Resampling Methods

AU STAT-427/627

Emil Hvitfeldt

2021-2-16

Motivation

We are already familiar with train-test splits

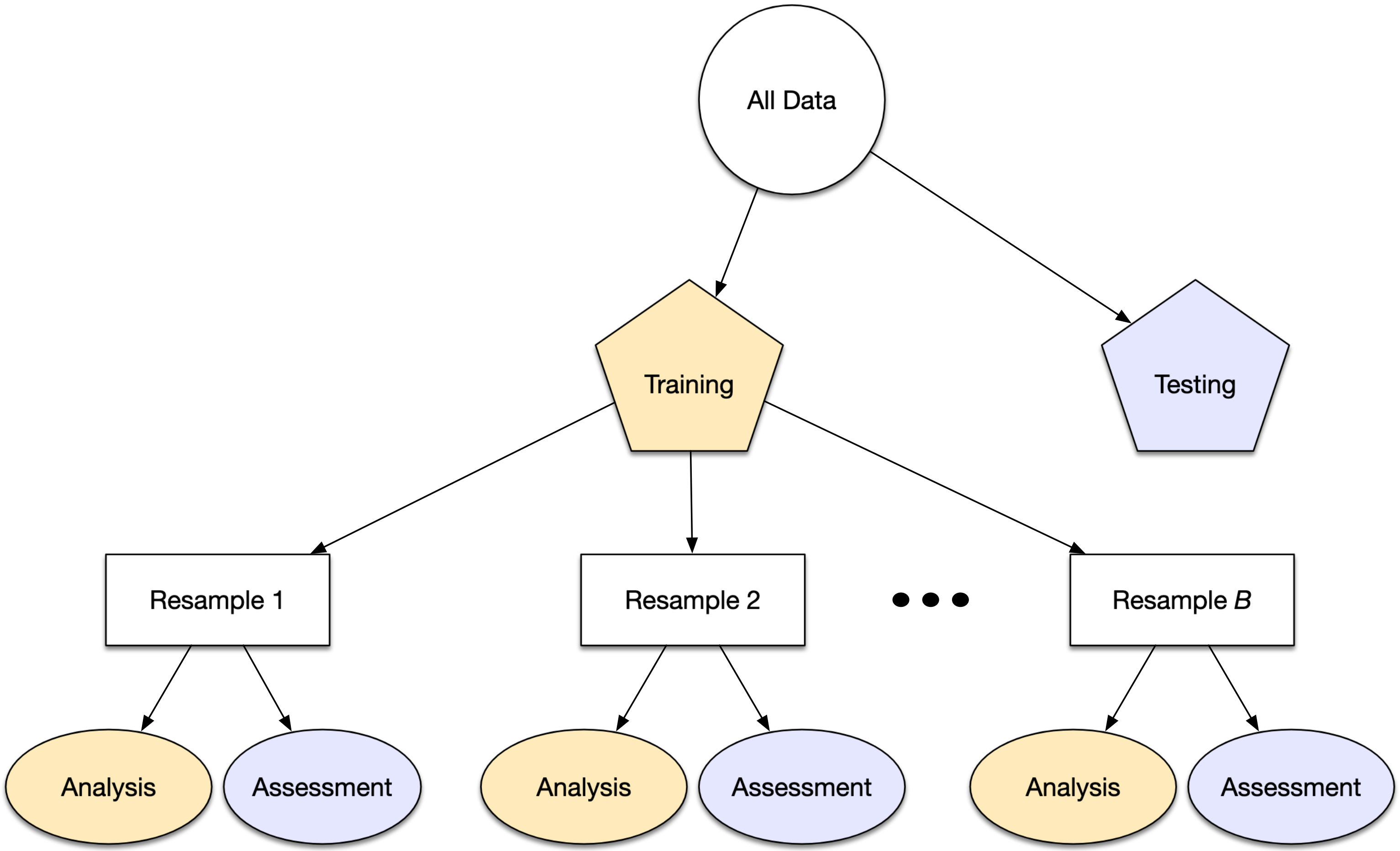
The main downside to to train-test splits so far is that we can only use them once

This means we effectively can't make any decisions about the models we are using

Resampling

Resampling estimates of performance can generalize to new data

Resampling Workflow



Resampling Workflow

The resampling is only conducted on the training set

We are still keeping the test set.The test set is not involved.

