Resampling

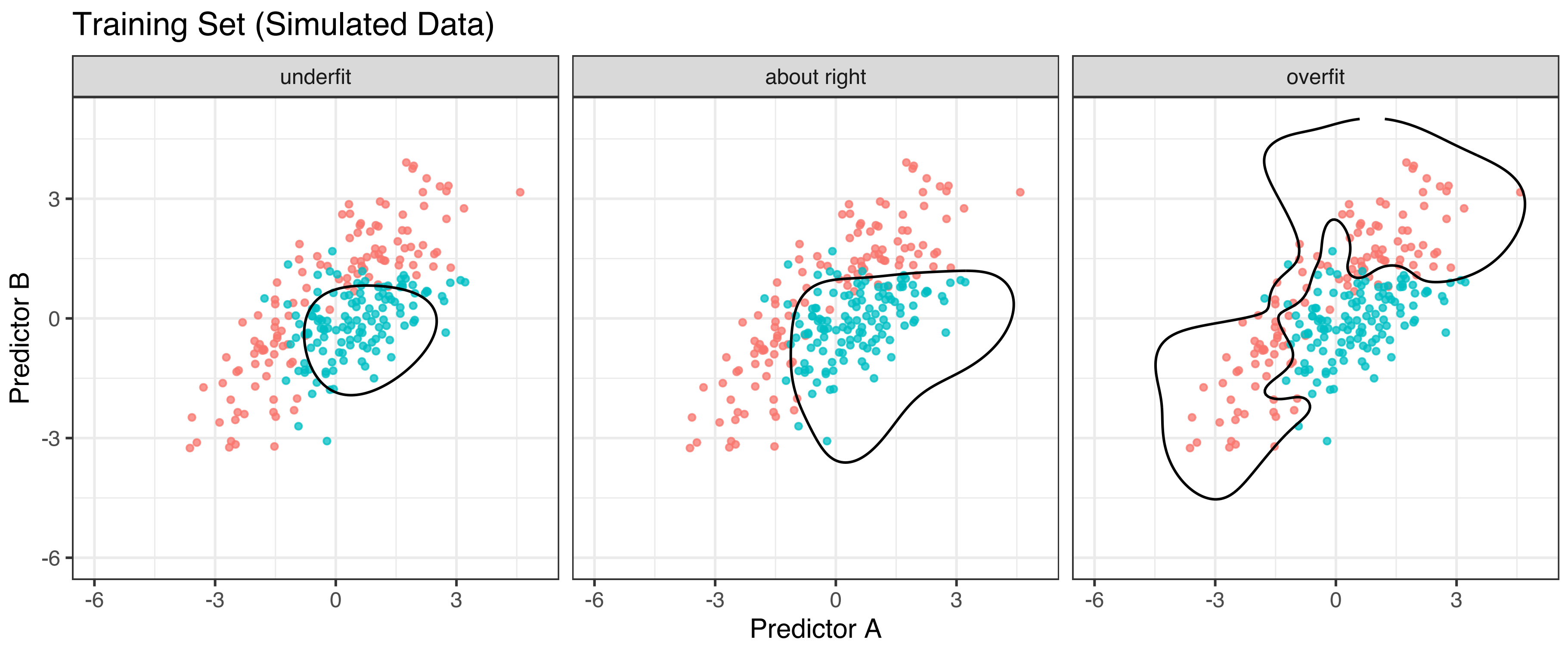
(based on tmwr.org)

