `step_dummy()` to handle missing values., ! There are new levels in `climate
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
                 `step_dummy()` to handle missing values., ! There are new levels in `climate
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
                 `step_dummy()` to handle missing values., ! There are new levels in `mnactio
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
                 `step_dummy()` to handle missing values., ! There are new levels in `trade_u
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
                 `step_dummy()` to handle missing values., ! There are new levels in `parents
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
                 `step_dummy()` to handle missing values., ! There are new levels in `value_w
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
                 `step_dummy()` to handle missing values., ! There are new levels in `value_c
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
                 `step_dummy()` to handle missing values., ! There are new levels in `value_s
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
                 `step_dummy()` to handle missing values., ! There are new levels in `value_t
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
                 `step_dummy()` to handle missing values., ! There are new levels in `value_u
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
                 `step_dummy()` to handle missing values., ! There are new levels in `value_l
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
                 `step_dummy()` to handle missing values., ! There are new levels in `value_i
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
                 `step_dummy()` to handle missing values., prediction from a rank-deficient f
There were issues with some computations   A: x1   B: x1   C: x1
There were issues with some computations   A: x1   B: x1   C: x1   D: x1   E: x1
```

```r
collect_metrics(fit1)
```

```
# A tibble: 3 x 6
  .metric .estimator  mean     n std_err .config
  <chr>   <chr>      <dbl> <int>   <dbl> <chr>
1 mae     standard   1.26      5  0.0222 Preprocessor1_Model1
2 rmse    standard   1.71      5  0.0461 Preprocessor1_Model1
3 rsq     standard   0.424     5  0.0189 Preprocessor1_Model1
```

If we are happy with the performance of our model (evaluated using resampling), we can fit it on the full training set and use the resulting parameters to obtain predictions in the test dataset. Subsequently we calculate the accuracy in the test data.

```
# Fit model in full training dataset
  fit_final <- workflow1 %>%
              parsnip::fit(data = data_train)
  # Inspect coefficients: tidy(fit_final)

# Test data: Predictions + accuracy
  metrics_combined <- metric_set(mae, rmse, rsq) # Use several metrics

  augment(fit_final , new_data = data_test) %>%
    metrics_combined(truth = life_satisfaction, estimate = .pred)
```

```
# A tibble: 3 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 mae     standard       1.21
2 rmse    standard       1.70
3 rsq     standard       0.379
```

Lasso

We first import the data into R:

```
load(file = here("data/data_ess.Rdata"))
```

Below we split the data and create resampled partitions with `vfold_cv()` stored in an object called `data_folds`. Hence, we have the original `data_train` and `data_test` but also already resamples of `data_train` in `data_folds`.

```
# Extract data with missing outcome
  data_missing_outcome <- data %>% filter(is.na(life_satisfaction))
  dim(data_missing_outcome)
```

```
[1] 205 346
```

```
# Omit individuals with missing outcome from data
  data <- data %>% drop_na(life_satisfaction) # ?drop_na
  dim(data)
```

```
[1] 1772  346
```

```r
# Split the data into training and test data
  set.seed(345)
  data_split <- initial_split(data, prop = 0.80)
  data_split # Inspect
```

```
<Training/Testing/Total>
<1417/355/1772>
```

```r
# Extract the two datasets
  data_train <- training(data_split)
  data_test <- testing(data_split) # Do not touch until the end!

# Create resampled partitions
  data_folds <- vfold_cv(data_train, v = 5) # V-fold/k-fold cross-validation
  data_folds # data_folds now contains several resamples of our training data
```

```
#  5-fold cross-validation
# A tibble: 5 x 2
  splits             id
  <list>             <chr>
1 <split [1133/284]> Fold1
2 <split [1133/284]> Fold2
3 <split [1134/283]> Fold3
4 <split [1134/283]> Fold4
5 <split [1134/283]> Fold5
```

```r
# Define the recipe
recipe2 <- recipe(life_satisfaction ~ ., data = data_train) %>%
    update_role(respondent_id, new_role = "ID") %>% # define as ID variable
    step_filter_missing(all_predictors(), threshold = 0.01) %>%
    step_naomit(all_predictors()) %>%
    step_zv(all_numeric_predictors()) %>% # remove predictors with zero variance
    step_normalize(all_numeric_predictors()) %>% # normalize predictors
    step_dummy(all_nominal_predictors())

# Specify the model
  model2 <-
    linear_reg(penalty = 0.1, mixture = 1) %>%
    set_mode("regression") %>%
    set_engine("glmnet")
```

```
# Define the workflow
  workflow2 <- workflow() %>%
    add_recipe(recipe2) %>%
    add_model(model2)


# Fit the workflow (including recipe and model)
  fit2 <- workflow2 %>%
    fit_resamples(resamples = data_folds, # specify cv-datasets
    metrics = metric_set(rmse, rsq, mae))
```

```
> A | warning: ! There are new levels in `political_interest`: NA.
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
                 `step_dummy()` to handle missing values., ! There are new levels in `childre
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
                 `step_dummy()` to handle missing values., ! There are new levels in `discuss
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
                 `step_dummy()` to handle missing values., ! There are new levels in `social_
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
                 `step_dummy()` to handle missing values., ! There are new levels in `religio
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
                 `step_dummy()` to handle missing values., ! There are new levels in `climate
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
                 `step_dummy()` to handle missing values., ! There are new levels in `climate
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
                 `step_dummy()` to handle missing values., ! There are new levels in `mnactio
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
                 `step_dummy()` to handle missing values., ! There are new levels in `trade_u
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
                 `step_dummy()` to handle missing values., ! There are new levels in `feeling
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
                 `step_dummy()` to handle missing values., ! There are new levels in `parents
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
                 `step_dummy()` to handle missing values., ! There are new levels in `value_w
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
                 `step_dummy()` to handle missing values., ! There are new levels in `value_s
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
                 `step_dummy()` to handle missing values.

> B | warning: ! There are new levels in `children_learns_obedience`: NA.
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
```

```
                `step_dummy()` to handle missing values., ! There are new levels in `social_
              i Consider using step_unknown() (`?recipes::step_unknown()`) before
                `step_dummy()` to handle missing values., ! There are new levels in `discuss
              i Consider using step_unknown() (`?recipes::step_unknown()`) before
                `step_dummy()` to handle missing values., ! There are new levels in `subject
              i Consider using step_unknown() (`?recipes::step_unknown()`) before
                `step_dummy()` to handle missing values., ! There are new levels in `daily_a
              i Consider using step_unknown() (`?recipes::step_unknown()`) before
                `step_dummy()` to handle missing values., ! There are new levels in `religio
              i Consider using step_unknown() (`?recipes::step_unknown()`) before
                `step_dummy()` to handle missing values., ! There are new levels in `climate
              i Consider using step_unknown() (`?recipes::step_unknown()`) before
                `step_dummy()` to handle missing values., ! There are new levels in `climate
              i Consider using step_unknown() (`?recipes::step_unknown()`) before
                `step_dummy()` to handle missing values., ! There are new levels in `climate
              i Consider using step_unknown() (`?recipes::step_unknown()`) before
                `step_dummy()` to handle missing values., ! There are new levels in `marital
              i Consider using step_unknown() (`?recipes::step_unknown()`) before
                `step_dummy()` to handle missing values., ! There are new levels in `mnactio
              i Consider using step_unknown() (`?recipes::step_unknown()`) before
                `step_dummy()` to handle missing values., ! There are new levels in `trade_u
              i Consider using step_unknown() (`?recipes::step_unknown()`) before
                `step_dummy()` to handle missing values., ! There are new levels in `househo
              i Consider using step_unknown() (`?recipes::step_unknown()`) before
                `step_dummy()` to handle missing values., ! There are new levels in `parents
              i Consider using step_unknown() (`?recipes::step_unknown()`) before
                `step_dummy()` to handle missing values., ! There are new levels in `value_s
              i Consider using step_unknown() (`?recipes::step_unknown()`) before
                `step_dummy()` to handle missing values., ! There are new levels in `value_t
              i Consider using step_unknown() (`?recipes::step_unknown()`) before
                `step_dummy()` to handle missing values., ! There are new levels in `value_u
              i Consider using step_unknown() (`?recipes::step_unknown()`) before
                `step_dummy()` to handle missing values., ! There are new levels in `value_l
              i Consider using step_unknown() (`?recipes::step_unknown()`) before
                `step_dummy()` to handle missing values.

There were issues with some computations   A: x1   B: x1

> C | warning: ! There are new levels in `children_learns_obedience`: NA.
              i Consider using step_unknown() (`?recipes::step_unknown()`) before
                `step_dummy()` to handle missing values., ! There are new levels in `discuss
              i Consider using step_unknown() (`?recipes::step_unknown()`) before
```

```
    `step_dummy()` to handle missing values., ! There are new levels in `social_
  i Consider using step_unknown() (`?recipes::step_unknown()`) before
    `step_dummy()` to handle missing values., ! There are new levels in `daily_a
  i Consider using step_unknown() (`?recipes::step_unknown()`) before
    `step_dummy()` to handle missing values., ! There are new levels in `religio
  i Consider using step_unknown() (`?recipes::step_unknown()`) before
    `step_dummy()` to handle missing values., ! There are new levels in `prayer_
  i Consider using step_unknown() (`?recipes::step_unknown()`) before
    `step_dummy()` to handle missing values., ! There are new levels in `climate
  i Consider using step_unknown() (`?recipes::step_unknown()`) before
    `step_dummy()` to handle missing values., ! There are new levels in `marital
  i Consider using step_unknown() (`?recipes::step_unknown()`) before
    `step_dummy()` to handle missing values., ! There are new levels in `mnactic
  i Consider using step_unknown() (`?recipes::step_unknown()`) before
    `step_dummy()` to handle missing values., ! There are new levels in `trade_u
  i Consider using step_unknown() (`?recipes::step_unknown()`) before
    `step_dummy()` to handle missing values., ! There are new levels in `feeling
  i Consider using step_unknown() (`?recipes::step_unknown()`) before
    `step_dummy()` to handle missing values., ! There are new levels in `parents
  i Consider using step_unknown() (`?recipes::step_unknown()`) before
    `step_dummy()` to handle missing values., ! There are new levels in `value_w
  i Consider using step_unknown() (`?recipes::step_unknown()`) before
    `step_dummy()` to handle missing values., ! There are new levels in `value_e
  i Consider using step_unknown() (`?recipes::step_unknown()`) before
    `step_dummy()` to handle missing values., ! There are new levels in `value_s
  i Consider using step_unknown() (`?recipes::step_unknown()`) before
    `step_dummy()` to handle missing values., ! There are new levels in `value_t
  i Consider using step_unknown() (`?recipes::step_unknown()`) before
    `step_dummy()` to handle missing values., ! There are new levels in `value_u
  i Consider using step_unknown() (`?recipes::step_unknown()`) before
    `step_dummy()` to handle missing values., ! There are new levels in `value_l
  i Consider using step_unknown() (`?recipes::step_unknown()`) before
    `step_dummy()` to handle missing values., ! There are new levels in `value_i
  i Consider using step_unknown() (`?recipes::step_unknown()`) before
    `step_dummy()` to handle missing values.

