Resampling and Cross-Validation

Nate Wells

Math 243: Stat Learning

September 27th, 2021

Outline

In today's class, we will. . .

- Define and discuss resampling and cross-validation
- Investigate methods of cross-validation (LOOCV and k-fold cv)
- Implement CV in R

Section 1

Cross Validation

Poll: Training Error

Which of the following methods are likely to have the smallest training error rate for regression problems?

a. Multiple linear regression
b. Simple linear regression
c. Non-linear regression with polynomials
d. KNN with $K = 1$
e. KNN with $K = p$

Validation Set

Assessing model accuracy only on training sets will usually under-estimate error

Validation Set

Assessing model accuracy only on training sets will usually under-estimate error

- But not all models will have the same bias, making comparison difficult

Validation Set

Assessing model accuracy only on training sets will usually under-estimate error

- But not all models will have the same bias, making comparison difficult

One fix is to partition data into training and test sets.

Validation Set

Assessing model accuracy only on training sets will usually under-estimate error

- But not all models will have the same bias, making comparison difficult

One fix is to partition data into training and test sets.

- Build the model using only the training data
- Assess accuracy using only test data.

Fuel Economy

The `FuelEconomy` data set from the `AppliedPredictiveModeling` package contains fuel efficiency and other variables for 1107 cars and trucks from 2010, with data taken from the http://fueleconomy.gov website

```
library(AppliedPredictiveModeling)
data(FuelEconomy)
head(cars2010)
```

```
##      EngDispl NumCyl Transmission      FE AirAspirationMethod NumGears
## 1088      4.7      8          AM6 28.0198  NaturallyAspirated        6
## 1089      4.7      8           M6 25.6094  NaturallyAspirated        6
## 1090      4.2      8           M6 26.8000  NaturallyAspirated        6
## 1091      4.2      8          AM6 25.0451  NaturallyAspirated        6
## 1092      5.2     10          AM6 24.8000  NaturallyAspirated        6
## 1093      5.2     10           M6 23.9000  NaturallyAspirated        6
##      TransLockup TransCreeperGear         DriveDesc IntakeValvePerCyl
## 1088           1                0 TwoWheelDriveRear                 2
## 1089           1                0 TwoWheelDriveRear                 2
## 1090           1                0     AllWheelDrive                 2
## 1091           1                0     AllWheelDrive                 2
## 1092           0                0     AllWheelDrive                 2
## 1093           0                0     AllWheelDrive                 2
##      ExhaustValvesPerCyl CarlineClassDesc VarValveTiming VarValveLift
## 1088                   2         2Seaters              1            0
## 1089                   2         2Seaters              1            0
## 1090                   2         2Seaters              1            0
## 1091                   2         2Seaters              1            0
## 1092                   2         2Seaters              1            0
## 1093                   2         2Seaters              1            0
```

Important Predictors

Let's consider just numeric variable first:

```
cars2010 %>%
  select_if(is.numeric) %>%
  cor(cars2010$FE)
```

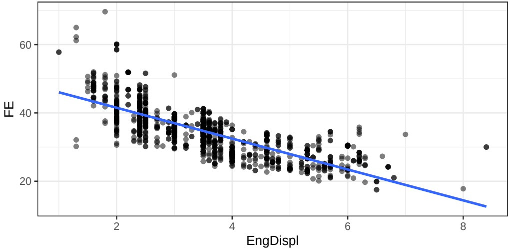
```
##                          [,1]
## EngDispl          -0.78739383
## NumCyl            -0.74021798
## FE                 1.00000000
## NumGears          -0.21128488
## TransLockup       -0.27193887
## TransCreeperGear  -0.06962168
## IntakeValvePerCyl  0.28034403
## ExhaustValvesPerCyl 0.33565285
## VarValveTiming     0.12495278
## VarValveLift       0.09621127
```

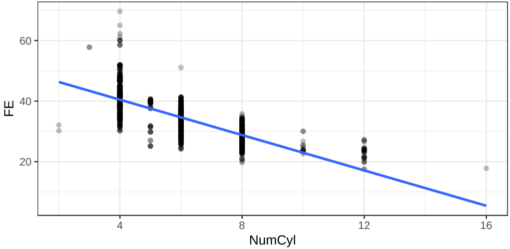
Important Predictors

Let's consider just numeric variable first:

```
cars2010 %>%
  select_if(is.numeric) %>%
  cor(cars2010$FE)
```

```
##                           [,1]
## EngDispl           -0.78739383
## NumCyl             -0.74021798
## FE                  1.00000000
## NumGears           -0.21128488
## TransLockup        -0.27193887
## TransCreeperGear   -0.06962168
## IntakeValvePerCyl   0.28034403
## ExhaustValvesPerCyl 0.33565285
## VarValveTiming      0.12495278
## VarValveLift        0.09621127
```