Matthew McDonald

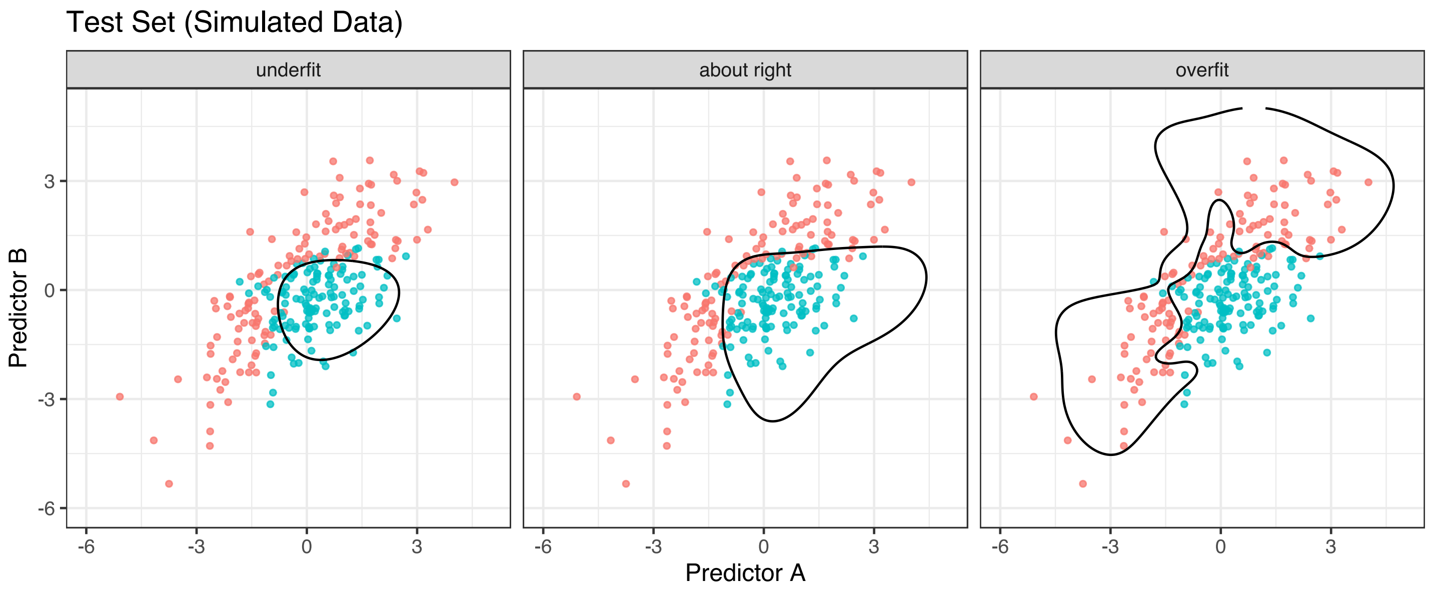
Resampling

- We have already covered the idea of data spending, and we recommended the test set for obtaining an unbiased estimate of performance.
- However, we usually need to understand the performance of a model or even multiple models *before using the test set*.
- **Resampling** is a technique for creating estimates of performance can generalize to new data in a similar way as estimates from a test set.

Dangers of Overfitting



Dangers of Overfitting



We call this “resubstitution” or “repredicting the training set”

What if we want to compare more models?

And/or more model configurations?

And we want to understand if these are important differences?

Resubstitution Example

When we measure performance on the same data that we used for training (as opposed to new data or testing data), we say we have *resubstituted* the data.

We can compare the linear model we built in the Feature Engineering section to a model created using a different algorithm called *Random forests*.

Random Forests

- **Random forests**: A tree ensemble method that operates by creating a large number of decision trees from slightly different versions of the training set.
- When predicting a new sample, each ensemble member makes a separate prediction that are averaged to create the final ensemble prediction for the new data point.
- Can emulate the underlying data patterns very closely.
- Computationally intensive
- Low maintenance; very little preprocessing is required

Creating the Random Forest Model

```r
rf_model <-
  rand_forest(trees = 1000) %>%
  set_engine("ranger") %>%
  set_mode("regression")

rf_wflow <-
  workflow() %>%
  add_formula(
    Sale_Price ~ Neighborhood + Gr_Liv_Area + Year_Built + Bldg_Type +
      Latitude + Longitude) %>%
  add_model(rf_model)

rf_fit <- rf_wflow %>% fit(data = ames_train)
```

A Function to Evaluate Performance

```r
estimate_perf <- function(model, dat) {
  # Capture the names of the `model` and `dat` objects
  cl <- match.call()
  obj_name <- as.character(cl$model)
  data_name <- as.character(cl$dat)
  data_name <- gsub("ames_", "", data_name)

  # Estimate these metrics:
  reg_metrics <- metric_set(rmse, rsq)

  model %>%
    predict(dat) %>%
    bind_cols(dat %>% select(Sale_Price)) %>%
    reg_metrics(Sale_Price, .pred) %>%
    select(-.estimator) %>%
    mutate(object = obj_name, data = data_name)
}
```

Comparing Models

```r
estimate_perf(rf_fit,
              ames_train)
```

```
# A tibble: 2 × 4
  .metric .estimate object data 
  <chr>       <dbl> <chr>  <chr>
1 rmse       0.0367 rf_fit train
2 rsq        0.959  rf_fit train
```

```r
estimate_perf(lm_fit,
              ames_train)
```

```
# A tibble: 2 × 4
  .metric .estimate object data 
  <chr>       <dbl> <chr>  <chr>
1 rmse       0.0754 lm_fit train
2 rsq        0.816  lm_fit train
```

Using the Random Forest on the Test Data

```r
estimate_perf(rf_fit, ames_test)
```

```
# A tibble: 2 × 4
  .metric .estimate object data 
  <chr>       <dbl> <chr>  <chr>
1 rmse       0.0704 rf_fit test 
2 rsq        0.852  rf_fit test 
```

Wait…what happened? The Random Forest Model got way **worse**!

Low Bias Models

- Many predictive models are capable of learning complex trends from the data. In statistics, these are commonly referred to as *low bias models*.
- *bias* is the difference between the true pattern or relationships in data and the types of patterns that the model can emulate. Many black-box machine learning models have low bias, meaning they can reproduce complex relationships.
- Other models (such as linear/logistic regression, discriminant analysis, and others) are not as adaptable and are considered *high bias* models.
- For a low bias model, the high degree of predictive capacity can sometimes result in the model nearly memorizing the training set data.

Summary of Model Performance

For both models, The following table summarizes the RMSE estimate for the training and test sets:

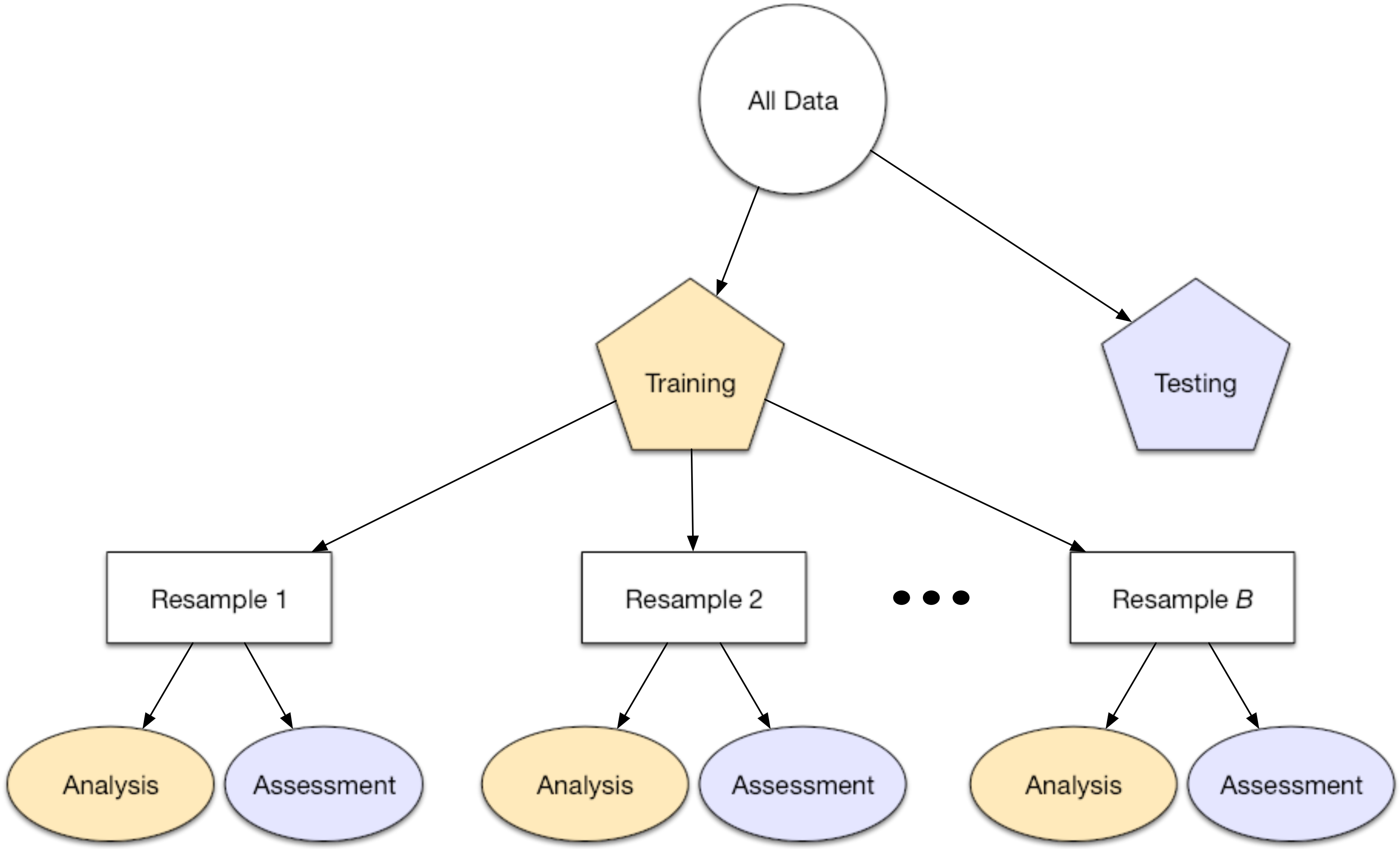
Performance statistics for training and test sets.

	RMSE Estimates	
object	**train**	**test**
lm_fit	0.0754450	0.0736297
rf_fit	0.0367377	0.0704400

Resampling

- **Resampling methods**: Empirical simulation systems that emulate the process of using some data for modeling and different data for evaluation.
- Mostly iterative processes repeated multiple times.

Resampling Visualization



Data splitting scheme from the initial data split to resampling

Resampling

Resampling is conducted only on the training 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*.

These two subsamples are somewhat analogous to training and test sets. The language of *analysis* and *assessment* avoids confusion with the initial split of the data. These data sets are mutually exclusive. The partitioning scheme used to create the analysis and assessment sets is usually the defining characteristic of the method.