There were issues with some computations   A: x1   B: x1
> D | warning: ! There are new levels in `family_member_gay_shame`: NA.
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
                 `step_dummy()` to handle missing values., ! There are new levels in `childre
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
                 `step_dummy()` to handle missing values., ! There are new levels in `social_
```

```
                i Consider using step_unknown() (`?recipes::step_unknown()`) before
                  `step_dummy()` to handle missing values., ! There are new levels in `climate
                i Consider using step_unknown() (`?recipes::step_unknown()`) before
                  `step_dummy()` to handle missing values., ! There are new levels in `climate
                i Consider using step_unknown() (`?recipes::step_unknown()`) before
                  `step_dummy()` to handle missing values., ! There are new levels in `mnactio
                i Consider using step_unknown() (`?recipes::step_unknown()`) before
                  `step_dummy()` to handle missing values., ! There are new levels in `trade_u
                i Consider using step_unknown() (`?recipes::step_unknown()`) before
                  `step_dummy()` to handle missing values., ! There are new levels in `parents
                i Consider using step_unknown() (`?recipes::step_unknown()`) before
                  `step_dummy()` to handle missing values., ! There are new levels in `value_w
                i Consider using step_unknown() (`?recipes::step_unknown()`) before
                  `step_dummy()` to handle missing values., ! There are new levels in `value_e
                i Consider using step_unknown() (`?recipes::step_unknown()`) before
                  `step_dummy()` to handle missing values., ! There are new levels in `value_s
                i Consider using step_unknown() (`?recipes::step_unknown()`) before
                  `step_dummy()` to handle missing values., ! There are new levels in `value_t
                i Consider using step_unknown() (`?recipes::step_unknown()`) before
                  `step_dummy()` to handle missing values., ! There are new levels in `value_u
                i Consider using step_unknown() (`?recipes::step_unknown()`) before
                  `step_dummy()` to handle missing values., ! There are new levels in `value_c
                i Consider using step_unknown() (`?recipes::step_unknown()`) before
                  `step_dummy()` to handle missing values., ! There are new levels in `value_l
                i Consider using step_unknown() (`?recipes::step_unknown()`) before
                  `step_dummy()` to handle missing values., ! There are new levels in `value_l
                i Consider using step_unknown() (`?recipes::step_unknown()`) before
                  `step_dummy()` to handle missing values., ! There are new levels in `value_i
                i Consider using step_unknown() (`?recipes::step_unknown()`) before
                  `step_dummy()` to handle missing values.
There were issues with some computations   A: x1   B: x1
> E | warning: ! There are new levels in `political_interest`: NA.
                i Consider using step_unknown() (`?recipes::step_unknown()`) before
                  `step_dummy()` to handle missing values., ! There are new levels in `active_
                i Consider using step_unknown() (`?recipes::step_unknown()`) before
                  `step_dummy()` to handle missing values., ! There are new levels in `social_
                i Consider using step_unknown() (`?recipes::step_unknown()`) before
                  `step_dummy()` to handle missing values., ! There are new levels in `discuss
                i Consider using step_unknown() (`?recipes::step_unknown()`) before
                  `step_dummy()` to handle missing values., ! There are new levels in `social_
                i Consider using step_unknown() (`?recipes::step_unknown()`) before
                  `step_dummy()` to handle missing values., ! There are new levels in `religio
                i Consider using step_unknown() (`?recipes::step_unknown()`) before
```

```
                 `step_dummy()` to handle missing values., ! There are new levels in `climate
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
                 `step_dummy()` to handle missing values., ! There are new levels in `climate
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
                 `step_dummy()` to handle missing values., ! There are new levels in `mnactic
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
                 `step_dummy()` to handle missing values., ! There are new levels in `trade_u
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
                 `step_dummy()` to handle missing values., ! There are new levels in `parents
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
                 `step_dummy()` to handle missing values., ! There are new levels in `value_v
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
                 `step_dummy()` to handle missing values., ! There are new levels in `value_c
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
                 `step_dummy()` to handle missing values., ! There are new levels in `value_s
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
                 `step_dummy()` to handle missing values., ! There are new levels in `value_t
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
                 `step_dummy()` to handle missing values., ! There are new levels in `value_u
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
                 `step_dummy()` to handle missing values., ! There are new levels in `value_l
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
                 `step_dummy()` to handle missing values., ! There are new levels in `value_i
               i Consider using step_unknown() (`?recipes::step_unknown()`) before
                 `step_dummy()` to handle missing values.
There were issues with some computations   A: x1   B: x1
There were issues with some computations   A: x1   B: x1   C: x1   D: x1   E: x1
```

```r
collect_metrics(fit2)
```

```
# A tibble: 3 x 6
  .metric .estimator  mean     n std_err .config
  <chr>   <chr>      <dbl> <int>   <dbl> <chr>
1 mae     standard   1.13      5  0.0230 Preprocessor1_Model1
2 rmse    standard   1.58      5  0.0289 Preprocessor1_Model1
3 rsq     standard   0.499     5  0.0158 Preprocessor1_Model1
```

```r
# Fit model in full training dataset
  fit_final <- workflow2 %>%
              parsnip::fit(data = data_train)
```

```r
# Inspect coefficients
  tidy(fit_final) %>% filter(estimate!=0)
```

```
# A tibble: 21 x 3
   term                           estimate penalty
   <chr>                             <dbl>   <dbl>
 1 (Intercept)                    6.51e+ 0     0.1
 2 people_fair                    1.68e- 1     0.1
 3 trust_legal_system             9.44e- 2     0.1
 4 trust_police                   1.08e- 1     0.1
 5 state_health_services          1.01e- 1     0.1
 6 happiness                      1.03e+ 0     0.1
 7 attachment_country             4.26e- 2     0.1
 8 discrimination_group_membership  2.18e- 4     0.1
 9 discrimination_not_applicable  6.28e-18     0.1
10 female                        -7.14e- 3     0.1
# i 11 more rows
```

```r
# Test data: Predictions + accuracy
  metrics_combined <- metric_set(mae, rmse, rsq) # Use several metrics

  augment(fit_final , new_data = data_test) %>%
    metrics_combined(truth = life_satisfaction, estimate = .pred)
```

```
# A tibble: 3 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 mae     standard       1.12
2 rmse    standard       1.59
3 rsq     standard       0.447
```

Exercise

Revise the code chunk below by replacing all ...

1. Use the data from above (European Social Survey (ESS)). Use the code below to load it, drop missings and to produce training, test as well as resampled data.
2. Define three models (`model1` = linear regression, `model2` = ridge regression, `model3` = lasso regression) and create two recipes, `recipe1` for the linear regression (the model should only include three predictors: `unemployed + age + education`) and `recipe2`

for the other two models. Store the three models and two recipes in three workflows `workflow1`, `workflow2`, `workflow3`

3. Train these three workflows and evaluate their accuracy using resampling (below some code to get you started).
4. Pick the best workflow (and corresponding model), fit it on the whole training data and evaluate the accuracy on the test data.

We first import the data into R:

```
load(file = here("data/data_ess.Rdata"))
```

```
# 1. ####
# Drop missings on outcome
  data <- data %>%
    drop_na(life_satisfaction) %>% # Drop missings on life satisfaction
    select(where(~mean(is.na(.)) < 0.1)) %>% # keep vars with lower than 10% missing
    na.omit()

# Split the data into training and test data
  data_split <- initial_split(data, prop = 0.80)
  data_split # Inspect

# Extract the two datasets
  data_train <- training(data_split)
  data_... <- testing(data_split) # Do not touch until the end!

# Create resampled partitions
  set.seed(345)
  data_folds <- vfold_cv(..., v = 5) # V-fold/k-fold cross-validation
  data_folds # data_folds now contains several resamples of our training data




# 2. ####
# RECIPES
  recipe1 <- recipe(life_satisfaction ~ unemployed + age + education, data = data_train) %>%
    step_zv(all_predictors()) %>% # remove predictors with zero variance
    step_normalize(all_numeric_predictors())  %>% # normalize predictors
    step_dummy(all_nominal_predictors())


  recipe2 <- recipe(life_satisfaction ~ ., data = ...) %>%
    update_role(respondent_id, new_role = "ID") %>% # define as ID variable
```

```
    step_zv(all_predictors()) %>% # remove predictors with zero variance
    step_normalize(all_numeric_predictors()) %>% # normalize predictors
    step_dummy(all_nominal_predictors())

# MODELS
  model1 <- linear_reg() %>% # linear model
    set_engine("lm") %>% # lm engine
    set_mode("regression") # regression problem

  model2 <- linear_reg(mixture = 0, penalty = 0) %>% # ridge regression model
    set_engine("glmnet") %>%
    set_mode("regression") # regression problem

  model3 <-
    linear_reg(penalty = 0.05, mixture = 1) %>%
    ... %>%
    ...


# WORKFLOWS
  workflow1 <- workflow() %>%
                add_recipe(recipe1) %>%
                add_model(model1)

  workflow2 <- ...

  workflow3 <- workflow() %>%
                add_recipe(recipe2) %>%
                add_model(model3)


# 3. ####
# TRAINING/FITTING
  fit1 <- workflow1 %>% fit_resamples(resamples = data_folds)
  fit2 <- ...
  fit3 <- ...

# RESAMPLED DATA: COLLECT METRICS
  collect_metrics(fit1)
  collect_metrics(...)
  collect_metrics(...)
  # Lasso performed best!!
```

```r
# FIT LASSO TO FULL TRAINING DATA
  fit_lasso <- ... %>% parsnip::fit(data = ...)

# Metrics: Training data
  metrics_combined <- metric_set(rsq, rmse, mae)

  augment(fit_lasso, new_data = ...) %>%
    metrics_combined(truth = life_satisfaction, estimate = .pred)

# Metrics: Test data
  augment(fit_lasso, new_data = ...) %>%
    metrics_combined(truth = life_satisfaction, estimate = ...)
```

4. Tree-based methods

- Below the steps we would pursue WITHOUT tuning our random forest.
  - **Step 1**: Load data, recode and rename variables
  - **Step 2**: Split the data
  - **Step 3**: Specify recipe, model and workflow
  - **Step 4**: Evaluate model using resampling
  - **Step 5**: Fit final model to full training data and assess accuracy
  - **Step 6**: Fit final model to test data and assess accuracy

Step 1: Loading, renaming and recoding

We first import the data into R:

```r
load(file = here("data/data_ess.Rdata"))
```

Step 2: Prepare & split the data

Below we split the data and create resampled partitions with `vfold_cv()` stored in an object called `data_folds`. Hence, we have the original `data_train` and `data_test` but also already resamples of `data_train` in `data_folds`.

```
# Take subset of variables to speed things up!
  data <- data  %>%
    select(life_satisfaction,
           unemployed,
           trust_people,
           education,
           age,
           religion,
           subjective_health) %>%
  mutate(religion = if_else(is.na(religion), "unknown", religion),
         religion = as.factor(religion)) %>%
    drop_na()


# Extract data with missing outcome
  data_missing_outcome <- data %>% filter(is.na(life_satisfaction))
  dim(data_missing_outcome)
```

```
[1] 0 7
```

```
# Omit individuals with missing outcome from data
  data <- data %>% drop_na(life_satisfaction) # ?drop_na
  dim(data)
```

```
[1] 1760    7
```

```
# Split the data into training and test data
  set.seed(100)
  data_split <- initial_split(data, prop = 0.80)
  data_split # Inspect
```

```
<Training/Testing/Total>
<1408/352/1760>
```

```
# Extract the two datasets
  data_train <- training(data_split)
  data_test <- testing(data_split) # Do not touch until the end!