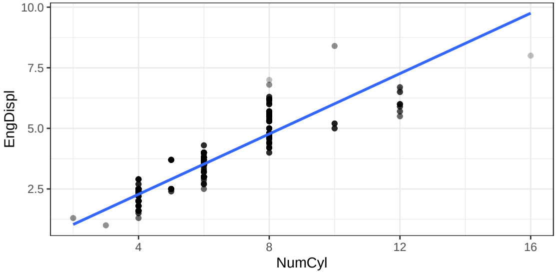


Collinearity

- We may want to include both `EngDispl` and `NumCyl` in our model for `FE`.
    - But if both are strongly correlated with `FE`, they may also be strongly correlated with each other. . .

Collinearity

- We may want to include both `EngDispl` and `NumCyl` in our model for `FE`.
    - But if both are strongly correlated with `FE`, they may also be strongly correlated with each other...




```r
cor(cars2010$EngDispl, cars2010$NumCyl)
```

```
## [1] 0.90626
```

Validation Set

Let's create a validation set using `initial_split` in the `rsample` package

Validation Set

Let's create a validation set using `initial_split` in the `rsample` package

```
library(rsample)
set.seed(999)
cars_initial <- initial_split(cars2010)
cars_train <- training(cars_initial)
cars_val <- testing(cars_initial)
```

Validation Set

Let's create a validation set using `initial_split` in the `rsample` package

```
library(rsample)
set.seed(999)
cars_initial <- initial_split(cars2010)
cars_train <- training(cars_initial)
cars_val <- testing(cars_initial)
```

- The `dim` function in `rsample` returns the number of observations and variables present in a split:

```
cars_train %>% dim()
```

```
## [1] 831  14
```

```
cars_val %>% dim()
```

```
## [1] 276  14
```

Two Models

- Since `EngDispl` is most strongly correlated with `FE`, we will include it in our models.
- And we'll create another model that also includes `NumCyl`.

Two Models

- Since `EngDispl` is most strongly correlated with `FE`, we will include it in our models.
- And we'll create another model that also includes `NumCyl`.

```
mod1 <- lm(FE ~ EngDispl, data = cars_train)
summary(mod1)
```

```
##
## Call:
## lm(formula = FE ~ EngDispl, data = cars_train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -12.6152  -3.2808  -0.4195   2.6322  27.1747
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  50.5639     0.4667  108.34   <2e-16 ***
## EngDispl     -4.4990     0.1252  -35.92   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.648 on 829 degrees of freedom
## Multiple R-squared:  0.6088, Adjusted R-squared:  0.6084
## F-statistic:  1290 on 1 and 829 DF,  p-value: < 2.2e-16
```

Two Models

- Since `EngDispl` is most strongly correlated with `FE`, we will include it in our models.
- And we'll create another model that also includes `NumCyl`.

```
mod1 <- lm(FE ~ EngDispl, data = cars_train)
summary(mod1)
```

```
##
## Call:
## lm(formula = FE ~ EngDispl, data = cars_train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -12.6152  -3.2808  -0.4195   2.6322  27.1747
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  50.5639     0.4667  108.34   <2e-16 ***
## EngDispl     -4.4990     0.1252  -35.92   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.648 on 829 degrees of freedom
## Multiple R-squared:  0.6088, Adjusted R-squared:  0.6084
## F-statistic:  1290 on 1 and 829 DF,  p-value: < 2.2e-16
```

```
mod2 <- lm(FE ~ EngDispl + NumCyl, data = cars_train)
summary(mod2)
```

```
##
## Call:
## lm(formula = FE ~ EngDispl + NumCyl, data = cars_train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -13.4549  -3.1297  -0.3406   2.5093  27.3736
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  51.6430     0.5480  94.246  < 2e-16 ***
## EngDispl     -3.5000     0.2982 -11.738  < 2e-16 ***
## NumCyl       -0.7691     0.2086  -3.686 0.000242 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.613 on 828 degrees of freedom
## Multiple R-squared:  0.6151, Adjusted R-squared:  0.6142
## F-statistic: 661.7 on 2 and 828 DF,  p-value: < 2.2e-16
```