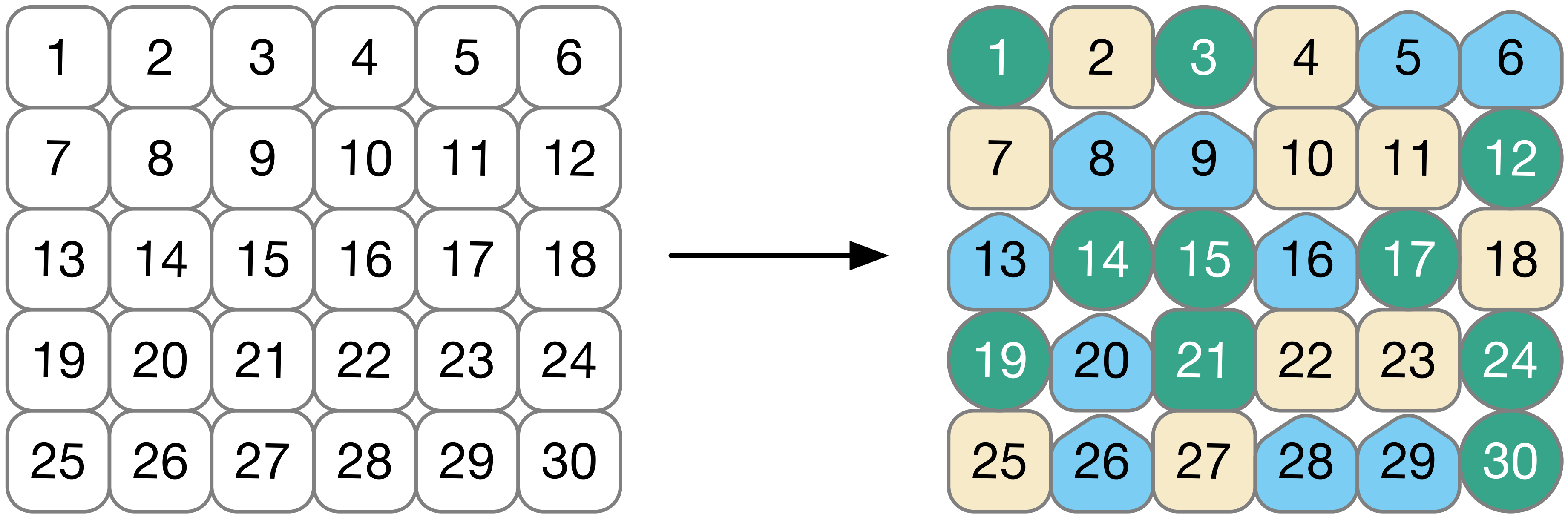
Using Resampled Data

- Suppose 20 iterations of resampling are conducted.
- 20 separate models are fit on the analysis sets
- The corresponding assessment sets produce 20 sets of performance statistics.
- The final estimate of performance for a model is the average of the 20 replicates of the statistics.
- This average has very good generalization properties and is far better than the resubstitution estimates.

Cross-validation

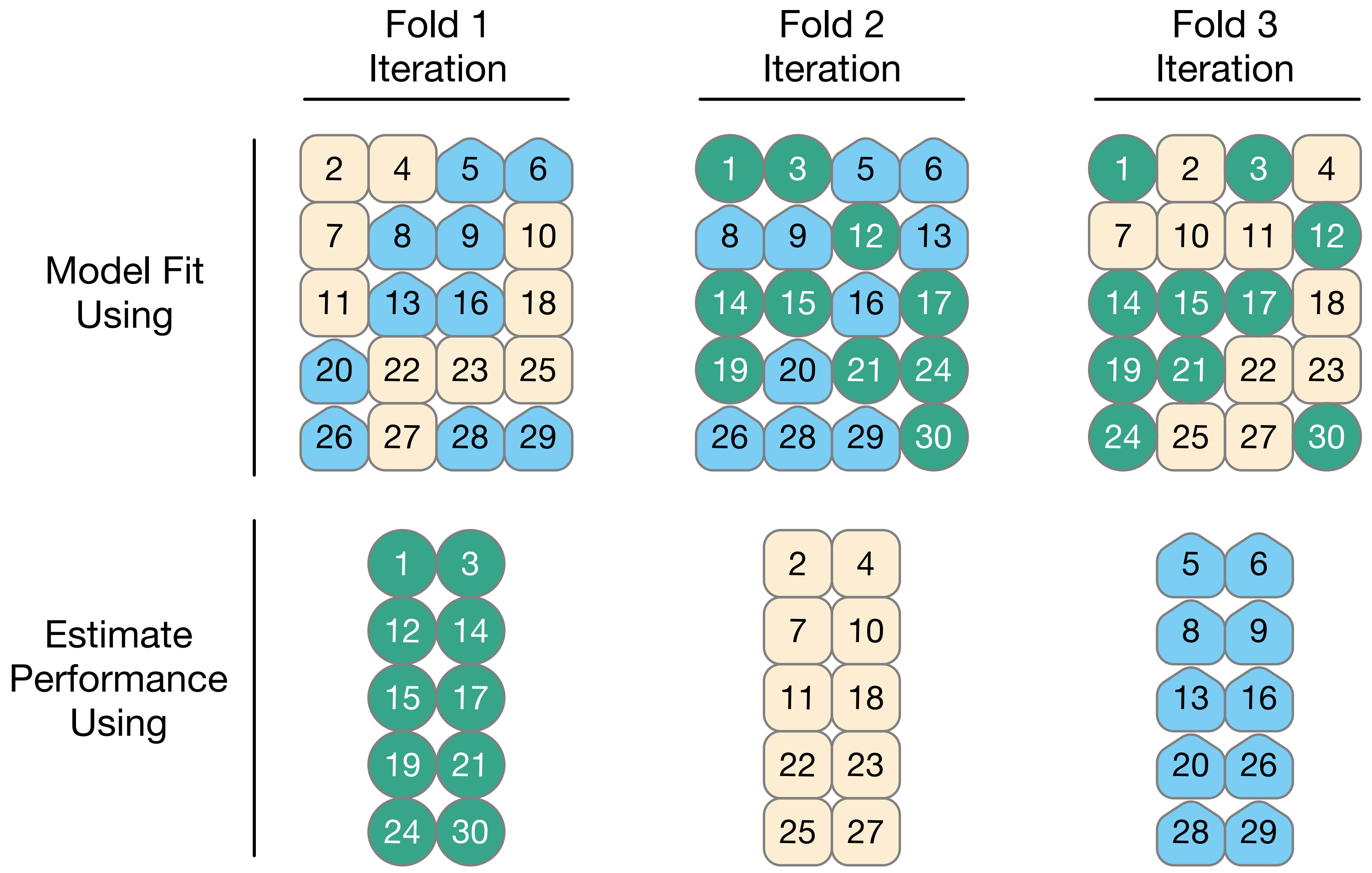
- **Cross-validation**: A well established resampling method
- Most common cross-validation method is *V*-fold cross-validation
- Data are randomly partitioned into *V* sets of roughly equal size (called the **folds**)
- Stratified sampling is also an option for assigning folds