# Create resampled partitions
  set.seed(345)
```

```
data_folds <- vfold_cv(data_train, v = 5) # V-fold/k-fold cross-validation
data_folds # data_folds now contains several resamples of our training data
```

```
#  5-fold cross-validation
# A tibble: 5 x 2
  splits             id
  <list>             <chr>
1 <split [1126/282]> Fold1
2 <split [1126/282]> Fold2
3 <split [1126/282]> Fold3
4 <split [1127/281]> Fold4
5 <split [1127/281]> Fold5
```

Our test data data_test contains 352 observations. The training data (from which we generate the folds) contains 1408 observations.

Step 3: Specify recipe, model and workflow

Below we define different pre-processing steps in our recipe (see `# comments` in the chunk):

```
# Define recipe
recipe1 <-
  recipe(formula = life_satisfaction ~ ., data = data_train) %>%
  step_nzv(all_predictors()) %>% # remove variables with zero variances
  step_novel(all_nominal_predictors()) %>% # prepares data to handle previously unseen facto
  #step_dummy(all_nominal_predictors()) %>% # dummy codes categorical variables
  step_zv(all_predictors()) #%>% # remove vars with only single value
  #step_normalize(all_predictors()) # normale predictors

# Inspect the recipe
  recipe1
```

```
-- Recipe ---------------------------------------------------------------------



-- Inputs
```

```
Number of variables by role

outcome:   1
predictor: 6



-- Operations

* Sparse, unbalanced variable filter on: all_predictors()

* Novel factor level assignment for: all_nominal_predictors()

* Zero variance filter on: all_predictors()
```

```
# Extract and preview data + recipe (direclty with $)
data_preprocessed <- prep(recipe1, data_train)$template
dim(data_preprocessed)
```

```
[1] 1408    6
```

Then we specify our random forest model choosing a `mode`, `engine` and specifying the workflow with `workflow()`:

```
# show engines/package that include random forest
show_engines("rand_forest")
```

```
# A tibble: 6 x 2
  engine       mode
  <chr>        <chr>
1 ranger       classification
2 ranger       regression
3 randomForest classification
4 randomForest regression
5 spark        classification
6 spark        regression
```

```r
# Specify model
model1 <-
  rand_forest(trees = 1000) %>% # grow 1000 trees!
  set_engine("ranger",
             importance = "permutation") %>%
  set_mode("regression")

# Specify workflow
workflow1 <-
  workflow() %>%
  add_recipe(recipe1) %>%
  add_model(model1)
```

Step 4: Fit/train & evalute model using resampling

Then we fit the random forest to our resamples of the training data (different splits into analysis and assessment data) to be better able to evaluate it accounting for possible variation across subsets:

```r
# Fit the random forest to the cross-validation datasets
fit1 <- workflow1 %>%
  fit_resamples(resamples = data_folds, # specify cv-datasets
  metrics = metric_set(rmse, rsq, mae)) # save models
```

And we can evaluate the metrics:

```r
# RMSE and RSQ
collect_metrics(fit1)
```

```
# A tibble: 3 x 6
  .metric .estimator  mean     n std_err .config
  <chr>   <chr>      <dbl> <int>   <dbl> <chr>
1 mae     standard   1.66      5  0.0475 Preprocessor1_Model1
2 rmse    standard   2.15      5  0.0412 Preprocessor1_Model1
3 rsq     standard   0.104     5  0.0146 Preprocessor1_Model1
```

Q: What do the different variables and measures stand for?

- `.metric` = the measures; `.estimator` = type of estimator; `mean` = mean of measure across folds; `n` = number of folds; `std_err` = standard error across folds; `.config` = ?

- **MAE (Mean Absolute Error)**: This is a measure of the average magnitude of errors between predicted and actual values. It is calculated as the sum of absolute differences between predictions and actuals divided by the total number of data points. MAE is easy to interpret because it represents the average distance between predictions and actuals, but it can be less sensitive to large errors than other measures like RMSE.
- **RMSE (Root Mean Squared Error)**: This is another measure of the difference between predicted and actual values that takes into account both the size and direction of the errors. It is calculated as the square root of the mean of squared differences between predictions and actuals. RMSE penalizes larger errors more heavily than smaller ones, making it a useful metric when outliers or extreme values may have a significant impact on model performance. However, its units are not always easily interpreted since they depend on the scale of the dependent variable.
- **Rsquared (R^2)**: Also known as coefficient of determination, this metric compares the goodness-of-fit of a regression line by measuring the proportion of variance in the dependent variable that can be explained by the independent variables. An R^2 score ranges from 0 to 1, with 1 indicating perfect correlation between the predicted and actual values. However, keep in mind that high R^2 does not necessarily imply causality or generalizability outside the sample used to train the model.

Step 5: Fit final model to training data

Once we we are happy with our random forest model (after having used resampling to assess it and potential alternatives) we can fit our workflow that includes the model to the complete training data `data_train` and also assess it's accuracy for the training data again.

```
# Fit final model
fit_final <- fit(workflow1, data = data_train)

# Check accuracy in for full training data
  metrics_combined <-
    metric_set(rsq, rmse, mae) # Set accuracy metrics


  augment(fit_final, new_data = data_train) %>%
      metrics_combined(truth = life_satisfaction, estimate = .pred)
```

```
# A tibble: 3 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 rsq     standard       0.726
2 rmse    standard       1.35
3 mae     standard       1.05
```

Now, we can also explore the variable importance of different predictors and visualize it in Figure 3. The `vip()` does only like objects of class ranger, hence we have to directly access ist in the layer object using `fit_final$fit$fit$fit`

```
# Visualize variable importance
fit_final$fit$fit$fit %>%
    vip::vip(geom = "point") +
  ylab("Importance of different predictors")+
  xlab("Variables")
```

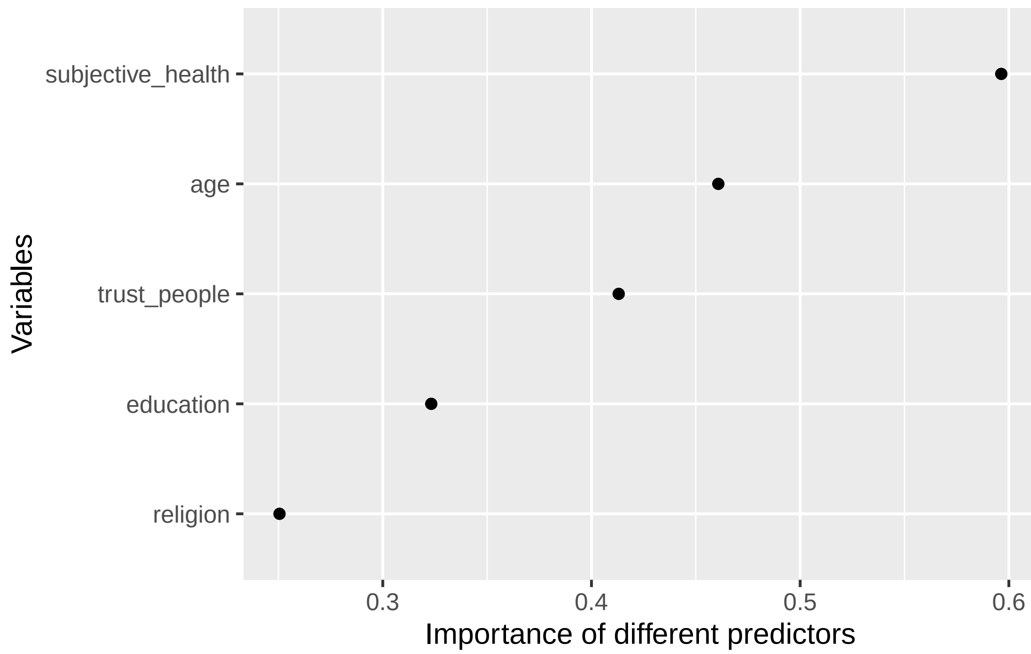


Figure 3: Variable importance for different predictors

Q: What do we see here?

- Figure 3 indicates that trust_people is among the most important predictors.

Step 6: Fit final model to test data and assess accuracy

We then use the model `fit_final` fitted/trained on the training data and evaluate the accuracy of this model which is based on the training data using the test data `data_test`. We use `augment()` to obtain the predictions:

```r
# Test data: Predictions & accuracy
# Test data: Predictions + accuracy
  regression_metrics <- metric_set(mae, rmse, rsq) # Use several metrics

  augment(fit_final , new_data = data_test) %>%
    regression_metrics(truth = life_satisfaction, estimate = .pred)
```

```
# A tibble: 3 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 mae     standard      1.65
2 rmse    standard      2.21
3 rsq     standard      0.0405
```

Example: Predicting internet use with a XGBoost

- Using the ESS, you are interested in building a predictive model of `internet_use_time`, i.e., the minutes an individual spends on the internet per day. Please use the code above to built a RF model to predict this outcome. Importantly, you are free to use the exact same predictors or add new ones. How well can you predict the outcome?

```r
load(file = here("data/data_ess.Rdata"))

# Take subset of variables to speed things up!
  data <- data  %>%
    select(internet_use_time,
           unemployed,
           trust_people,
           education,
           age,
           religion,
           subjective_health) %>%
  mutate(religion = if_else(is.na(religion), "unknown", religion),
         religion = as.factor(religion)) %>%
    drop_na()


# Extract data with missing outcome
  data_missing_outcome <- data %>% filter(is.na(internet_use_time))
  dim(data_missing_outcome)
```

```
[1] 0 7
```

```r
# Omit individuals with missing outcome from data
  data <- data %>% drop_na(internet_use_time) # ?drop_na
  dim(data)
```

```
[1] 1591    7
```

```r
# Split the data into training and test data
  set.seed(100)
  data_split <- initial_split(data, prop = 0.80)
  data_split # Inspect
```

```
<Training/Testing/Total>
<1272/319/1591>
```

```r
# Extract the two datasets
  data_train <- training(data_split)
  data_test <- testing(data_split) # Do not touch until the end!