For each iteration of resampling, the data are partitioned into two subsamples:

- The model is fit with the analysis set
- The model is evaluated with the assessment set

Resampling Workflow

We have effectively created many train-test split out of our training data set.

The challange here now becomes how we are creating these resample sets

Resampling Workflow

Suppose we generate 10 different resamples

This mean that we will be:

- Fitting 10 different models
- Perform predictions 10 times
- Prodice 10 sets of performance statistics

The final estimate of the performance of the model will be the average of these 10 models

Resampling Workflow

If the resampling is done in a appropiate way then this average has very good generalization properties

Leave-One-Out Cross Validation

- 1 observation is used as the assessment set
- The remaining observations make up the analysis set

Notes:

We are fitting the model on $n - 1$ observations and a prediction $\hat{y}_1$ is made on the assessment set using the value x_1

Leave-One-Out Cross Validation

Since (x_1, y_1) is not used in the fitting process, then $MSE_1 = (y_1 - \hat{y}_1)^2$ provides an approximately unbiased estimate for the test error.

While this estimate is approximately unbiased, it is quite poor since it is highly variable

Leave-One-Out Cross Validation

We can repeat this for

- $MSE_2 = (y_2 - \hat{y}_2)^2$
- $MSE_3 = (y_3 - \hat{y}_3)^2$
- ...
-

$MSE_n = (y_n - \hat{y}_n)^2$

to get n estimates of the test error

Leave-One-Out Cross Validation

The LOOCV estimate of the test MSE is

$$CV_{(n)} = \frac{1}{n} \sum_{i=1}^{n} MSE_i$$

Leave-One-Out Cross Validation

Pros

The LOOCV estimate of the test MSE is going to have low bias

There is no randomness in the LOOCV estimate

Cons

You need a lot of computational power even for modest data sets

(Some models don't need to be repeatedly refit)

K-Fold Cross Validation

Could we think of a compromise between fitting 1 model and n models?

K-Fold Cross Validation has an answer:

Randomly divide the observations into k groups (or folds) or approximately equal size

K-Fold Cross Validation

Randomly divide the observations into k groups (or folds) or approximately equal size

- 1 fold is used as the assessment set
- The remaining folds make up the analysis set

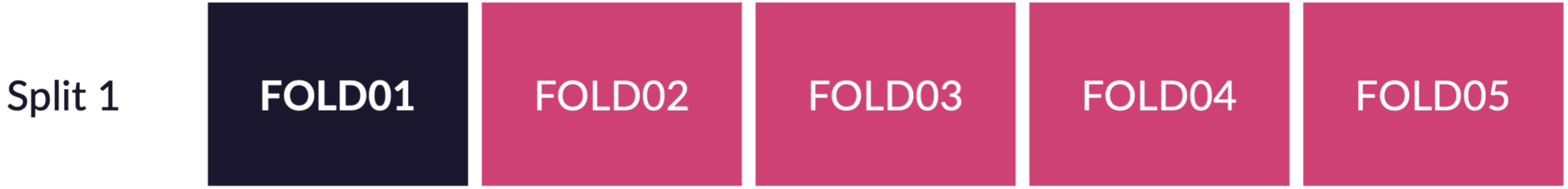
Everything else happens as before.

