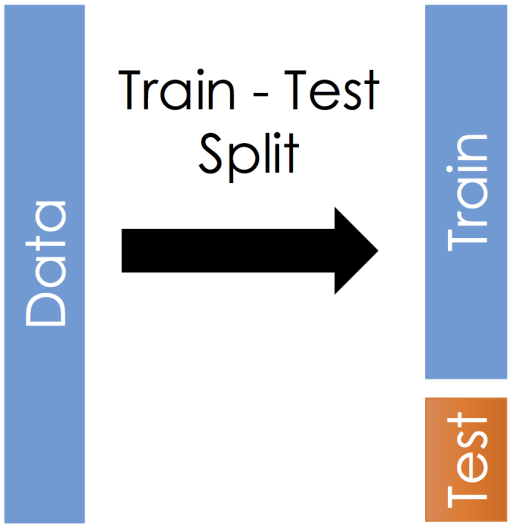
Model Selection and Tuning (Part 1: Cross-Validation)

Note 1: Images throughout this document illustrating train/validation/test sets are adapted from an image used at http://www.ds100.org

Note 2: The use of the terms as I'm defining them today isn't completely consistent. I'm describing the most common usage.

(Previously) Train-Test Split:

To get accurate measures of model performance, hold out a test set:



1. Training set:
   - Used for model estimation
2. Test set:
   - Used to evaluate model performance

Validation Split:

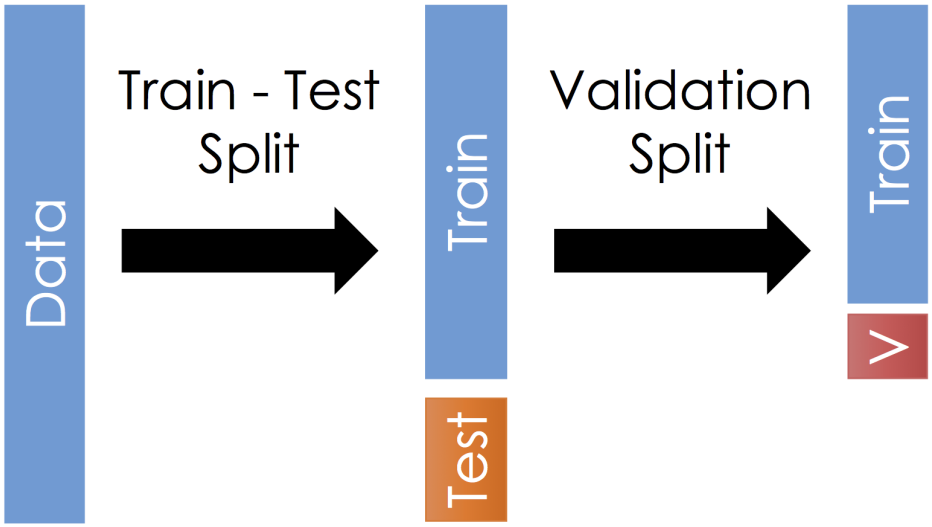
The model estimation process often involves two distinct parts:

1. Model Training
   - For example, parameter estimation via ordinary least squares for a linear regression model
2. Model and "Tuning Parameter" Selection
   - Which explanatory variables should be used?
   - What polynomial degrees and interactions should be included?

Both of these steps go into determining our final model. If we use test data in either step, performance on the test set will not be a reliable indicator of model performance!

To compare and select models, we need a way to estimate test set error **without looking at the test set data**.

Idea: introduce a third split of the data: **validation set**



Now, as part of model selection and estimation, we use:

1. Training set
   - actually used for estimating model parameters
2. Validation set
   - for comparing a set of candidate models (e.g. different sets of explanatory variables, different polynomial degree) and picking one (or a small number) to use

Only after selecting one (or a small number of) final model(s) do we use the test set data:

3. Test set
   - for final evaluation of performance of a small number of models

Typically, the selected models from step 2 will be re-fit to the combined training/validation data before looking at test set performance.