# Create resampled partitions
  set.seed(345)
  data_folds <- vfold_cv(data_train, v = 5) # V-fold/k-fold cross-validation
  data_folds # data_folds now contains several resamples of our training data
```

```
#  5-fold cross-validation
# A tibble: 5 x 2
  splits             id
  <list>             <chr>
1 <split [1017/255]> Fold1
2 <split [1017/255]> Fold2
3 <split [1018/254]> Fold3
4 <split [1018/254]> Fold4
5 <split [1018/254]> Fold5
```

```r
# Define recipe
  recipe1 <-
    recipe(formula = internet_use_time ~ ., data = data_train) %>%
    step_nzv(all_predictors()) %>% # remove variables with zero variances
    step_novel(all_nominal_predictors()) %>% # prepares data to handle previously unseen fac
```

```r
    step_dummy(all_nominal_predictors()) %>% # dummy codes categorical variables
    step_zv(all_predictors()) %>% # remove vars with only single value
    step_normalize(all_predictors()) # normale predictors

# Inspect the recipe
  recipe1


#  Specify model
set.seed(100)
model1 <-
  boost_tree(  mode = "regression",
  trees = 200,
  tree_depth = 5,
  learn_rate = 0.05,
  engine = "xgboost") %>%
  set_mode("regression")

# Specify workflow
workflow1 <-
  workflow() %>%
  add_recipe(recipe1) %>%
  add_model(model1)


# Fit the random forest to the cross-validation datasets
fit1 <- fit_resamples(
  object = workflow1, # specify workflow
  resamples = data_folds, # specify cv-datasets
  metrics = metric_set(rmse, rsq, mae), # specify metrics to return
  control = control_resamples(verbose = TRUE, # show live comments
                              save_pred = TRUE, # save predictions
                              extract = function(x) extract_fit_engine(x))) # save models


# RMSE and RSQ
collect_metrics(fit1)
```

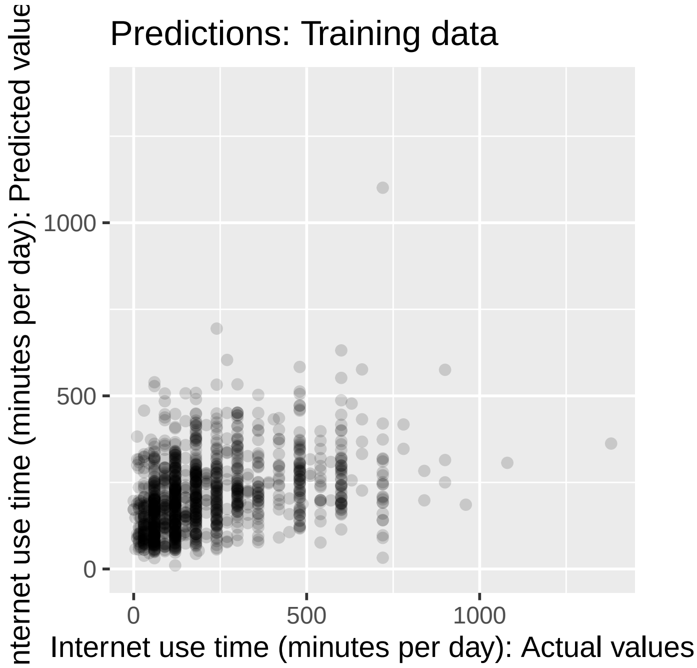
```
# A tibble: 3 x 6
  .metric .estimator    mean     n std_err .config
  <chr>   <chr>        <dbl> <int>   <dbl> <chr>
1 mae     standard   128.        5   1.75  Preprocessor1_Model1
```

```
2 rmse    standard   173.         5  3.94   Preprocessor1_Model1
3 rsq     standard     0.135      5  0.0137 Preprocessor1_Model1
```

```
# Collect average predictions
assessment_data_predictions <- collect_predictions(fit1, summarize = TRUE)
assessment_data_predictions
```

```
# A tibble: 1,272 x 4
   .pred  .row internet_use_time .config
   <dbl> <int>             <dbl> <chr>
 1 174.      1               480 Preprocessor1_Model1
 2 259.      2               420 Preprocessor1_Model1
 3 233.      3               120 Preprocessor1_Model1
 4 251.      4                90 Preprocessor1_Model1
 5 319.      5                60 Preprocessor1_Model1
 6 275.      6                90 Preprocessor1_Model1
 7 175.      7               180 Preprocessor1_Model1
 8  70.8     8                90 Preprocessor1_Model1
 9 207.      9               480 Preprocessor1_Model1
10 226.     10               300 Preprocessor1_Model1
# i 1,262 more rows
```

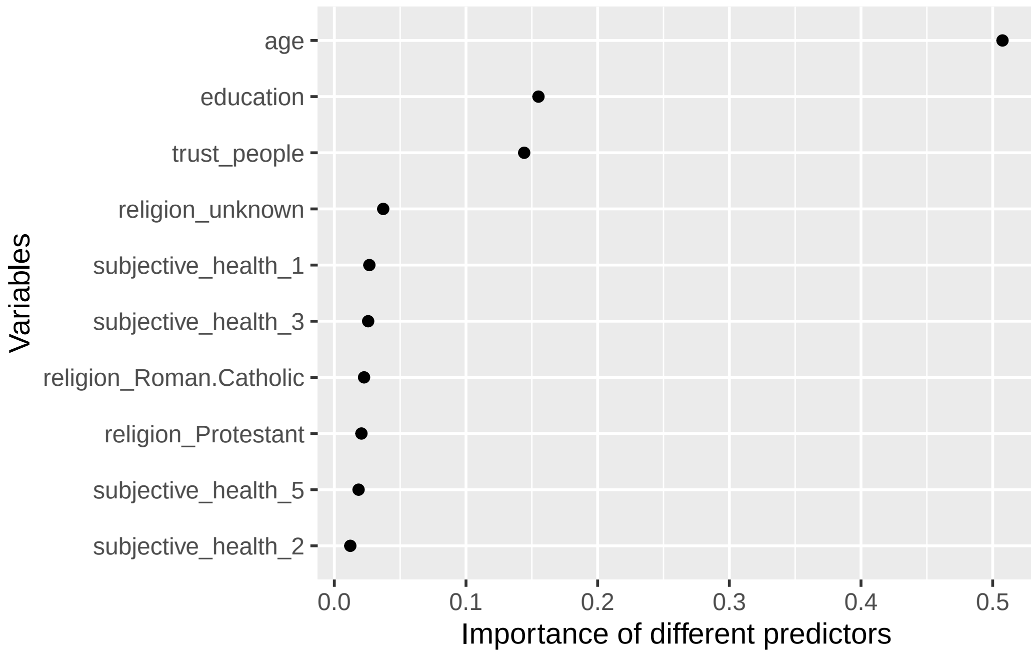
```
# Visualize actual vs. predicted values
assessment_data_predictions %>%
  ggplot(aes(x = internet_use_time, y = .pred)) +
  geom_point(alpha = .15) +
  #geom_abline(color = "red") +
  coord_obs_pred() +
  ylab("Internet use time (minutes per day): Predicted values") +
  xlab("Internet use time (minutes per day): Actual values") +
  ggtitle("Predictions: Training data")
```




```
# Fit final model
fit_final <- fit(workflow1, data = data_train)

# Check accuracy in for complete training data
  augment(fit_final, new_data = data_train) %>%
    mae(truth = internet_use_time, estimate = .pred)
```

```
# A tibble: 1 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 mae     standard        91.3
```

```
# Visualize variable importance
fit_final$fit$fit$fit %>%
    vip::vip(geom = "point") +
  ylab("Importance of different predictors")+
  xlab("Variables")
```




```
# Test data: Predictions + accuracy
  regression_metrics <- metric_set(mae, rmse, rsq) # Use several metrics

  augment(fit_final , new_data = data_test) %>%
    regression_metrics(truth = internet_use_time, estimate = .pred)
```

```
# A tibble: 3 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 mae     standard     123.
2 rmse    standard     171.
3 rsq     standard       0.171
```

Example: Classification with Random Forests

Below you can find and example of building a random forest model for our binary outcome recidivism (without resampling).

```
load(file = here("data/data_compas.Rdata"))
```

```r
# Extract data with missing outcome
  data_missing_outcome <- data %>% filter(is.na(is_recid_factor))
  dim(data_missing_outcome)
```

```
[1] 614  14
```

```r
# Omit individuals with missing outcome from data
  data <- data %>% drop_na(is_recid_factor) # ?drop_na
  dim(data)
```

```
[1] 6600   14
```

```r
# Split the data into training and test data
  set.seed(100)
  data_split <- initial_split(data, prop = 0.80)
  data_split # Inspect
```

```
<Training/Testing/Total>
<5280/1320/6600>
```

```r
# Extract the two datasets
  data_train <- training(data_split)
  data_test <- testing(data_split) # Do not touch until the end!

# Create resampled partitions
  set.seed(345)
  data_folds <- vfold_cv(data_train, v = 5) # V-fold/k-fold cross-validation
  data_folds # data_folds now contains several resamples of our training data
```

```
#  5-fold cross-validation
# A tibble: 5 x 2
  splits              id
  <list>              <chr>
1 <split [4224/1056]> Fold1
2 <split [4224/1056]> Fold2
3 <split [4224/1056]> Fold3
4 <split [4224/1056]> Fold4
5 <split [4224/1056]> Fold5
```

```r
## Step 3: Specify recipe, model and workflow


# Define recipe
recipe1 <-
  recipe(formula = is_recid_factor ~ age + priors_count + sex + juv_fel_count + juv_misd_cour
    step_filter_missing(all_predictors(), threshold = 0.01) %>%
    step_naomit(is_recid_factor, all_predictors()) %>% # better deal with missings beforehanc
    step_nzv(all_predictors(), freq_cut = 0, unique_cut = 0) %>% # remove variables with zerc
    step_dummy(all_nominal_predictors())

  # Inspect the recipe
    recipe1
```

```
-- Recipe ----------------------------------------------------------------------


-- Inputs

Number of variables by role

outcome:   1
predictor: 6


-- Operations

* Missing value column filter on: all_predictors()

* Removing rows with NA values in: is_recid_factor and all_predictors()

* Sparse, unbalanced variable filter on: all_predictors()

* Dummy variables from: all_nominal_predictors()
```

```r
  # Check preprocessed data
    data_preprocessed <- prep(recipe1, data_train)$template
    dim(data_preprocessed)
```

```
[1] 5280    7
```

```r
# show engines/package that include random forest
show_engines("rand_forest")
```

```
# A tibble: 6 x 2
  engine       mode
  <chr>        <chr>
1 ranger       classification
2 ranger       regression
3 randomForest classification
4 randomForest regression
5 spark        classification
6 spark        regression
```

```r
# Specify model
model1 <-
  rand_forest(trees = 1000) %>% # grow 1000 trees!
  set_engine("ranger",
             importance = "permutation") %>%
  set_mode("classification")

# Specify workflow
workflow1 <-
  workflow() %>%
  add_recipe(recipe1) %>%
  add_model(model1)


## Step 4: Fit/train & evalute model using resampling

# Fit the random forest to the cross-validation datasets
fit1 <- workflow1 %>%
  fit_resamples(resamples = data_folds, # specify cv-datasets
  metrics = metric_set(accuracy, precision, recall, f_meas)) # save models
```

```r
# RMSE and RSQ
collect_metrics(fit1)
```

```
# A tibble: 4 x 6
  .metric   .estimator  mean     n std_err .config
  <chr>     <chr>      <dbl> <int>   <dbl> <chr>
1 accuracy  binary     0.676     5 0.00238 Preprocessor1_Model1
2 f_meas    binary     0.697     5 0.00362 Preprocessor1_Model1
3 precision binary     0.673     5 0.00506 Preprocessor1_Model1
4 recall    binary     0.725     5 0.0111  Preprocessor1_Model1
```