V = 3



V-fold cross-validation randomly assigns data to folds

Using the Folds



V-fold cross-validation data usage

Choosing V

- Values of *V* are most often 5 or 10
- 10-fold cross-validation is preferred as a default because it is large enough for good results in most situations
- Larger values for *V* result in resampling estimates with small bias but substantial variance
- Smaller values of *V* have large bias but low variance
- 10-fold is preferred since noise is reduced by replication, but bias is not

Cross Validation with rsample

```r
set.seed(1001)
ames_folds <- vfold_cv(ames_train, v = 10)
ames_folds
```

```
#  10-fold cross-validation
# A tibble: 10 × 2
   splits             id
   <list>             <chr>
 1 <split [2107/235]> Fold01
 2 <split [2107/235]> Fold02
 3 <split [2108/234]> Fold03
 4 <split [2108/234]> Fold04
 5 <split [2108/234]> Fold05
 6 <split [2108/234]> Fold06
 7 <split [2108/234]> Fold07
 8 <split [2108/234]> Fold08
 9 <split [2108/234]> Fold09
10 <split [2108/234]> Fold10
```

Retrieving the Resampled Data

The `analysis()` and `assessment()` functions return the corresponding data frames:

```r
# For the first fold:
ames_folds$splits[[1]] %>% analysis() %>% dim()
```

```
[1] 2107   74
```

The functions that use the resampled data contain high-level user interfaces so that functions like `analysis()` are not generally needed for day-to-day work.

Repeated Cross-Validation

- The most important variation on cross-validation
- Depending on data size or other characteristics, the resampling estimate produced by V-fold cross-validation may be excessively noisy
- One way to reduce noise is to gather more data. For cross-validation, this means averaging more than V statistics

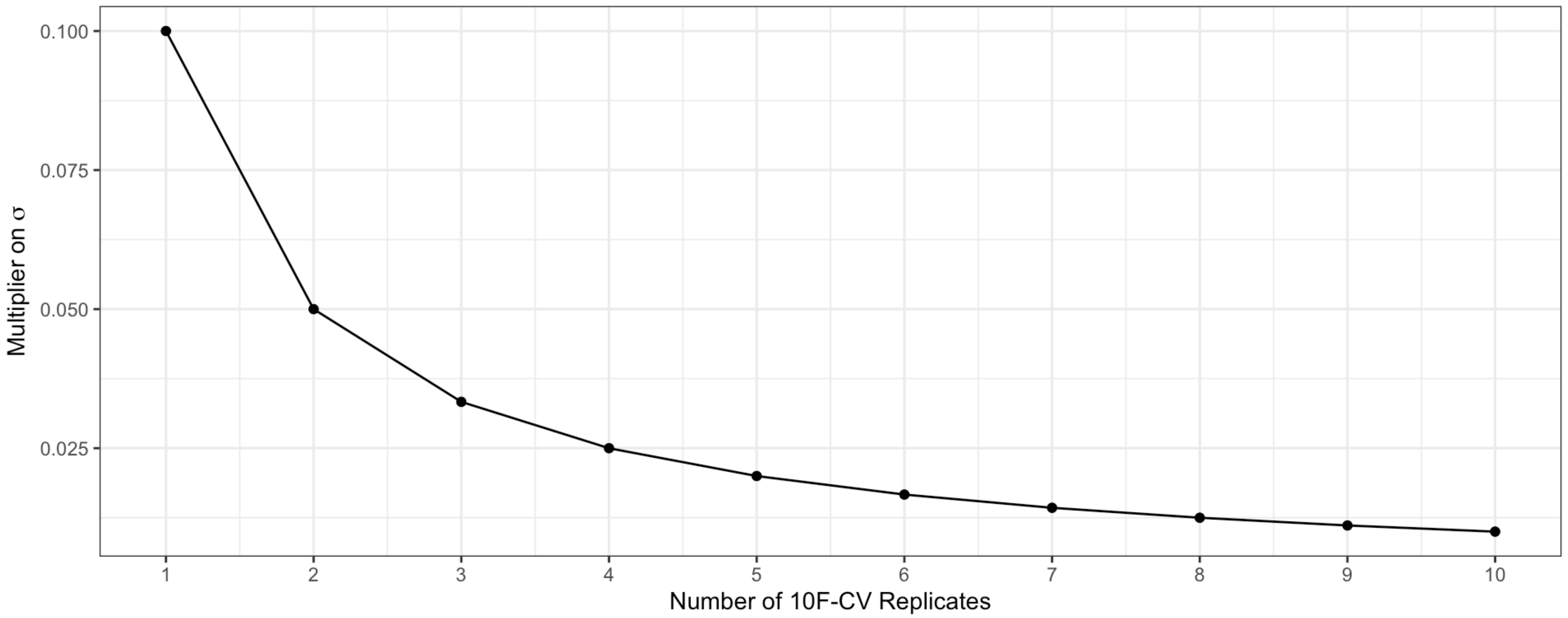
Repeated Cross-Validation

- To create R repeats of V-fold cross-validation, the same fold generation process is done R times to generate R collections of V partitions.
- Instead of averaging V statistics, \(V \times R\) statistics produce the final resampling estimate.
- Due to the Central Limit Theorem, the summary statistics from each model tend toward a normal distribution, as long as we have a lot of data relative to \(V \times R\).