Two Models

- Since `EngDispl` is most strongly correlated with `FE`, we will include it in our models.
- And we'll create another model that also includes `NumCyl`.

```
mod1 <- lm(FE ~ EngDispl, data = cars_train)
summary(mod1)
```

```
##
## Call:
## lm(formula = FE ~ EngDispl, data = cars_train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -12.6152  -3.2808  -0.4195   2.6322  27.1747
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  50.5639     0.4667  108.34   <2e-16 ***
## EngDispl     -4.4990     0.1252  -35.92   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.648 on 829 degrees of freedom
## Multiple R-squared:  0.6088, Adjusted R-squared:  0.6084
## F-statistic:  1290 on 1 and 829 DF,  p-value: < 2.2e-16
```

```
mod2 <- lm(FE ~ EngDispl + NumCyl, data = cars_train)
summary(mod2)
```

```
##
## Call:
## lm(formula = FE ~ EngDispl + NumCyl, data = cars_train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -13.4549  -3.1297  -0.3406   2.5093  27.3736
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  51.6430     0.5480  94.246  < 2e-16 ***
## EngDispl     -3.5000     0.2982 -11.738  < 2e-16 ***
## NumCyl       -0.7691     0.2086  -3.686 0.000242 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.613 on 828 degrees of freedom
## Multiple R-squared:  0.6151, Adjusted R-squared:  0.6142
## F-statistic: 661.7 on 2 and 828 DF,  p-value: < 2.2e-16
```

- The MLR model has lower RSE, higher R^2, and all predictors are significant. But is it really the better model?

Assess on Validation Set

Let's check MSE on the validation set.

Assess on Validation Set

Let's check MSE on the validation set.

```
mod1_preds <- predict(mod1, cars_val)
mod1_mse <- mean( (cars2011$FE - mod1_preds)^2)
mod1_mse
```

```
## [1] 115.4423
```

Assess on Validation Set

Let's check MSE on the validation set.

```
mod1_preds <- predict(mod1, cars_val)
mod1_mse <- mean( (cars2011$FE - mod1_preds)^2)
mod1_mse
```

```
## [1] 115.4423
```

```
mod2_preds <- predict(mod2, cars_val)
mod2_mse <- mean( (cars2011$FE - mod2_preds)^2)
mod2_mse
```

```
## [1] 117.9783
```

Assess on Validation Set

Let's check MSE on the validation set.

```
mod1_preds <- predict(mod1, cars_val)
mod1_mse <- mean( (cars2011$FE - mod1_preds)^2)
mod1_mse
```

```
## [1] 115.4423
```

```
mod2_preds <- predict(mod2, cars_val)
mod2_mse <- mean( (cars2011$FE - mod2_preds)^2)
mod2_mse
```

```
## [1] 117.9783
```

- The MLR model (`mod2`) now has slightly higher MSE than the SLR model (`mod1`)
  - But could this be a fluke of a random validation set?

Problems with Validation Sets

What are some problems with the Training / Validation approach?

Problems with Validation Sets

What are some problems with the Training / Validation approach?

- If initial data set is small, this further restricts sample size available for model building.
    - Both model and test performance may have high variance.

Problems with Validation Sets

What are some problems with the Training / Validation approach?

- If initial data set is small, this further restricts sample size available for model building.
  - Both model and test performance may have high variance.
- A single test set doesn't give estimates for the range of error

Problems with Validation Sets

What are some problems with the Training / Validation approach?

- If initial data set is small, this further restricts sample size available for model building.
  - Both model and test performance may have high variance.
- A single test set doesn't give estimates for the range of error
- Susceptible to bias from particular choice of training set.

Problems with Validation Sets

What are some problems with the Training / Validation approach?

- If initial data set is small, this further restricts sample size available for model building.
    - Both model and test performance may have high variance.
- A single test set doesn't give estimates for the range of error
- Susceptible to bias from particular choice of training set.

Resampling is drawing many samples from your training data and refitting the model for each, in order to learn more about your model.

Problems with Validation Sets

What are some problems with the Training / Validation approach?

- If initial data set is small, this further restricts sample size available for model building.
    - Both model and test performance may have high variance.
- A single test set doesn't give estimates for the range of error
- Susceptible to bias from particular choice of training set.

Resampling is drawing many samples from your training data and refitting the model for each, in order to learn more about your model.