```r
## Step 5: Fit final model to training data


# Fit final model
fit_final <- fit(workflow1, data = data_train)

# Check accuracy in for full training data
  metrics_combined <-
    metric_set(accuracy, precision, recall, f_meas) # Set accuracy metrics

  data_train %>%
    augment(x = fit_final, type.predict = "response") %>%
        metrics_combined(truth = is_recid_factor, estimate = .pred_class)
```

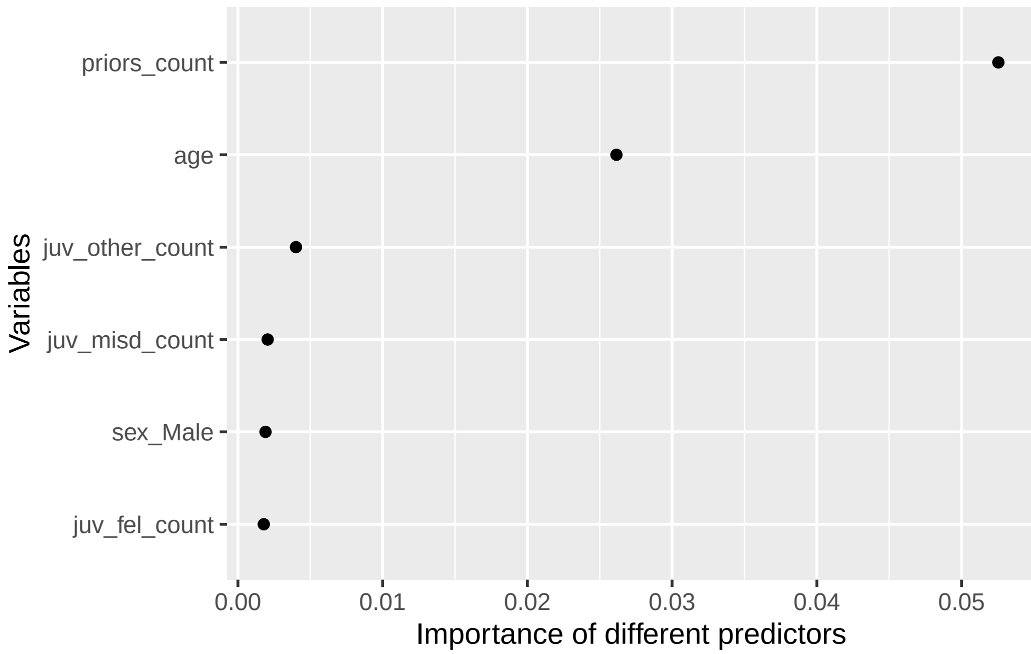
```
# A tibble: 4 x 3
  .metric   .estimator .estimate
  <chr>     <chr>          <dbl>
1 accuracy  binary         0.708
2 precision binary         0.701
3 recall    binary         0.754
4 f_meas    binary         0.727
```

```r
# Confusion matrix
  data_train %>%
    augment(x = fit_final, type.predict = "response") %>%
        conf_mat(truth = is_recid_factor, estimate = .pred_class)
```

```
          Truth
Prediction   no  yes
```

```
      no  2049  872
      yes  670 1689
```

```r
# Visualize variable importance
fit_final$fit$fit$fit %>%
    vip::vip(geom = "point") +
  ylab("Importance of different predictors")+
  xlab("Variables")
```




```r
# Test data: Predictions + accuracy
  data_test %>%
    augment(x = fit_final, type.predict = "response") %>%
        metrics_combined(truth = is_recid_factor, estimate = .pred_class)
```

```
# A tibble: 4 x 3
  .metric   .estimator .estimate
  <chr>     <chr>          <dbl>
1 accuracy  binary         0.686
2 precision binary         0.701
3 recall    binary         0.714
4 f_meas    binary         0.708
```

5. Tuning models

Tuning a random forest

Step 1: Loading, renaming and recoding

```
load(file = here("data/data_ess.Rdata"))
```

```
# Take subset of variables to speed things up!
  data <- data  %>%
    select(life_satisfaction,
           unemployed,
           trust_people,
           education,
           age,
           religion,
           subjective_health) %>%
  mutate(religion = if_else(is.na(religion), "unknown", religion),
         religion = as.factor(religion)) %>%
    drop_na()
```

Step 2: Prepare & split the data

```
# Extract data with missing outcome
  data_missing_outcome <- data %>% filter(is.na(life_satisfaction))
  dim(data_missing_outcome)
```

```
[1] 0 7
```

```
# Omit individuals with missing outcome from data
  data <- data %>% drop_na(life_satisfaction) # ?drop_na
  dim(data)
```

```
[1] 1760    7
```

```
# STEP 2
  # Split the data into training and test data
    data_split <- initial_split(data, prop = 0.80)
    data_split # Inspect
```

```
<Training/Testing/Total>
<1408/352/1760>
```

```
  # Extract the two datasets
    data_train <- training(data_split)
    data_test <- testing(data_split) # Do not touch until the end!

  # Create resampled partitions
    set.seed(345)
    data_folds <- vfold_cv(data_train, v = 2) # V-fold/k-fold cross-validation
    data_folds # data_folds now contains several resamples of our training data
```

```
#  2-fold cross-validation
# A tibble: 2 x 2
  splits            id
  <list>            <chr>
1 <split [704/704]> Fold1
2 <split [704/704]> Fold2
```

Step 3: Specify recipe, model, workflow and tuning parameters

Similar to ridge or lasso regression a random forest has parameters that we can try to tune. Below, we create a model specification for a RF where we will tune…

- **Number of Predictors to Sample at Each Split** (`mtry`): This hyperparameter controls the number of predictors randomly sampled at each split in the decision tree building process. A **smaller value** of `mtry` can lead to **less correlation between individual trees** in the forest, **potentially reducing overfitting**, but it may also increase the randomness in the model. Conversely, a **larger value** of `mtry` can lead to **stronger individual trees** but might increase the **risk of overfitting**. Typically, you can try different values of `mtry` and choose the one that provides the best performance based on cross-validation or other evaluation methods.
- **Minimum Number of Observations in Each Node** (`min_n`): This hyperparameter determines the minimum number of observations required in a node for it to be eligible for further splitting. If a node has fewer than `min_n` observations, it won't be split further, effectively **controlling the depth and complexity of the trees**. Setting a **higher value** of `min_n` can **help prevent overfitting by creating simpler trees**, but it **may also lead to underfitting**if set too high.

These are hyperparameters that can't be learned from data when training the model. (cf. Source)

```r
# Define recipe
model_recipe <-
  recipe(formula = life_satisfaction ~ ., data = data_train) %>%
  step_naomit(life_satisfaction, all_predictors()) %>% # better deal with missings beforeha
  step_nzv(all_predictors(), freq_cut = 0, unique_cut = 0) %>% # remove variables with zero
  step_novel(all_nominal_predictors()) %>% # prepares data to handle previously unseen fact
  #step_unknown(all_nominal_predictors()) %>% # categorizes missing categorical data (NA's)
  step_dummy(all_nominal_predictors(), -has_role("id vars")) %>% # dummy codes categorical
  step_zv(all_predictors()) %>% # remove vars with only single value
  step_normalize(all_predictors()) # normale predictors

# Inspect the recipe
  model_recipe
```

```
-- Recipe ------------------------------------------------------------------


-- Inputs

Number of variables by role

outcome:   1
predictor: 6


-- Operations

* Removing rows with NA values in: life_satisfaction and all_predictors()

* Sparse, unbalanced variable filter on: all_predictors()

* Novel factor level assignment for: all_nominal_predictors()

* Dummy variables from: all_nominal_predictors() and -has_role("id vars")
```

```
* Zero variance filter on: all_predictors()

* Centering and scaling for: all_predictors()
```

```
# Specify model with tuning
model1 <- rand_forest(
  mtry = tune(), # tune mtry parameter (number of predictors to sample at each split)
  trees = 1000, # grow 1000 trees
  min_n = tune() # tune min_n parameter (min N in Node to split)
) %>%
  set_mode("regression") %>%
  set_engine("ranger",
             importance = "permutation") # potentially computational intensive


# Specify workflow (with tuning)
workflow1 <- workflow() %>%
  add_recipe(model_recipe) %>%
  add_model(model1)
```

Step 4: Tune, evaluate the model using resampling and choose/explore the best model

Tune, evaluate the model using resampling

Below we use `tune_grid()` to compute performance metrics (e.g. accuracy or RMSE) for pre-defined set of tuning parameters that correspond to a model or recipe across one or more resamples of the data (below 10 stored in `data_folds`).

```
# Specify to use parallel processing
doParallel::registerDoParallel()

set.seed(345)
tune_result <- tune_grid(
  workflow1,
  resamples = data_folds,
  grid = 5 # choose 10 grid points automatically
)
```

```
i Creating pre-processing data to finalize unknown parameter: mtry
```

```r
tune_result
```

```
# Tuning results
# 2-fold cross-validation
# A tibble: 2 x 4
  splits            id    .metrics          .notes
  <list>            <chr> <list>            <list>
1 <split [704/704]> Fold1 <tibble [10 x 6]> <tibble [0 x 3]>
2 <split [704/704]> Fold2 <tibble [10 x 6]> <tibble [0 x 3]>
```

Visualizing the results helps us to evaluate the tuning results. Figure 4 indicates that higher values of `min_n` and lower values of `mtry` seem to work better in terms of accuracy.

```r
tune_result %>%
  collect_metrics() %>% # extract metrics
  filter(.metric == "rmse") %>% # keep rmse only
  select(mean, min_n, mtry) %>% # subset variables
  pivot_longer(min_n:mtry, # convert to longer
    values_to = "value",
    names_to = "parameter"
  ) %>%
  ggplot(aes(value, mean, color = parameter)) + # plot!
  geom_point(show.legend = FALSE) +
  facet_wrap(~parameter, scales = "free_x") +
  labs(x = NULL, y = "RMSE")
```

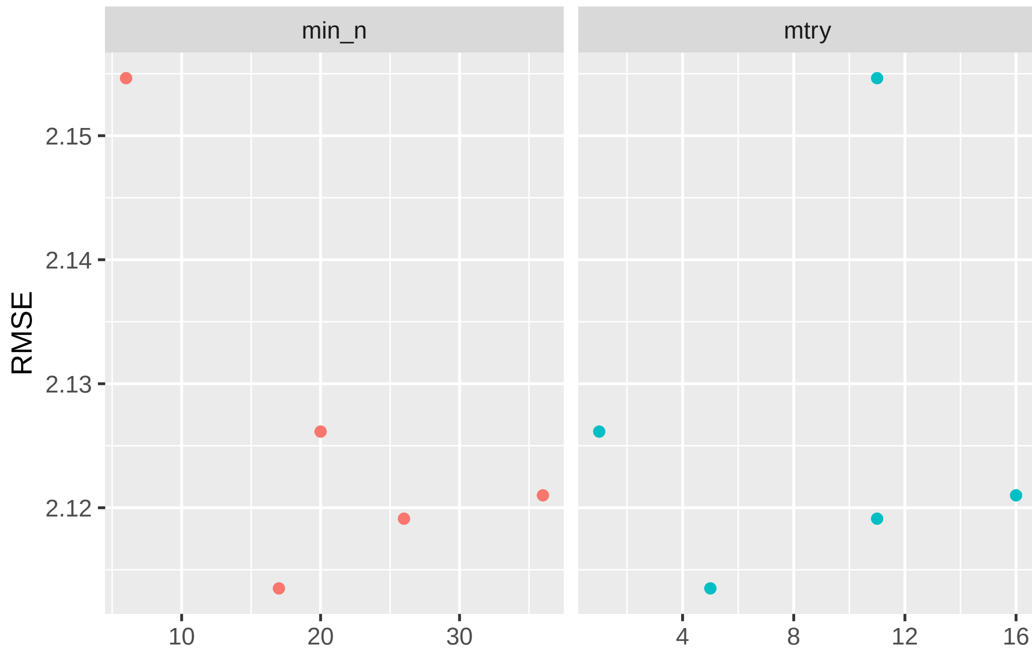


Figure 4: Tuning: RMSE across different parameter values of min_n and mtry

After getting this first glimpse in Figure 4 we might want to make further changes to the grid values that we use for tuning. Below we pick ranges that turned out to be better in Figure 4:

```
grid1 <- grid_regular(
  mtry(range = c(0, 10)), # define range for mtry
  min_n(range = c(20, 40)), # define range for min_n
  levels = 4 # number of values of each parameter to use to make the regular grid
)

grid1
```

```
# A tibble: 16 x 2
    mtry min_n
   <int> <int>
 1     0    20
 2     3    20
 3     6    20
 4    10    20
 5     0    26
 6     3    26
 7     6    26
```

```
 8    10    26
 9     0    33
10     3    33
11     6    33
12    10    33
13     0    40
14     3    40
15     6    40
16    10    40
```

Then we re-do the tuning using those values:

```r
set.seed(456)
tune_result2 <- tune_grid(
  workflow1,
  resamples = data_folds,
  grid = grid1
)

tune_result2
```

```
# Tuning results
# 2-fold cross-validation
# A tibble: 2 x 4
  splits            id    .metrics          .notes
  <list>            <chr> <list>            <list>
1 <split [704/704]> Fold1 <tibble [32 x 6]> <tibble [0 x 3]>
2 <split [704/704]> Fold2 <tibble [32 x 6]> <tibble [0 x 3]>
```

Again we visualize the results in Figure 5:

```r
tune_result2 %>%
  collect_metrics() %>%
  filter(.metric == "rmse") %>%
  select(mean, min_n, mtry) %>%
  pivot_longer(min_n:mtry,
    values_to = "value",
    names_to = "parameter"
  ) %>%
  ggplot(aes(value, mean, color = parameter)) +
  geom_point(show.legend = FALSE) +
```

```r
  facet_wrap(~parameter, scales = "free_x") +
  labs(x = NULL, y = "RMSE")
```

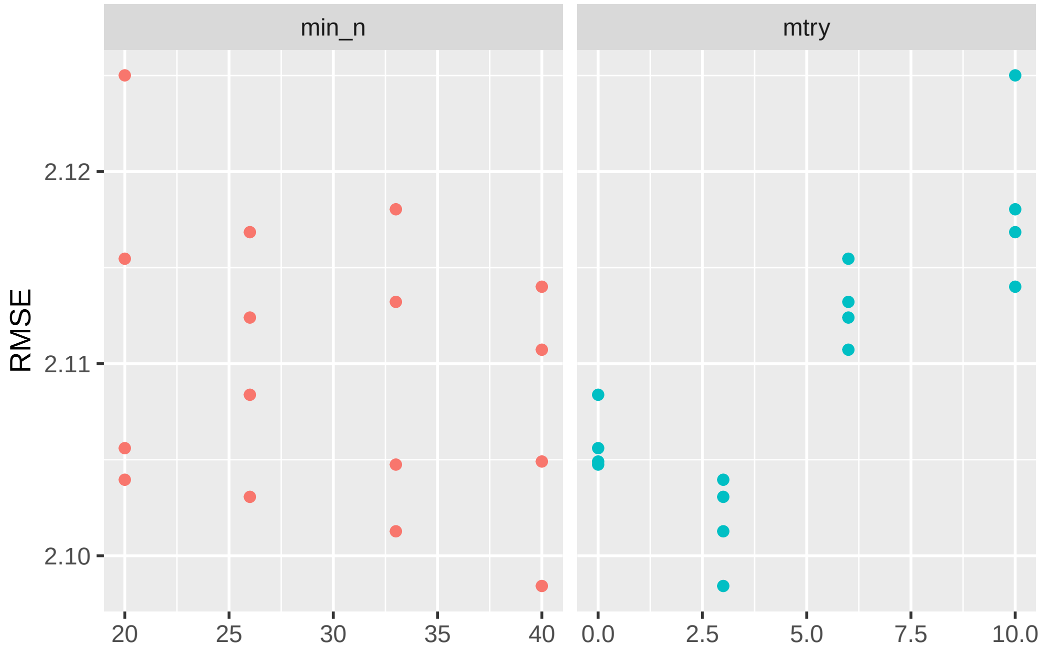


Figure 5: Tuning: RMSE across different parameter values of min_n and mtry

Choose best model after tuning

Choosing the best model, i.e., the model with the best parameter choices obtained in the tuning above (`tune_result2`), can be done with the `select_best()` function. After having selected the best parameter values, we update our original model specification stored in `model1` using the `finalize_model()` function.

```r
# Find tuning parameter combination with best performance values
best_hyperparameters <- select_best(tune_result2, metric = "rmse")

# Take list/tibble of tuning parameter values
# and update model1 with those values.
model_final <- finalize_model(model1,
                              parameters = best_hyperparameters # define
                              )
```

Step 5: Fit final model to training data and assess accuracy

Once we are happy with our tuned random forest model and have chosen the best model specification stored in `model_final` we can fit our workflow to the training data `data_train` again and also assess it's accuracy again.

```
# Define final workflow
workflow_final <- workflow() %>%
  add_recipe(model_recipe) %>% #  use standard recipe
  add_model(model_final) # use final model

# Fit final model
fit_final <- fit(workflow_final, data = data_train)
fit_final
```

```
== Workflow [trained] ==========================================================
Preprocessor: Recipe
Model: rand_forest()

-- Preprocessor ----------------------------------------------------------------
6 Recipe Steps

* step_naomit()
* step_nzv()
* step_novel()
* step_dummy()
* step_zv()
* step_normalize()

-- Model -----------------------------------------------------------------------
Ranger result

Call:
 ranger::ranger(x = maybe_data_frame(x), y = y, mtry = min_cols(~3L,      x), num.trees = ~1(

Type:                             Regression
Number of trees:                  1000
Sample size:                      1408
Number of independent variables:  17
Mtry:                             3
Target node size:                 40
Variable importance mode:         permutation
```

```
Splitrule:                            variance
OOB prediction error (MSE):           4.406323
R squared (OOB):                      0.1187923
```

```r
# Q: What do the values for `mtry` and `min_n` in the final model mean?

# Check accuracy in training data using MAE
  augment(fit_final, new_data = data_train) %>%
    mae(truth = life_satisfaction, estimate = .pred)
```

```
# A tibble: 1 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 mae     standard        1.50
```

Now, that we have our final model we can also explore it assessing variable importance (i.e., which variables where the most relevant to find splits) using the `vip` package. We can use `vi()` and `vip()` to extract or extract+plot the variable importance of different predictors as shown in Table 3 and Figure 6. The `vi()` and `vip()` function does only like objects of class ranger, hence we have to directly access is in the layer object using `fit_final$fit$fit$fit`

```r
# Visualize variable importance
fit_final$fit$fit %>%
    vip::vi() %>%
  kable()
```

Table 3: Variable importance for different predictors

Variable	Importance
subjective_health_2	0.3257620
subjective_health_1	0.2732320
subjective_health_5	0.2357552
subjective_health_4	0.1971936
trust_people	0.1681840
education	0.1628918
subjective_health_3	0.1520783
religion_Roman.Catholic	0.0839632
age	0.0703828
religion_unknown	0.0458498
unemployed	0.0313067

Table 3: Variable importance for different predictors

Variable	Importance
religion_Islam	0.0188054
religion_Other.Non.Christian.religions	0.0031265
religion_Other.Christian.denomination	-0.0009881
religion_Jewish	-0.0010353
religion_Eastern.religions	-0.0015410
religion_Protestant	-0.0069502

```
fit_final$fit$fit %>%
  vip(geom = "point")
```

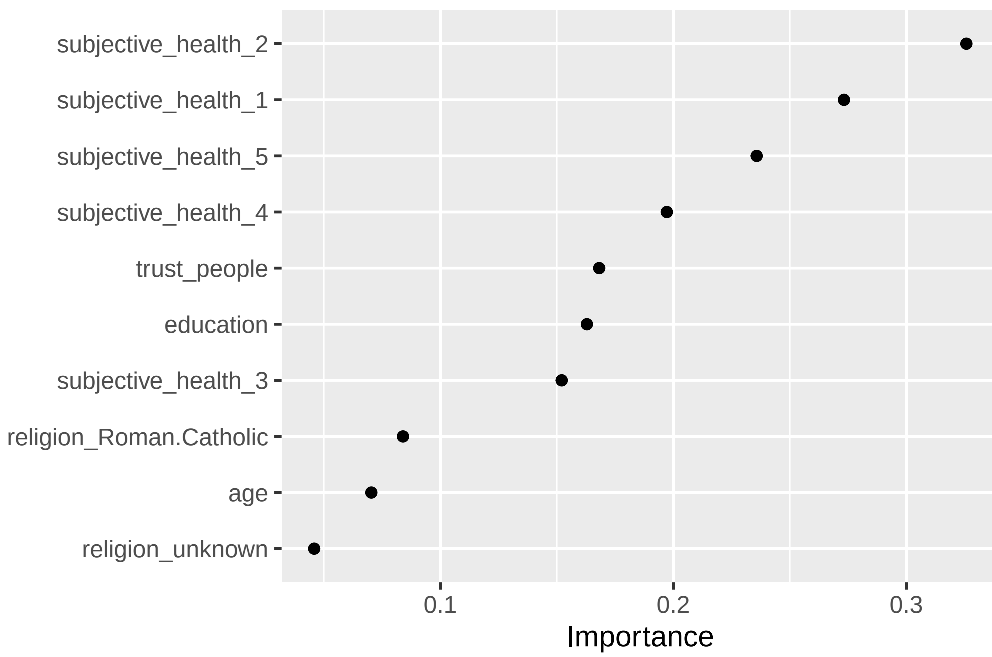


Figure 6: Variable importance for different predictors

Step 6: Fit final model to test data and assess accuracy

We then evaluate the accuracy of this model which is based on the training data using the test data `data_test`. We use `augment()` to obtain the predictions:

```r
# Use fitted model to predict values
  augment(fit_final, new_data = data_test)
```

```
# A tibble: 352 x 8
   .pred life_satisfaction unemployed trust_people education   age religion
   <dbl>              <dbl>      <dbl>        <dbl>     <dbl> <dbl> <fct>
 1  7.36                  7          0            5         3    25 unknown
 2  8.05                  9          0            5         4    36 Roman Cathol~
 3  7.01                  9          0            7         2    38 Roman Cathol~
 4  5.71                  7          0            1         0    46 unknown
 5  6.84                  8          0            3         3    31 unknown
 6  7.30                  8          0            0         3    59 Roman Cathol~
 7  6.82                  0          0            2         6    40 Islam
 8  7.90                  8          0            5         6    32 unknown
 9  7.30                  8          0            7         2    56 Roman Cathol~
10  7.64                 10          0            5         3    25 unknown
# i 342 more rows
# i 1 more variable: subjective_health <ord>
```

And can directly pipe the result into functions such as `rsq()`, `rmse()` and `mae()` to obtain the corresponding measures of accuracy in the test data `data_test`.