Once you have looked at performance on the test set, you are done! You can't go back! If you make any more changes to the model, any future evaluations of performance on the test set are invalid! (Or you need an entirely new test set.)

R Commands for getting a train/test or validation split.

Two basic steps to getting a data split:

1. Choose some indices to use in each part of the split.
   - We could use `sample` for this.
   - `caret::createDataPartition` is another choice: does stratified sampling to ensure test and train sets have similar sets of values for the response. But this can result in partitions that are not exactly the sizes you asked for.
2. Subset the data to the selected indices.
   - Could use square bracket notation.
   - I'll suggest using `dplyr::slice` since it can be more easily inserted into a sequence of steps with the pipe (`%>%`).

Example: Polynomial Regression with the cars data

```
# Read in data and take a look
cars <- read_csv("http://www.evanlray.com/data/sdm4/Cars.csv")
head(cars)
```

```
## # A tibble: 6 x 10
##   Country Car     MPG Weight `Drive Ratio` Horsepower Displacement
##   <chr>   <chr> <dbl>  <dbl>         <dbl>      <dbl>        <dbl>
## 1 U.S.    Buic~  16.9   4.36          2.73        155          350
## 2 U.S.    Ford~  15.5   4.05          2.26        142          351
## 3 U.S.    Chev~  19.2   3.60          2.56        125          267
## 4 U.S.    Chry~  18.5   3.94          2.45        150          360
## 5 U.S.    Chev~  30     2.15          3.7          68           98
## 6 Japan   Toyo~  27.5   2.56          3.05         95          134
## # ... with 3 more variables: Cylinders <dbl>, MPG_1 <dbl>, Weight_1 <dbl>
```

Recall we are treating `Weight` as the explanatory (X) variable and `MPG` as the response (Y).

Step 1: Split the data into training, validation, and test sets

```
library(caret)

# Set seed for reproducibility
set.seed(7304) # generated at random.org

# Divide into "estimation" (will be used for all parts of estimation) and test sets
# "Estimation set" is not official terminology, but I needed something to call this...
train_val_inds <- caret::createDataPartition(
  y = cars$MPG, # response variable as a vector
  p = 0.8 # approximate proportion of data used for training and validation
)
# any observations not listed below will be part of the test set.
train_val_inds
```

```
## $Resample1
##  [1]  1  4  5  6  7  8  9 10 11 12 13 14 15 16 18 19 20 22 23 24 25 26 27
## [24] 28 29 30 31 33 34 35 36 37
```

```
cars_train_val <- cars %>% slice(train_val_inds[[1]])
cars_test <- cars %>% slice(-train_val_inds[[1]])

# Further divide the "estimation" set into a training part and a validation part.
train_inds <- caret::createDataPartition(
  y = cars_train_val$MPG, # response variable as a vector -- note, splitting up cars_train_val
  p = 0.8 # approximate proportion of estimation-phase data used for training
)

cars_train <- cars_train_val %>% slice(train_inds[[1]])
cars_val <- cars_train_val %>% slice(-train_inds[[1]])
```

Here's what we've achieved so far:

```
nrow(cars)
```

```
## [1] 38
```

```
nrow(cars_train)
```

```
## [1] 28
```