Cross-Validation is using resampling techniques to assess model accuracy.

Section 2

Cross-Validation

k-fold Cross Validation

- k-fold CV randomly partitions data into k sets of size n/k.
  - One subset of size n/k is chosen to be the validation set
  - Remaining $k-1$ subsets are used as training set to build the model.

k-fold Cross Validation

- k-fold CV randomly partitions data into k sets of size n/k.
  - One subset of size n/k is chosen to be the validation set
  - Remaining $k-1$ subsets are used as training set to build the model.
- The process is repeated for each possible validation set, and the average error rate computed among all partitions is computed

k-fold Cross Validation

- k-fold CV randomly partitions data into k sets of size n/k.
    - One subset of size n/k is chosen to be the validation set
    - Remaining $k-1$ subsets are used as training set to build the model.
- The process is repeated for each possible validation set, and the average error rate computed among all partitions is computed
- The cross-validation estimate $CV_{(k)}$ for average test MSE is therefore:

$$CV_{(k)} = \frac{1}{k} \sum_{i=1}^{k} \mathrm{MSE}_i$$

k-fold Cross Validation

- k-fold CV randomly partitions data into k sets of size n/k.
    - One subset of size n/k is chosen to be the validation set
    - Remaining $k-1$ subsets are used as training set to build the model.
- The process is repeated for each possible validation set, and the average error rate computed among all partitions is computed
- The cross-validation estimate $CV_{(k)}$ for average test MSE is therefore:

$$CV_{(k)} = \frac{1}{k}\sum_{i=1}^{k} \mathrm{MSE}_i$$

- Here, MSE_i is test MSE when the ith fold is used as validation set.

k-fold Cross Validation

- k-fold CV randomly partitions data into k sets of size n/k.
    - One subset of size n/k is chosen to be the validation set
    - Remaining $k-1$ subsets are used as training set to build the model.
- The process is repeated for each possible validation set, and the average error rate computed among all partitions is computed
- The cross-validation estimate $CV_{(k)}$ for average test MSE is therefore:

$$CV_{(k)} = \frac{1}{k}\sum_{i=1}^{k} \mathrm{MSE}_i$$

- Here, MSE_i is test MSE when the ith fold is used as validation set.
- Since the partition into folds is random, $CV_{(k)}$ still has some variability. But less than just using a single validation set.

k-fold Cross Validation

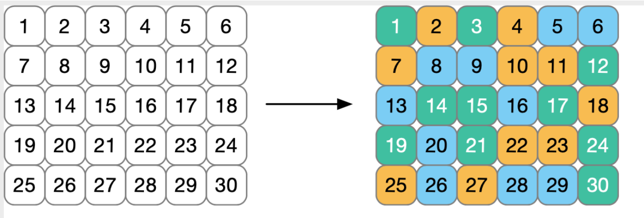
- k-fold CV randomly partitions data into k sets of size n/k.
  - One subset of size n/k is chosen to be the validation set
  - Remaining $k-1$ subsets are used as training set to build the model.
- The process is repeated for each possible validation set, and the average error rate computed among all partitions is computed
- The cross-validation estimate $CV_{(k)}$ for average test MSE is therefore:

$$CV_{(k)} = \frac{1}{k}\sum_{i=1}^{k} \mathrm{MSE}_i$$

- Here, MSE_i is test MSE when the ith fold is used as validation set.
- Since the partition into folds is random, $CV_{(k)}$ still has some variability. But less than just using a single validation set.
  - To reduce variability further, k-fold CV can be performed multiple times, and the results of $CV_{(k)}$ themselves averaged.

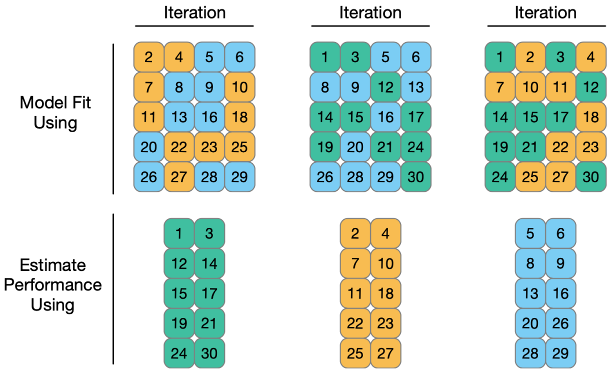
3-fold CV

- Consider 30 training observations below. Colors indicate a random fold allocation.