```r
augment(fit_final, new_data = data_test) %>%
  rsq(truth = life_satisfaction, estimate = .pred)
```

```
# A tibble: 1 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 rsq     standard      0.0829
```

```r
augment(fit_final, new_data = data_test) %>%
  rmse(truth = life_satisfaction, estimate = .pred)
```

```
# A tibble: 1 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 rmse    standard        2.08
```

```r
augment(fit_final, new_data = data_test) %>%
    mae(truth = life_satisfaction, estimate = .pred)
```

```
# A tibble: 1 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 mae     standard        1.61
```

The corresponding accuracy measures can now be compared to those of an untuned random forest model.

Tuning ridge regression

We first import the data into R:

```r
load(file = here("data/data_ess.Rdata"))
```

```r
# Drop missings on outcome variable
  data <- data %>%
  drop_na(life_satisfaction) %>%
  select(where(~mean(is.na(.)) < 0.1))  %>%
    na.omit()

# Split the data into training and test data
  data_split <- initial_split(data, prop = 0.80)
  data_split # Inspect
```

```
<Training/Testing/Total>
<696/174/870>
```

```r
# Extract the two datasets
  data_train <- training(data_split)
  data_test <- testing(data_split) # Do not touch until the end!

# Create resampled partitions
  set.seed(345)
  data_folds <- vfold_cv(data_train, v = 2) # V-fold/k-fold cross-validation
  data_folds # data_folds now contains several resamples of our training data
```

```
#  2-fold cross-validation
# A tibble: 2 x 2
  splits            id
  <list>            <chr>
1 <split [348/348]> Fold1
2 <split [348/348]> Fold2
```

```
  # You can also try loo_cv(data_train) instead

# Define recipe
recipe1 <-
  recipe(formula = life_satisfaction ~ ., data = data_train) %>%
  update_role(respondent_id, new_role = "ID") %>%
  # step_naomit(all_predictors()) %>% # we did this above
  step_dummy(all_nominal_predictors()) %>%
  step_zv(all_predictors()) %>% # remove any numeric variables that have zero variance.
  step_normalize(all_numeric(), -all_outcomes()) # normalize (center and scale) the numeric v
```

Then we specify the model. It's similar to previous labs only that we set `penalty = tune()` to tell `tune_grid()` that the penalty parameter should be tuned.

```
# Define model
model1 <-
  linear_reg(penalty = tune(), mixture = 0) %>%
  set_mode("regression") %>%
  set_engine("glmnet")
```

Then we define a workflow called `workflow1` that includes our recipe and model specification.

```
# Define workflow
workflow1 <- workflow() %>%
  add_recipe(recipe1) %>%
  add_model(model1)
```

And we use `grid_regular()` to create a grid of evenly spaced parameter values. In it we use the `penalty()` function (`dials` package) to denote the parameter and set the range of the grid. Computation is fast so we can choose a fine-grained grid with 50 levels.

```
# Set up grid for search
penalty_grid <- grid_regular(penalty(range = c(-5, 5)), levels = 10)
penalty_grid
```

```
# A tibble: 10 x 1
         penalty
           <dbl>
 1      0.00001
 2      0.000129
 3      0.00167
 4      0.0215
 5      0.278
 6      3.59
 7     46.4
 8    599.
 9   7743.
10 100000
```

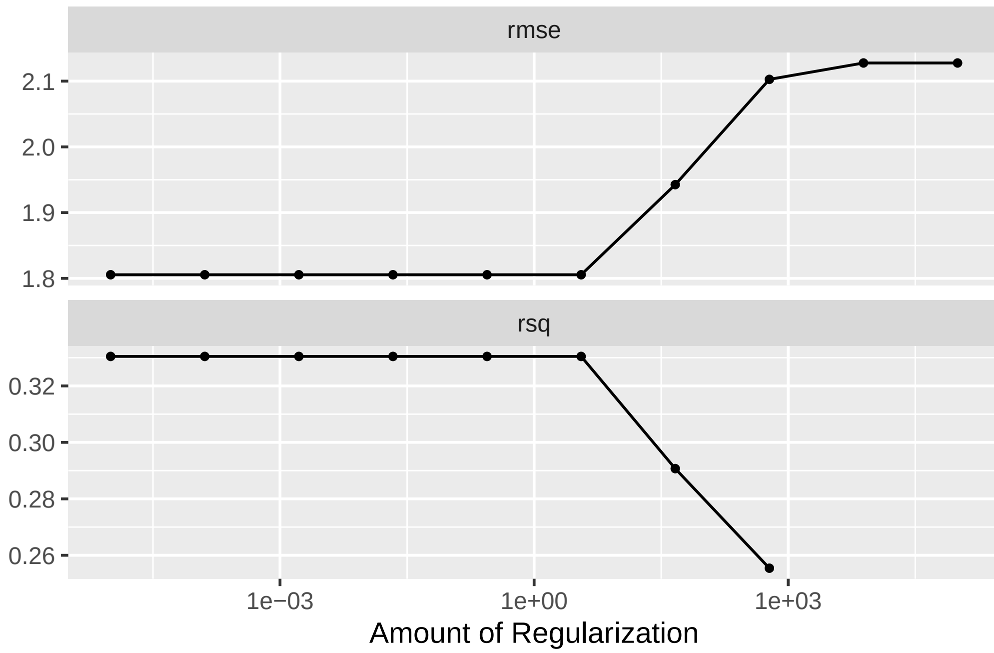
Then we can fit all our models using the resamples in data_folds using the tune_grid function.

```
# Tune the model
tune_result <- tune_grid(
  workflow1,
  resamples = data_folds,
  grid = penalty_grid
)

tune_result # display result
```

```
# Tuning results
# 2-fold cross-validation
# A tibble: 2 x 4
  splits            id    .metrics          .notes
  <list>            <chr> <list>            <list>
1 <split [348/348]> Fold1 <tibble [20 x 5]> <tibble [1 x 3]>
2 <split [348/348]> Fold2 <tibble [20 x 5]> <tibble [1 x 3]>

There were issues with some computations:

  - Warning(s) x2: A correlation computation is required, but `estimate` is constant...

Run `show_notes(.Last.tune.result)` for more information.
```

And use autoplot to visualize the results.

```r
# Visualize tuning results
autoplot(tune_result)
```



It visualizes how the performance metrics are impacted by our parameter choice of the regularization parameter, i.e, penalty λ.

We can also collect the metrics:

```r
# Collect metrics
collect_metrics(tune_result)
```

```
# A tibble: 20 x 7
         penalty .metric .estimator  mean     n std_err .config
           <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
 1      0.00001  rmse    standard   1.81      2  0.0361 Preprocessor1_Model01
 2      0.00001  rsq     standard   0.330     2  0.0530 Preprocessor1_Model01
 3      0.000129 rmse    standard   1.81      2  0.0361 Preprocessor1_Model02
 4      0.000129 rsq     standard   0.330     2  0.0530 Preprocessor1_Model02
 5      0.00167  rmse    standard   1.81      2  0.0361 Preprocessor1_Model03
 6      0.00167  rsq     standard   0.330     2  0.0530 Preprocessor1_Model03
 7      0.0215   rmse    standard   1.81      2  0.0361 Preprocessor1_Model04
 8      0.0215   rsq     standard   0.330     2  0.0530 Preprocessor1_Model04
 9      0.278    rmse    standard   1.81      2  0.0361 Preprocessor1_Model05
```

```
10       0.278    rsq     standard     0.330     2  0.0530 Preprocessor1_Model05
11       3.59     rmse    standard     1.81      2  0.0361 Preprocessor1_Model06
12       3.59     rsq     standard     0.330     2  0.0530 Preprocessor1_Model06
13      46.4      rmse    standard     1.94      2  0.0631 Preprocessor1_Model07
14      46.4      rsq     standard     0.291     2  0.0603 Preprocessor1_Model07
15     599.       rmse    standard     2.10      2  0.0733 Preprocessor1_Model08
16     599.       rsq     standard     0.255     2  0.0648 Preprocessor1_Model08
17    7743.       rmse    standard     2.13      2  0.0725 Preprocessor1_Model09
18    7743.       rsq     standard   NaN         0 NA     Preprocessor1_Model09
19 100000         rmse    standard     2.13      2  0.0725 Preprocessor1_Model10
20 100000         rsq     standard   NaN         0 NA     Preprocessor1_Model10
```

Afterwards `select_best()` can be used to extract the best parameter value.

```
# Extract best penalty after tuning
best_penalty <- select_best(tune_result, metric = "rsq")
best_penalty
```

```
# A tibble: 1 x 2
  penalty .config
    <dbl> <chr>
1 0.00001 Preprocessor1_Model01
```

Finalize workflow, assess accuracy and extract predicted values

Finally, we can update our workflow with `finalize_workflow()` and set the penalty to `best_penalty` that we stored above, and fit the model on our training data.

```
# Define final workflow
workflow_final <- finalize_workflow(workflow1, best_penalty)
workflow_final
```

```
== Workflow ================================================================
Preprocessor: Recipe
Model: linear_reg()

-- Preprocessor ------------------------------------------------------------
3 Recipe Steps

* step_dummy()
* step_zv()
```

```
* step_normalize()

-- Model -----------------------------------------------------------------------
Linear Regression Model Specification (regression)

Main Arguments:
  penalty = 1e-05
  mixture = 0

Computational engine: glmnet
```

```
# Fit the model
fit_final <- fit(workflow_final, data = data_train)
```

We then evaluate the accuracy of this model (that is based on the training data) using the test data `data_test`. We use `augment()` to obtain the predictions:

```
# Use fitted model to predict values
  augment(fit_final, new_data = data_test)
```

```
# A tibble: 174 x 201
   .pred respondent_id life_satisfaction country unemployed_active unemployed
   <dbl>         <dbl>             <dbl> <fct>               <dbl>      <dbl>
 1  6.36         10125                 7 FR                      0          0
 2  9.90         10349                 9 FR                      0          0
 3  8.77         10358                 9 FR                      0          0
 4  7.53         10436                 8 FR                      0          0
 5  7.17         10625                 7 FR                      0          0
 6  6.19         10663                 6 FR                      0          0
 7  6.88         10701                 8 FR                      0          0
 8  7.56         10726                 6 FR                      0          0
 9  9.41         10742                 6 FR                      0          0
10  7.63         10743                 8 FR                      0          0
# i 164 more rows
# i 195 more variables: education <dbl>, news_politics_minutes <dbl>,
#   internet_use_frequency <ord>, trust_people <dbl>, people_fair <dbl>,
#   people_helpful <dbl>, political_interest <ord>, system_allows_say <ord>,
#   active_role_politics <ord>, system_allows_influence <ord>,
#   confident_participate_politics <ord>, trust_parliament <dbl>,
#   trust_legal_system <dbl>, trust_police <dbl>, trust_politicians <dbl>, ...
```

And can directly pipe the result into functions such as `rsq()`, `rmse()` and `mae()` to obtain the corresponding measures of accuracy.