Impact in the Ames Analysis

- On average, 10-fold cross-validation uses assessment sets that contain roughly 234 properties
- If RMSE is the statistic of choice, we can denote that estimate's standard deviation as \(\sigma\)
- With simple 10-fold cross-validation, the standard error of the mean RMSE is \(\sigma/\sqrt{10}\)
- Repeats reduce the standard error to \(\sigma/\sqrt{10R}\)

Reducing the Noise



Relationship between the relative variance in performance estimates versus the number of cross-validation repeats

Creating Repeated Cross Validation Datasets

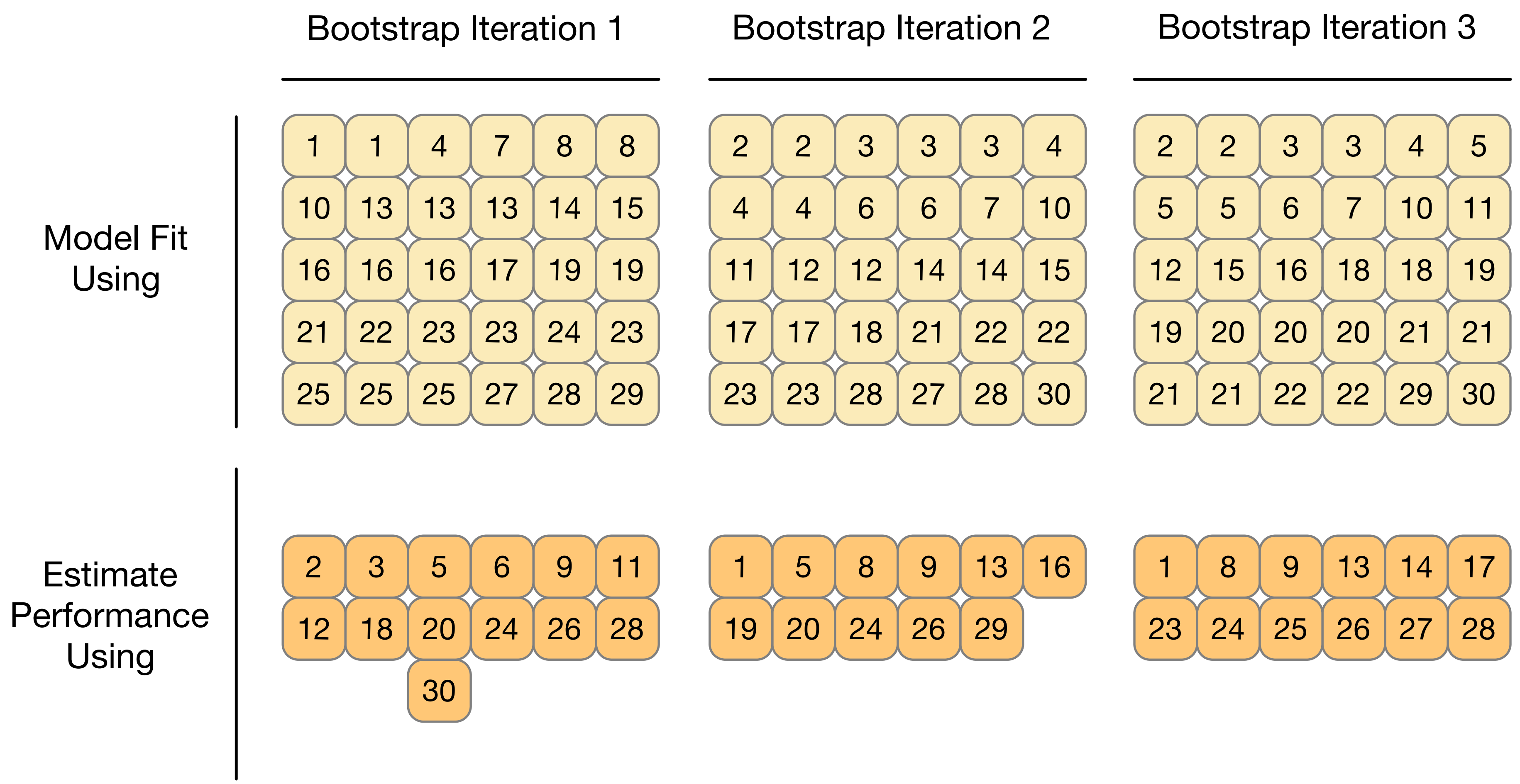
```r
vfold_cv(ames_train, v = 10, repeats = 5)
```

```
#  10-fold cross-validation repeated 5 times
# A tibble: 50 × 3
   splits             id      id2
   <list>             <chr>   <chr>
 1 <split [2107/235]> Repeat1 Fold01
 2 <split [2107/235]> Repeat1 Fold02
 3 <split [2108/234]> Repeat1 Fold03
 4 <split [2108/234]> Repeat1 Fold04
 5 <split [2108/234]> Repeat1 Fold05
 6 <split [2108/234]> Repeat1 Fold06
 7 <split [2108/234]> Repeat1 Fold07
 8 <split [2108/234]> Repeat1 Fold08
 9 <split [2108/234]> Repeat1 Fold09
10 <split [2108/234]> Repeat1 Fold10
# ℹ 40 more rows
```