We now get fewer performance metrics, BUT they are each less variable

Art by Alison Hill

Art by Alison Hill

Art by Alison Hill



Art by Alison Hill

Art by Alison Hill

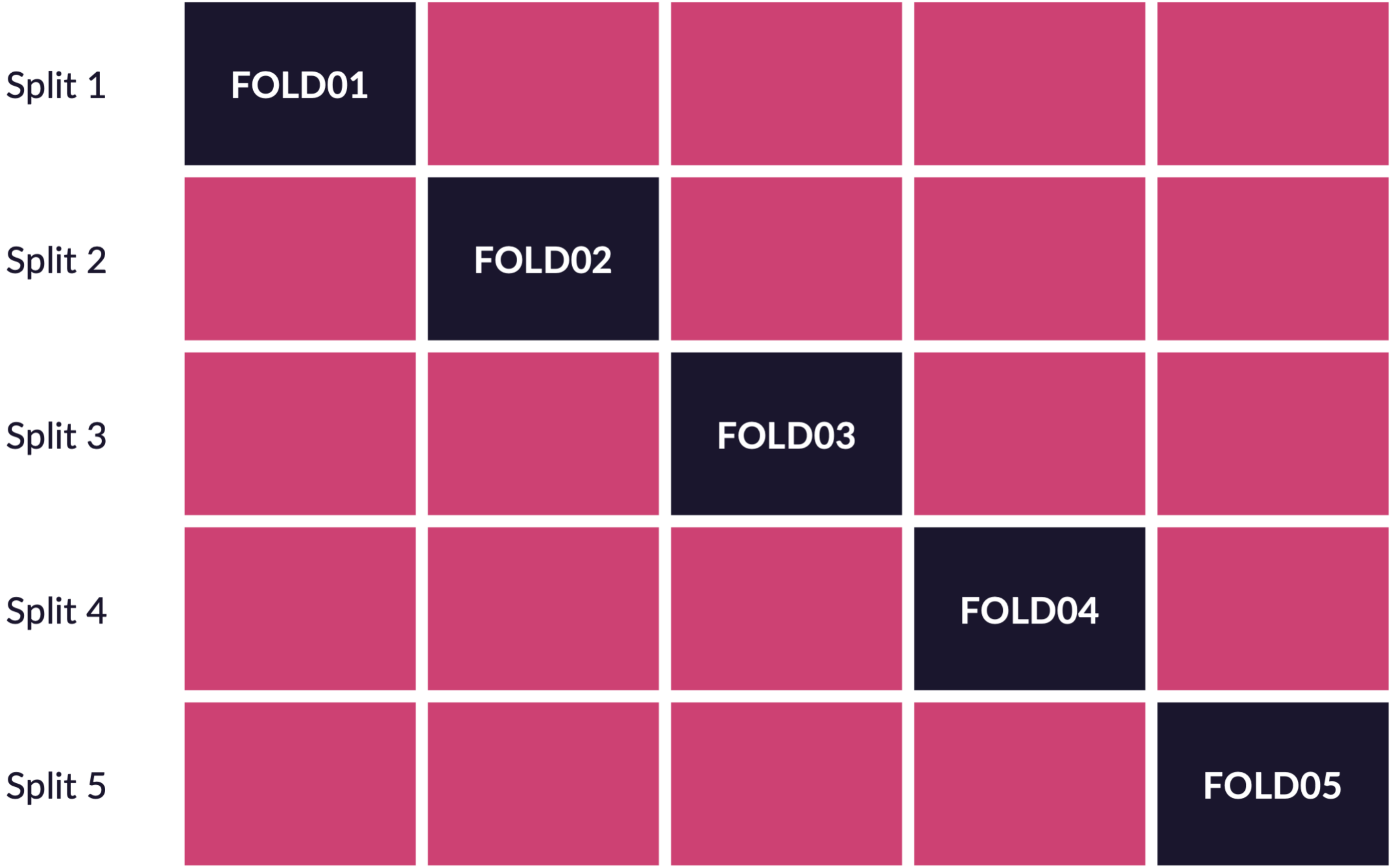
Art by Alison Hill

Split 1	**FOLD01**	FOLD02	FOLD03	FOLD04	FOLD05
Split 2	FOLD01	**FOLD02**	FOLD03	FOLD04	FOLD05
Split 3	FOLD01	FOLD02	**FOLD03**	FOLD04	FOLD05
Split 4	FOLD01	FOLD02	FOLD03	**FOLD04**	FOLD05