3-fold CV

- Each iteration uses 2 of the folds to build a model, and the remaining fold to assess performance.



- Overall performance is obtained by averaging across iterations.

CV in R

We'll use the `vfold_cv` function in `rsample` to perform cross-validation.

```
set.seed(927)
folds_cars <- vfold_cv(cars2010, v = 10)
```

CV in R

We'll use the `vfold_cv` function in `rsample` to perform cross-validation.

```
set.seed(927)
folds_cars <- vfold_cv(cars2010, v = 10)
```

- The above code breaks the data into 10 (nearly) equal folds and stores results as a resample object with 2 parts:
  - `id`, a vector with fold identifiers (i.e "Fold01", "Fold02", . . .)
  - `Splits`, a list whose elements correspond to each split of the data into $k - 1$ training and 1 validation sets

CV in R

We'll use the `vfold_cv` function in `rsample` to perform cross-validation.

```
set.seed(927)
folds_cars <- vfold_cv(cars2010, v = 10)
```

- The above code breaks the data into 10 (nearly) equal folds and stores results as a resample object with 2 parts:
  - `id`, a vector with fold identifiers (i.e "Fold01", "Fold02", . . .)
  - `Splits`, a list whose elements correspond to each split of the data into $k - 1$ training and 1 validation sets
- To get the training set for one iteration, we apply `analysis` to one of the `Splits` elements.

CV in R

We'll use the `vfold_cv` function in `rsample` to perform cross-validation.

```
set.seed(927)
folds_cars <- vfold_cv(cars2010, v = 10)
```

- The above code breaks the data into 10 (nearly) equal folds and stores results as a resample object with 2 parts:
    - `id`, a vector with fold identifiers (i.e "Fold01", "Fold02", . . .)
    - `Splits`, a list whose elements correspond to each split of the data into $k - 1$ training and 1 validation sets
- To get the training set for one iteration, we apply `analysis` to one of the `Splits` elements.

```
folds_cars$splits[[1]] %>% analysis()
```

CV in R

We'll use the `vfold_cv` function in `rsample` to perform cross-validation.

```
set.seed(927)
folds_cars <- vfold_cv(cars2010, v = 10)
```

- The above code breaks the data into 10 (nearly) equal folds and stores results as a resample object with 2 parts:
  - `id`, a vector with fold identifiers (i.e "Fold01", "Fold02", . . .)
  - `Splits`, a list whose elements correspond to each split of the data into $k - 1$ training and 1 validation sets
- To get the training set for one iteration, we apply `analysis` to one of the `Splits` elements.

```
folds_cars$splits[[1]] %>% analysis()
```

- And to get the corresponding validation set, we apply `assessment` to the same element

CV in R

We'll use the `vfold_cv` function in `rsample` to perform cross-validation.

```
set.seed(927)
folds_cars <- vfold_cv(cars2010, v = 10)
```

- The above code breaks the data into 10 (nearly) equal folds and stores results as a resample object with 2 parts:
  - `id`, a vector with fold identifiers (i.e "Fold01", "Fold02", . . .)
  - `Splits`, a list whose elements correspond to each split of the data into $k - 1$ training and 1 validation sets
- To get the training set for one iteration, we apply `analysis` to one of the `Splits` elements.

```
folds_cars$splits[[1]] %>% analysis()
```

- And to get the corresponding validation set, we apply `assessment` to the same element

```
folds_cars$splits[[1]] %>% assessment()
```

Create Functions

- The `rsample` package does the heavy lifting for partitioning data into appropriate folds.
  - But it doesn't actually build models or compute MSE
  - Let's practice coding!

Create Functions

- The `rsample` package does the heavy lifting for partitioning data into appropriate folds.
  - But it doesn't actually build models or compute MSE
  - Let's practice coding!
- **Goal:** Write function to do each of the following

1. Obtain analysis set
2. Fit linear model
3. Predict on assessment data
4. Assess accuracy

Create Functions

- The `rsample` package does the heavy lifting for partitioning data into appropriate folds.
  - But it doesn't actually build models or compute MSE
  - Let's practice coding!
- **Goal:** Write function to do each of the following

1. Obtain analysis set
2. Fit linear model
3. Predict on assessment data
4. Assess accuracy

```
cv_model1 <- function(split){
  mod <- lm(FE ~ EngDispl, data = analysis(split))
  val <- assessment(split)
  preds <- predict(mod, val)
  mse <- mean( (val$FE - preds)^2)
  mse
  }
```

Get Results!