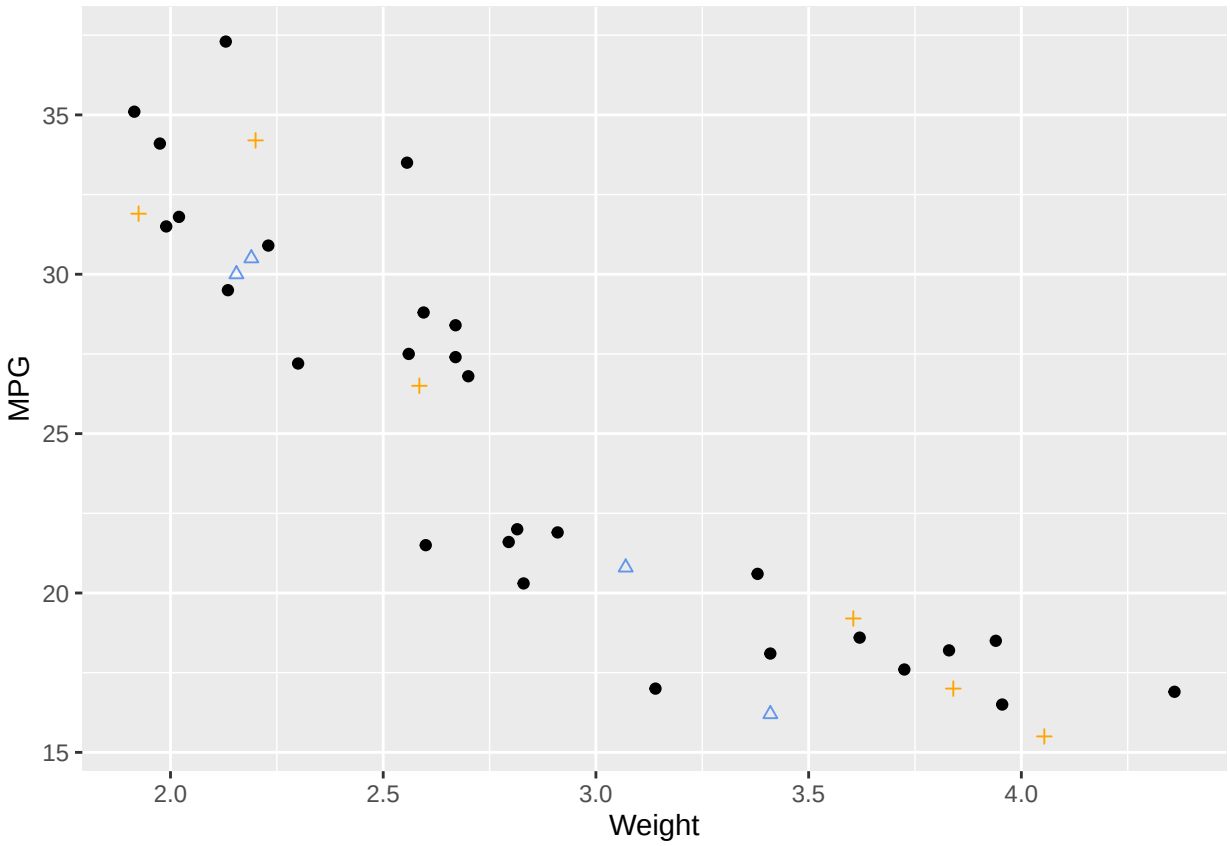
```
nrow(cars_val)
```

```
## [1] 4
```

```
nrow(cars_test)
```

```
## [1] 6
```

```
ggplot() +
  geom_point(data = cars_train, mapping = aes(x = Weight, y = MPG)) +
  geom_point(data = cars_val, mapping = aes(x = Weight, y = MPG), color = "cornflowerblue", shape = 2) +
  geom_point(data = cars_test, mapping = aes(x = Weight, y = MPG), color = "orange", shape = 3)
```