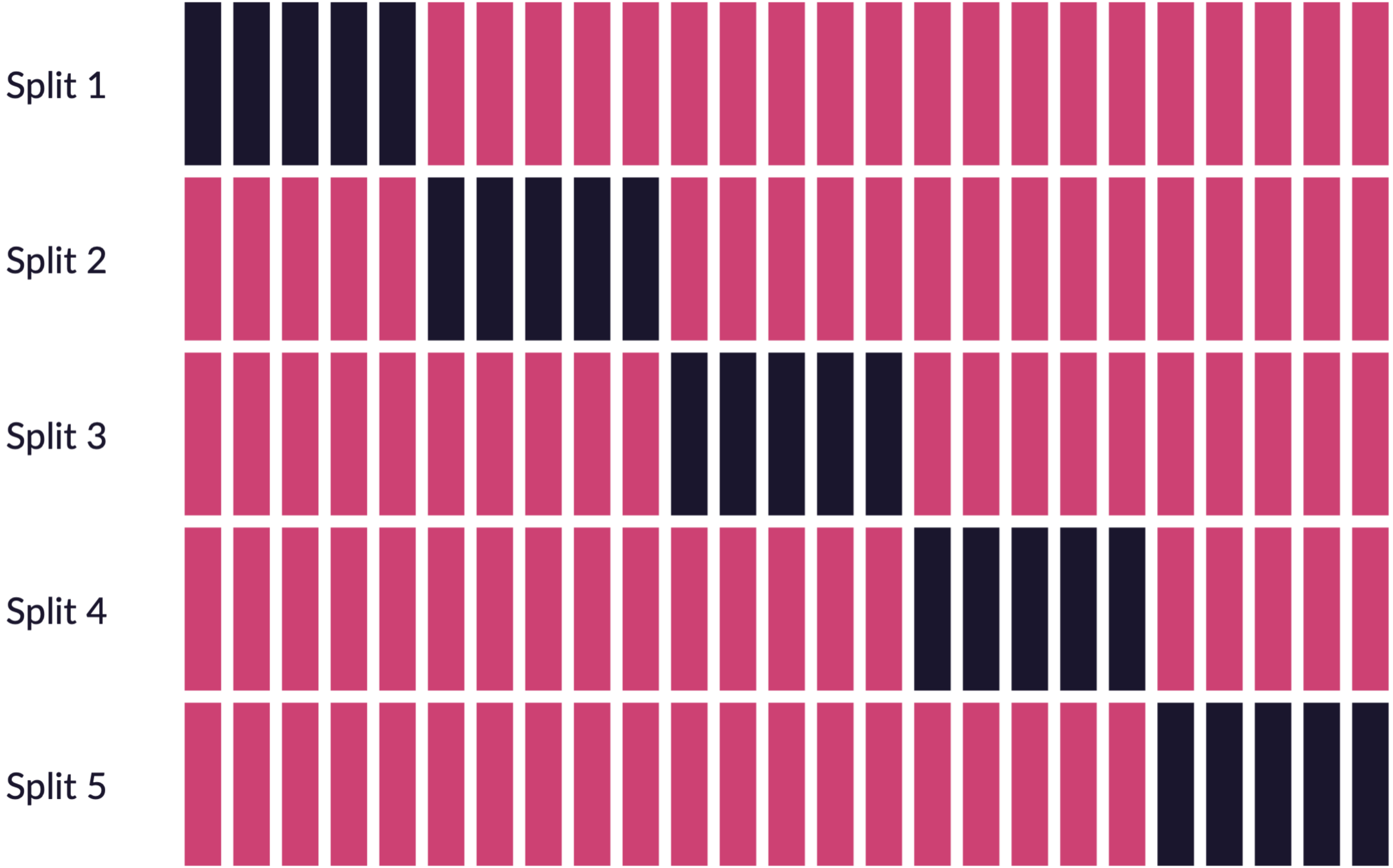
Art by Alison Hill

Split 1	**FOLD01**	FOLD02	FOLD03	FOLD04	FOLD05
Split 2	FOLD01	**FOLD02**	FOLD03	FOLD04	FOLD05
Split 3	FOLD01	FOLD02	**FOLD03**	FOLD04	FOLD05
Split 4	FOLD01	FOLD02	FOLD03	**FOLD04**	FOLD05
Split 5	FOLD01	FOLD02	FOLD03	FOLD04	**FOLD05**

Art by Alison Hill



Art by Alison Hill



Art by Alison Hill

Cross validation

When we perform cross-validation our goal might be to determine how well a given model is expected to perform on new data

Other times we are using cross-validation to find the best model/hyperparameters

Bias-Variance radeoff of LOOCV and k-fold Cross Validation

LOOCV has a lower bias then k-fold CV

However since the mean of many highly correlated quantities has higher variance then the mean of many not correlated quantities, we have that LOOCV has a higher variance then k-fold CV