Bootstrapping

- **Bootstrap sample**: a sample that is the same size as the training set but is drawn *with replacement*
- Some training set data points are selected multiple times for the analysis set
- Each data point has a 63.2% chance of inclusion in the training set at least once
- The assessment set contains all of the training set samples that were not selected for the analysis set (on average, with 36.8% of the training set)
- When bootstrapping, the assessment set is often called the *out-of-bag* sample
- Originally invented as a method for approximating the sampling distribution of statistics whose theoretical properties are intractable
- Produce performance estimates that have very low variance (unlike cross-validation) but have significant pessimistic bias

Bootstrapping



Bootstrapping data usage

Creating a Bootstrap Sample

Using the `rsample` package, we can create such bootstrap resamples:

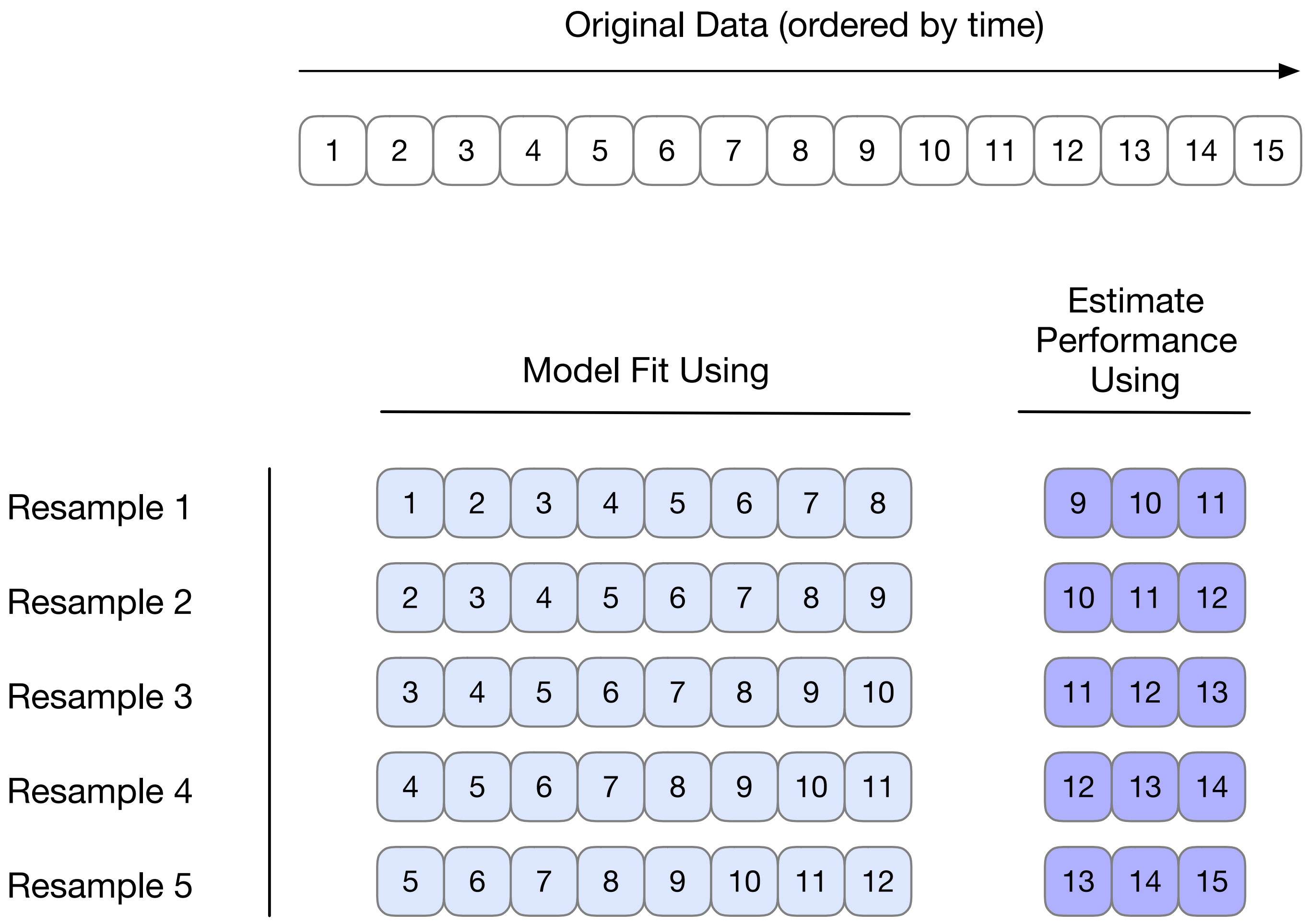
```r
bootstraps(ames_train, times = 5)
```

```
# Bootstrap sampling
# A tibble: 5 × 2
  splits             id
  <list>             <chr>
1 <split [2342/882]> Bootstrap1
2 <split [2342/882]> Bootstrap2
3 <split [2342/858]> Bootstrap3
4 <split [2342/877]> Bootstrap4
5 <split [2342/837]> Bootstrap5
```

Rolling forecasting origin resampling

- **Rolling forecast origin resampling**: Estimates the model with historical data and evaluating it with the most recent data.
- The size of the initial analysis and assessment sets are specified.
- The first iteration of resampling uses these sizes, starting from the beginning of the series.
- The second iteration uses the same data sizes but shifts over by a set number of samples.

Rolling forecasting origin resampling



Data usage for rolling forecasting origin resampling

Rolling forecasting origin resampling

Two different configurations of this method:

- The analysis set can cumulatively grow (as opposed to remaining the same size). After the first initial analysis set, new samples can accrue without discarding the earlier data.
- The resamples need not increment by one. For example, for large data sets, the incremental block could be a week or month instead of a day.