```
augment(fit_final, new_data = data_test) %>%
  rsq(truth = life_satisfaction, estimate = .pred)
```

```
# A tibble: 1 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 rsq     standard       0.411
```

```
augment(fit_final, new_data = data_test) %>%
  rmse(truth = life_satisfaction, estimate = .pred)
```

```
# A tibble: 1 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 rmse    standard        1.72
```

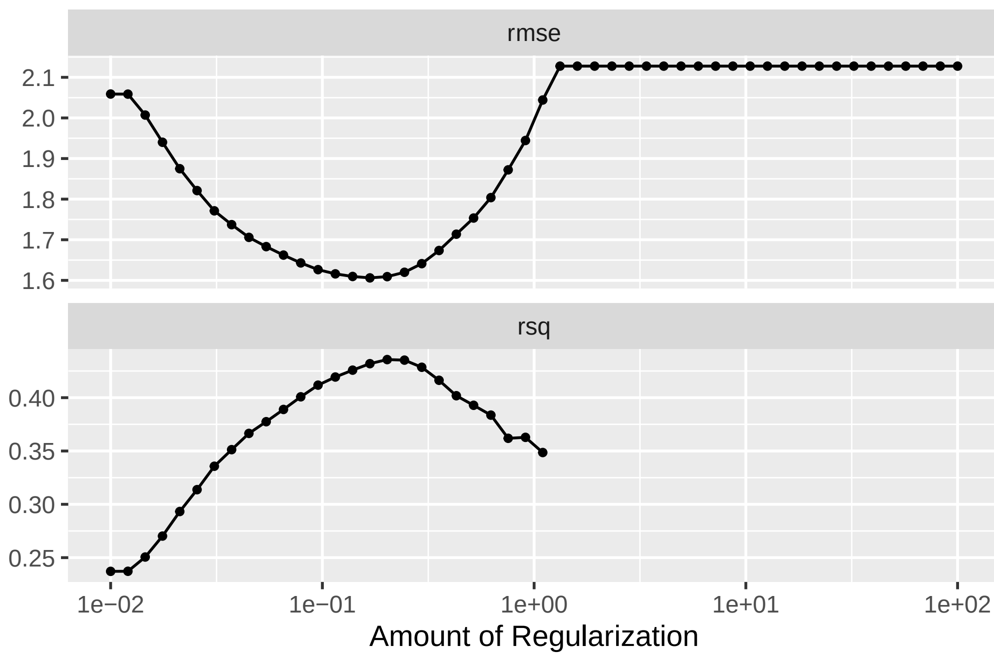
```
augment(fit_final, new_data = data_test) %>%
    mae(truth = life_satisfaction, estimate = .pred)
```

```
# A tibble: 1 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 mae     standard        1.29
```

Tuning lasso

```
# Define the recipe
recipe2 <- recipe(life_satisfaction ~ ., data = data_train) %>%
    step_zv(all_numeric_predictors()) %>% # remove predictors with zero variance
    step_normalize(all_numeric_predictors()) %>% # normalize predictors
    step_dummy(all_nominal_predictors())

# Specify the model
model2 <-
  linear_reg(penalty = tune(), mixture = 1) %>%
```

```r
  set_mode("regression") %>%
  set_engine("glmnet")

# Define the workflow
workflow2 <- workflow() %>%
  add_recipe(recipe2) %>%
  add_model(model2)

# Define the penalty grid
penalty_grid <- grid_regular(penalty(range = c(-2, 2)), levels = 50)

# Tune the model and visualize
  tune_result <- tune_grid(
    workflow2,
    resamples = data_folds,
    grid = penalty_grid
  )

  autoplot(tune_result)
```




```r
# Select best penalty
best_penalty <- select_best(tune_result, metric = "rsq")
```

```r
# Finalize workflow and fit model
workflow_final <- finalize_workflow(workflow2, best_penalty)

fit_final <- fit(workflow_final, data = data_train)


# Check accuracy in training data
  augment(fit_final, new_data = data_train) %>%
    mae(truth = life_satisfaction, estimate = .pred)
```

```
# A tibble: 1 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 mae     standard        1.12
```

```r
# Add predicted values to test data
augment(fit_final, new_data = data_test)
```

```
# A tibble: 174 x 201
   .pred respondent_id life_satisfaction country unemployed_active unemployed
   <dbl>         <dbl>             <dbl> <fct>               <dbl>      <dbl>
 1  6.55         10125                 7 FR                      0          0
 2  8.03         10349                 9 FR                      0          0
 3  7.88         10358                 9 FR                      0          0
 4  8.16         10436                 8 FR                      0          0
 5  6.38         10625                 7 FR                      0          0
 6  8.50         10663                 6 FR                      0          0
 7  7.82         10701                 8 FR                      0          0
 8  8.02         10726                 6 FR                      0          0
 9  7.68         10742                 6 FR                      0          0
10  8.02         10743                 8 FR                      0          0
# i 164 more rows
# i 195 more variables: education <dbl>, news_politics_minutes <dbl>,
#   internet_use_frequency <ord>, trust_people <dbl>, people_fair <dbl>,
#   people_helpful <dbl>, political_interest <ord>, system_allows_say <ord>,
#   active_role_politics <ord>, system_allows_influence <ord>,
#   confident_participate_politics <ord>, trust_parliament <dbl>,
#   trust_legal_system <dbl>, trust_police <dbl>, trust_politicians <dbl>, ...
```

```
# Assess accuracy (RSQ)
augment(fit_final, new_data = data_test) %>%
  rsq(truth = life_satisfaction, estimate = .pred)
```

```
# A tibble: 1 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 rsq     standard       0.526
```

```
# Assess accuracy (MAE)
augment(fit_final, new_data = data_test) %>%
  mae(truth = life_satisfaction, estimate = .pred)
```

```
# A tibble: 1 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 mae     standard        1.07
```

Tuning XGBoost model

Below we use XGBoost to built a predictive model of life satisfaction. See here for another example. And Section for and example of refining parameters after a first automated grid search.

```
load(file = here("data/data_ess.Rdata"))
```

```
    set.seed(345)

# Extract data with missing outcome
  data_missing_outcome <- data %>% filter(is.na(life_satisfaction))
  dim(data_missing_outcome)

# Omit individuals with missing outcome from data
  data <- data %>% drop_na(life_satisfaction) # ?drop_na
  dim(data)

# STEP 2
  # Split the data into training and test data
    data_split <- initial_split(data, prop = 0.80)
```

```
    data_split # Inspect

  # Extract the two datasets
    data_train <- training(data_split)
    data_test <- testing(data_split) # Do not touch until the end!

  # Create resampled partitions
    data_folds <- vfold_cv(data_train, v = 5) # V-fold/k-fold cross-validation
    data_folds # data_folds now contains several resamples of our training data

  # Define recipe
  recipe1 <-
    recipe(formula = life_satisfaction ~ ., data = data_train) %>%
    update_role(respondent_id, new_role = "ID") %>% # Declare ID variable
    step_filter_missing(all_predictors(), threshold = 0.01) %>%
    step_naomit(life_satisfaction, all_predictors()) %>% # better deal with missings beforeha
    step_nzv(all_predictors(), freq_cut = 0, unique_cut = 0) %>% # remove variables with zero
    #step_novel(all_nominal_predictors()) %>% # prepares data to handle previously unseen fac
    step_unknown(all_nominal_predictors()) %>% # categorizes missing categorical data (NA's)
    step_dummy(all_nominal_predictors())# %>% # dummy codes categorical variables
    #step_zv(all_predictors()) %>% # remove vars with only single value
    #step_normalize(all_predictors()) # normale predictors

  # Inspect the recipe
    recipe1

  # Check preprocessed data
    data_preprocessed <- prep(recipe1, data_train)$template
    dim(data_preprocessed)


# Specify model with tuning
  model1 <-
  boost_tree(
    trees = 1000,
    min_n = tune(), # Tune (see ?details_boost_tree_xgboost)
    mtry = tune(),
    stop_iter = tune(),
    learn_rate = 0.01, # Choose higher value for speed (e.g., 0.3)
    loss_reduction = tune(),
    sample_size = tune()
  ) %>%
```

```r
  set_engine("xgboost") %>%
  set_mode("regression")


# Specify workflow (with tuning)
  workflow1 <- workflow() %>%
    add_recipe(recipe1) %>%
    add_model(model1)



## Step 4: Tune, evaluate the model using resampling and choose/explore the best model


# Specify to use parallel processing
  doParallel::registerDoParallel()


  tune_result <- tune_grid(
    workflow1,
    resamples = data_folds,
    metrics = metric_set(mae, rmse, rsq), # Specify which metrics to calculate
    grid = 5 # Choose grid points automatically; Pick lower value for speed
  )

  tune_result


# Show accuracy for different hyperparameters
  tune_result %>%
  collect_metrics()


# Visualize accuracy for different hyperparameters
  tune_result %>%
    collect_metrics() %>% # extract metrics
    filter(.metric == "mae") %>% # keep mae only
    select(mean, mtry:stop_iter) %>% # subset variables
    pivot_longer(mtry:stop_iter, # convert to longer
      values_to = "value",
      names_to = "parameter"
    ) %>%
```

```
    ggplot(aes(value, mean, color = parameter)) + # plot!
    geom_point(show.legend = FALSE) +
    facet_wrap(~parameter, scales = "free_x") +
    labs(x = NULL, y = "MAE")


## Choose best model after tuning (for final fit)


# Find tuning parameter combination with best performance values
  best_hyperparameters <- select_best(tune_result, metric = "mae")
  best_hyperparameters


# Take list/tibble of tuning parameter values
# and update model1 with those values.
  model_final <- finalize_model(model1,
                                parameters = best_hyperparameters # define
                                )

## Step 5: Fit final model to training data and assess accuracy

# Define final workflow
  workflow_final <- workflow() %>%
    add_recipe(recipe1) %>% #  use standard recipe
    add_model(model_final) # use final model

# Fit final model
  fit_final <- fit(workflow_final, data = data_train)
  fit_final
# Q: What do the values for `mtry` and `min_n` in the final model mean?


# Check accuracy in for full training data
  metrics_combined <- metric_set(rsq, rmse, mae) # Set accuracy metrics

  augment(fit_final, new_data = data_train) %>%
      metrics_combined(truth = life_satisfaction, estimate = .pred)



# Visualize variable importance
```

```r
  fit_final$fit$fit %>%
      vip::vi() %>%
    kable()


fit_final$fit$fit %>%
  vip(geom = "point")



## Step 6: Fit final model to test data and assess accuracy
  augment(fit_final, new_data = data_test) %>%
      metrics_combined(truth = life_satisfaction, estimate = .pred)
```