Rsample

We are back with `rsample`

```
library(rsample)

mtcars
```

```
##                      mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4           21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag       21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710          22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive      21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout   18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Valiant             18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Duster 360          14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Merc 240D           24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 230            22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Merc 280            19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Merc 280C           17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## Merc 450SE          16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
## Merc 450SL          17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## Merc 450SLC         15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
```

Rsample

We can use the `vfold_cv()` function on a data.frame to create a `vfold_cv` object

```
mtcars_folds <- vfold_cv(mtcars, v = 4)
mtcars_folds
```

```
## #  4-fold cross-validation
## # A tibble: 4 x 2
##   splits          id
##   <list>          <chr>
## 1 <split [24/8]> Fold1
## 2 <split [24/8]> Fold2
## 3 <split [24/8]> Fold3
## 4 <split [24/8]> Fold4
```

Rsample

An under the hood, we have 4 analysis/assesment splits similar to `initial_split()`

```
mtcars_folds <- vfold_cv(mtcars, v = 4)
mtcars_folds$splits
```

```
## [[1]]
## <Analysis/Assess/Total>
## <24/8/32>
##
## [[2]]
## <Analysis/Assess/Total>
## <24/8/32>
##
## [[3]]
## <Analysis/Assess/Total>
## <24/8/32>
##
## [[4]]
## <Analysis/Assess/Total>
## <24/8/32>
```

Using resamples in action

We start by creating a linear regression sepecification

```
library(parsnip)
linear_spec <- linear_reg() %>%
  set_mode("regression") %>%
  set_engine("lm")
```

Workflows

Simple package that helps us formulate more about what happens to our model.

Main functions are `workflow()`, `add_model()`, `add_formula()` or `add_variables()` (we will see `add_recipe()` later in the course)

```
library(workflows)

linear_wf <- workflow() %>%
  add_model(linear_spec) %>%
  add_formula(mpg ~ disp + hp + wt)
```

Workflows

This allows up to combine the model with what variables it should expect

```r
library(workflows)

linear_wf <- workflow() %>%
  add_model(linear_spec) %>%
  add_formula(mpg ~ disp + hp + wt)
linear_wf
```

```
## ══ Workflow ═════════════════════════════════════
## Preprocessor: Formula
## Model: linear_reg()
##
## ── Preprocessor ─────────────────────────────────
## mpg ~ disp + hp + wt
##
## ── Model ────────────────────────────────────────
## Linear Regression Model Specification (regressio
##
## Computational engine: lm
```

`add_variables()` allows for a different way of specifying the the response and predictors in our model

Workflows

```r
library(workflows)

linear_wf <- workflow() %>%
  add_model(linear_spec) %>%
  add_variables(outcomes = mpg,
                predictors = c(disp, hp, wt))
linear_wf
```

```
## ══ Workflow ════════════════════════════════════════════
## Preprocessor: Variables
## Model: linear_reg()
##
## ── Preprocessor ────────────────────────────────────────
## Outcomes: mpg
## Predictors: c(disp, hp, wt)
##
## ── Model ───────────────────────────────────────────────
## Linear Regression Model Specification (regressio
##
## Computational engine: lm
```

Workflows

You can use a `workflow` just like a parsnip object and fit it directly

```
fit(linear_wf, data = mtcars)
```

```
## ══ Workflow [trained] ══════════════════════════════════════════════════════════
## Preprocessor: Variables
## Model: linear_reg()
##
## ── Preprocessor ────────────────────────────────────────────────────────────────
## Outcomes: mpg
## Predictors: c(disp, hp, wt)
##
## ── Model ───────────────────────────────────────────────────────────────────────
##
## Call:
## stats::lm(formula = ..y ~ ., data = data)
##
## Coefficients:
## (Intercept)         disp           hp           wt
##   37.105505    -0.000937    -0.031157    -3.800891
```

Tune

We introduce the **tune** package. This package helps us fit many models in a controlled manner in the tidymodels framework. It relies heavily on parsnip and rsample

Tune

We can use `fit_resamples()` to fit the workflow we created within each resample

```
library(tune)

linear_fold_fits <- fit_resamples(
    linear_wf,
    resamples = mtcars_folds
)
```