For a year's worth of data, suppose that six sets of 30-day blocks define the analysis set. For assessment sets of 30 days with a 29-day skip, we can use the `rsample` package to specify

Implementation

```r
time_slices <-
  tibble(x = 1:365) %>%
  rolling_origin(initial = 6 * 30,
                 assess = 30,
                 skip = 29,
                 cumulative = FALSE)

data_range <- function(x) {
  summarize(x, first = min(x), last = max(x))
}
```

Implementation

```r
map_dfr(time_slices$splits,
        ~   analysis(.x) %>%
          data_range())
```

```
# A tibble: 6 × 2
  first  last
  <int> <int>
1     1   180
2    31   210
3    61   240
4    91   270
5   121   300
6   151   330
```

```r
map_dfr(time_slices$splits,
        ~ assessment(.x) %>%
          data_range())
```

```
# A tibble: 6 × 2
  first  last
  <int> <int>
1   181   210
2   211   240
3   241   270
4   271   300
5   301   330
6   331   360
```

Other Resampling Approaches

- Leave One Out Cross-Validation
  - Extreme version of V-fold cross-validation where the assessment sets are only 1 observation
- Monte Carlo Cross-Validation
  - Like V-fold cross-validation, it allocates a fixed proportion of data to the assessment sets. The difference between MCCV and regular cross-validation is that, for MCCV, this proportion of the data is randomly selected each time
- Validation Set
  - A single partition that is set aside to estimate performance separate from the test set

Estimating Performance

Resampling methods are effective because different groups of data are used to train the model and assess the model. To reiterate, the process to use resampling is:

1. During resampling, the analysis set is used to preprocess the data, apply the preprocessing to itself, and use these processed data to fit the model.
2. The preprocessing statistics produced by the analysis set are applied to the assessment set. The predictions from the assessment set estimate performance on new data.

This sequence repeats for every resample. If there are B resamples, there are B replicates of each of the performance metrics. The final resampling estimate is the average of these B statistics.

Resampling Usage

```
model_spec %>% fit_resamples(formula,  resamples, ...)
model_spec %>% fit_resamples(recipe,   resamples, ...)
workflow   %>% fit_resamples(          resamples, ...)
```

Optional Arguments

There are a number of other optional arguments, such as:

- `metrics`: A metric set of performance statistics to compute. By default, regression models use RMSE and \(R^2\) while classification models compute the area under the ROC curve and overall accuracy.
- `control`: A list created by `control_resamples()` with various options.

Control Options

- `verbose`: A logical for printing logging.
- `extract`: A function for retaining objects from each model iteration (discussed later in this chapter).
- `save_pred`: A logical for saving the assessment set predictions.

Resampled Performance of RF Model

```r
keep_pred <- control_resamples(save_pred = TRUE, save_workflow = TRUE)

set.seed(1003)
rf_res <-
  rf_wflow %>%
  fit_resamples(resamples = ames_folds, control = keep_pred)
rf_res
```

```
# Resampling results
# 10-fold cross-validation
# A tibble: 10 × 5
   splits             id     .metrics         .notes           .predictions
   <list>             <chr>  <list>           <list>           <list>
 1 <split [2107/235]> Fold01 <tibble [2 × 4]> <tibble [0 × 3]> <tibble>
 2 <split [2107/235]> Fold02 <tibble [2 × 4]> <tibble [0 × 3]> <tibble>
 3 <split [2108/234]> Fold03 <tibble [2 × 4]> <tibble [0 × 3]> <tibble>
 4 <split [2108/234]> Fold04 <tibble [2 × 4]> <tibble [0 × 3]> <tibble>
 5 <split [2108/234]> Fold05 <tibble [2 × 4]> <tibble [0 × 3]> <tibble>
 6 <split [2108/234]> Fold06 <tibble [2 × 4]> <tibble [0 × 3]> <tibble>
 7 <split [2108/234]> Fold07 <tibble [2 × 4]> <tibble [0 × 3]> <tibble>
 8 <split [2108/234]> Fold08 <tibble [2 × 4]> <tibble [0 × 3]> <tibble>
 9 <split [2108/234]> Fold09 <tibble [2 × 4]> <tibble [0 × 3]> <tibble>
10 <split [2108/234]> Fold10 <tibble [2 × 4]> <tibble [0 × 3]> <tibble>
```

Return Values

The return value is a tibble similar to the input resamples, along with some extra columns:

- `.metrics` is a list column of tibbles containing the assessment set performance statistics.
- `.notes` is another list column of tibbles cataloging any warnings or errors generated during resampling. Note that errors will not stop subsequent execution of resampling.
- `.predictions` is present when `save_pred = TRUE`. This list column contains tibbles with the out-of-sample predictions.

Using the resampled Results

```
collect_metrics(rf_res)
```

```
# A tibble: 2 × 6
  .metric .estimator   mean     n std_err .config
  <chr>   <chr>       <dbl> <int>   <dbl> <chr>
1 rmse    standard   0.0721    10 0.00306 Preprocessor1_Model1
2 rsq     standard   0.832     10 0.0108  Preprocessor1_Model1
```

These are the resampling estimates averaged over the individual replicates. To get the metrics for each resample, use the option `summarize = FALSE`.

Notice how much more realistic the performance estimates are than the resubstitution estimates.