- Now, we'll apply this function to each split in `folds_cars` using the `map_dbl` function from the `purrr` package

```
library(purrr)
folds_cars$mod1_results <- map_dbl(folds_cars$splits, cv_model1)
folds_cars %>% head()
```

```
## # A tibble: 6 x 3
##   splits             id     mod1_results
##   <list>             <chr>         <dbl>
## 1 <split [996/111]> Fold01         18.0
## 2 <split [996/111]> Fold02         17.1
## 3 <split [996/111]> Fold03         25.0
## 4 <split [996/111]> Fold04         25.9
## 5 <split [996/111]> Fold05         21.2
## 6 <split [996/111]> Fold06         16.4
```

Get Results!

- Now, we'll apply this function to each split in `folds_cars` using the `map_dbl` function from the `purrr` package

```
library(purrr)
folds_cars$mod1_results <- map_dbl(folds_cars$splits, cv_model1)
folds_cars %>% head()
```

```
## # A tibble: 6 x 3
##   splits             id     mod1_results
##   <list>             <chr>         <dbl>
## 1 <split [996/111]> Fold01          18.0
## 2 <split [996/111]> Fold02          17.1
## 3 <split [996/111]> Fold03          25.0
## 4 <split [996/111]> Fold04          25.9
## 5 <split [996/111]> Fold05          21.2
## 6 <split [996/111]> Fold06          16.4
```

- And to find the average CV MSE, we take the mean of the `results` column:

Get Results!

- Now, we'll apply this function to each split in `folds_cars` using the `map_dbl` function from the `purrr` package

```
library(purrr)
folds_cars$mod1_results <- map_dbl(folds_cars$splits, cv_model1)
folds_cars %>% head()
```

```
## # A tibble: 6 x 3
##   splits              id     mod1_results
##   <list>              <chr>         <dbl>
## 1 <split [996/111]> Fold01         18.0
## 2 <split [996/111]> Fold02         17.1
## 3 <split [996/111]> Fold03         25.0
## 4 <split [996/111]> Fold04         25.9
## 5 <split [996/111]> Fold05         21.2
## 6 <split [996/111]> Fold06         16.4
```

- And to find the average CV MSE, we take the mean of the `results` column:

```
CV_MSE_mod1 <- mean(folds_cars$mod1_results)
CV_MSE_mod1
```

```
## [1] 21.44501
```

Repeat

And now we repeat, but for `mod2`:

Repeat

And now we repeat, but for mod2:

```
cv_model2 <- function(split){
  mod <- lm(FE ~ EngDispl + NumCyl, data = analysis(split))
  val <- assessment(split)
  preds <- predict(mod, val)
  mse <- mean( (val$FE - preds)^2)
  mse
}

folds_cars$mod2_results <- map_dbl(folds_cars$splits, cv_model2)

CV_MSE_mod2 <- mean(folds_cars$mod2_results)
```

Repeat

And now we repeat, but for mod2:

```
cv_model2 <- function(split){
  mod <- lm(FE ~ EngDispl + NumCyl, data = analysis(split))
  val <- assessment(split)
  preds <- predict(mod, val)
  mse <- mean( (val$FE - preds)^2)
  mse
}

folds_cars$mod2_results <- map_dbl(folds_cars$splits, cv_model2)

CV_MSE_mod2 <- mean(folds_cars$mod2_results)
```

```
data.frame(model = c("1", "2"), cv_mse = c(CV_MSE_mod1, CV_MSE_mod2))
```

```
##   model   cv_mse
## 1     1 21.44501
## 2     2 21.26763
```

Repeat

And now we repeat, but for mod2:

```
cv_model2 <- function(split){
  mod <- lm(FE ~ EngDispl + NumCyl, data = analysis(split))
  val <- assessment(split)
  preds <- predict(mod, val)
  mse <- mean( (val$FE - preds)^2)
  mse
}

folds_cars$mod2_results <- map_dbl(folds_cars$splits, cv_model2)

CV_MSE_mod2 <- mean(folds_cars$mod2_results)
```

```
data.frame(model = c("1", "2"), cv_mse = c(CV_MSE_mod1, CV_MSE_mod2))
```

```
##   model   cv_mse
## 1     1 21.44501
## 2     2 21.26763
```

- It looks like after performing 10-fold CV, model 2 is better afterall!