Tune

The results of this resampling comes as a data.frame

```
linear_fold_fits
```

```
## # Resampling results
## # 4-fold cross-validation
## # A tibble: 4 x 4
##   splits          id    .metrics          .notes
##   <list>          <chr> <list>            <list>
## 1 <split [24/8]> Fold1 <tibble [2 × 4]> <tibble [0 × 1]>
## 2 <split [24/8]> Fold2 <tibble [2 × 4]> <tibble [0 × 1]>
## 3 <split [24/8]> Fold3 <tibble [2 × 4]> <tibble [0 × 1]>
## 4 <split [24/8]> Fold4 <tibble [2 × 4]> <tibble [0 × 1]>
```

Tune

`collect_metrics()` can be used to extract the CV estimate

```
library(tune)

linear_fold_fits <- fit_resamples(
    linear_wf,
    resamples = mtcars_folds
)

collect_metrics(linear_fold_fits)
```

```
## # A tibble: 2 x 6
##   .metric .estimator  mean     n std_err .config
##   <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1 rmse    standard   2.83      4  0.242  Preprocessor1_Model1
## 2 rsq     standard   0.852     4  0.0400 Preprocessor1_Model1
```

Tune

Setting `summarize = FALSE` in `collect_metrics()` Allows us the see the individual performance metrics for each fold

```
collect_metrics(linear_fold_fits, summarize = FALSE)
```

```
## # A tibble: 8 x 5
##   id    .metric .estimator .estimate .config
##   <chr> <chr>   <chr>          <dbl> <chr>
## 1 Fold1 rmse    standard       2.84  Preprocessor1_Model1
## 2 Fold1 rsq     standard       0.890 Preprocessor1_Model1
## 3 Fold2 rmse    standard       2.98  Preprocessor1_Model1
## 4 Fold2 rsq     standard       0.743 Preprocessor1_Model1
## 5 Fold3 rmse    standard       2.17  Preprocessor1_Model1
## 6 Fold3 rsq     standard       0.928 Preprocessor1_Model1
## 7 Fold4 rmse    standard       3.33  Preprocessor1_Model1
## 8 Fold4 rsq     standard       0.846 Preprocessor1_Model1
```

Tune

There are some settings we can set with `control_resamples()`.

One of the most handy ones (IMO) is `verbose = TRUE`

```r
library(tune)

linear_fold_fits <- fit_resamples(
  linear_wf,
  resamples = mtcars_folds,
  control = control_resamples(verbose = TRUE)
)
```

```
i Fold1: preprocessor 1/1
✓ Fold1: preprocessor 1/1
i Fold1: preprocessor 1/1, model 1/1
✓ Fold1: preprocessor 1/1, model 1/1
i Fold1: preprocessor 1/1, model 1/1 (predictions)
i Fold2: preprocessor 1/1
✓ Fold2: preprocessor 1/1
i Fold2: preprocessor 1/1, model 1/1
✓ Fold2: preprocessor 1/1, model 1/1
i Fold2: preprocessor 1/1, model 1/1 (predictions)
i Fold3: preprocessor 1/1
✓ Fold3: preprocessor 1/1
i Fold3: preprocessor 1/1, model 1/1
✓ Fold3: preprocessor 1/1, model 1/1
i Fold3: preprocessor 1/1, model 1/1 (predictions)
i Fold4: preprocessor 1/1
✓ Fold4: preprocessor 1/1
i Fold4: preprocessor 1/1, model 1/1
✓ Fold4: preprocessor 1/1, model 1/1
i Fold4: preprocessor 1/1, model 1/1 (predictions)
i Fold5: preprocessor 1/1
✓ Fold5: preprocessor 1/1
i Fold5: preprocessor 1/1, model 1/1
✓ Fold5: preprocessor 1/1, model 1/1
i Fold5: preprocessor 1/1, model 1/1 (predictions)
```

Tune

We can also directly specify the metrics that are calculated within each resample

```
library(tune)

linear_fold_fits <- fit_resamples(
    linear_wf,
    resamples = mtcars_folds,
    metrics = metric_set(rmse, rsq, mase)
)

collect_metrics(linear_fold_fits)
```

```
## # A tibble: 3 x 6
##   .metric .estimator  mean     n std_err .config
##   <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1 mase    standard   0.338     4  0.0384 Preprocessor1_Model1
## 2 rmse    standard   2.83      4  0.242  Preprocessor1_Model1
## 3 rsq     standard   0.852     4  0.0400 Preprocessor1_Model1
```