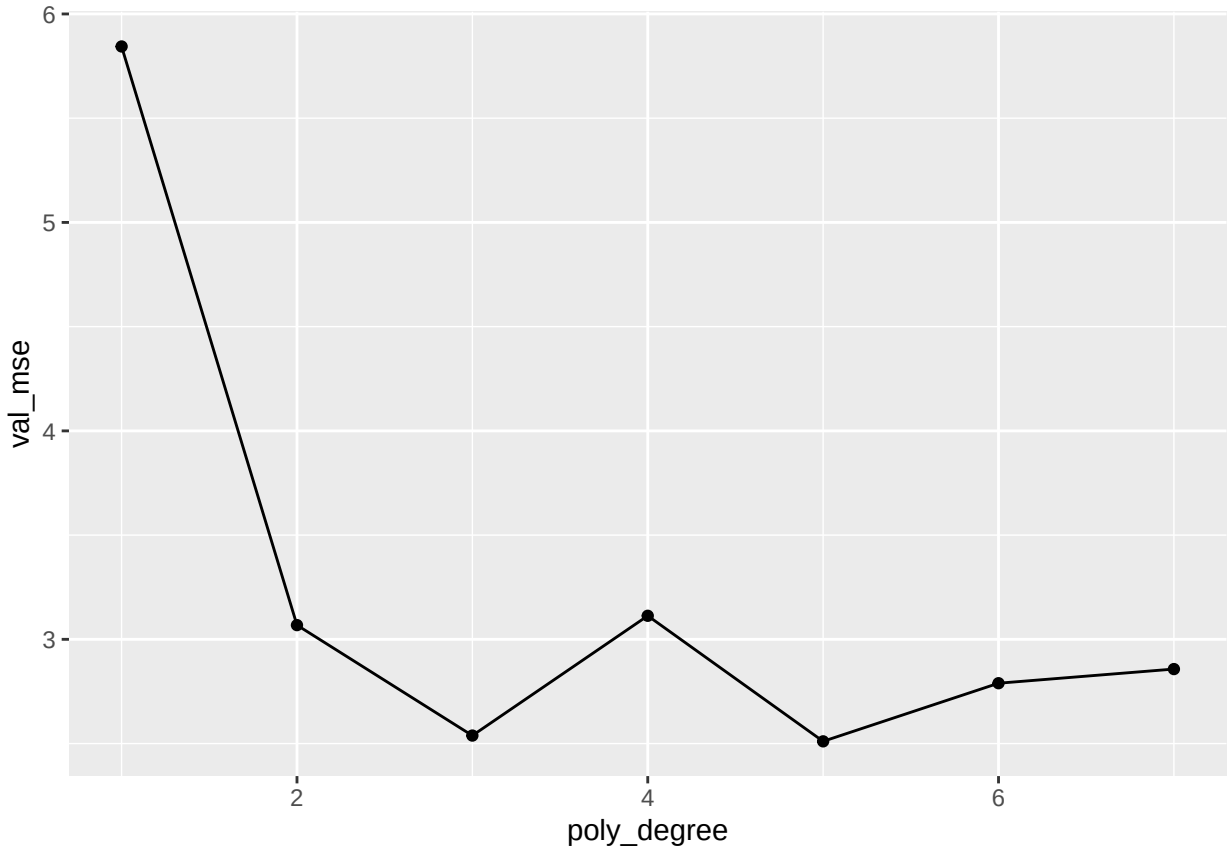
Step 2: Fit all candidate models to the training data and compare performance on validation data. Pick a small number of models for which to look at test set performance.

Here, we get validation set MSE for each candidate model:

```
train_val_mse <- data.frame(
  poly_degree = seq_len(7),
  train_mse = NA,
  val_mse = NA
)

for(degree in seq_len(7)) {
  fit <- lm(MPG ~ poly(Weight, degree), data = cars_train) # note fit to train data!

  train_resids <- cars_train$MPG - predict(fit) # by default, predictions are for training set
  train_val_mse$train_mse[degree] <- mean(train_resids^2)

  val_resids <- cars_val$MPG - predict(fit, cars_val) # here, get predictiosn for validation set
  train_val_mse$val_mse[degree] <- mean(val_resids^2)
}
```

Here's a plot of the results:

```
# Make a plot of the results!
ggplot(data = train_val_mse, mapping = aes(x = poly_degree, y = val_mse)) +
  geom_line() +
  geom_point()
```



According to the validation set, our best models have degree 3 and 5. Only now, evaluate on test set data!

Step 3: Find test set performance for chosen model(s)

```
fit <- lm(MPG ~ poly(Weight, 3), data = cars_train_val) # note fit to combined train_val data!
test_resids <- cars_test$MPG - predict(fit, newdata = cars_test) # based on predictions for test set
mean(test_resids^2)
```

```
## [1] 3.524412
```

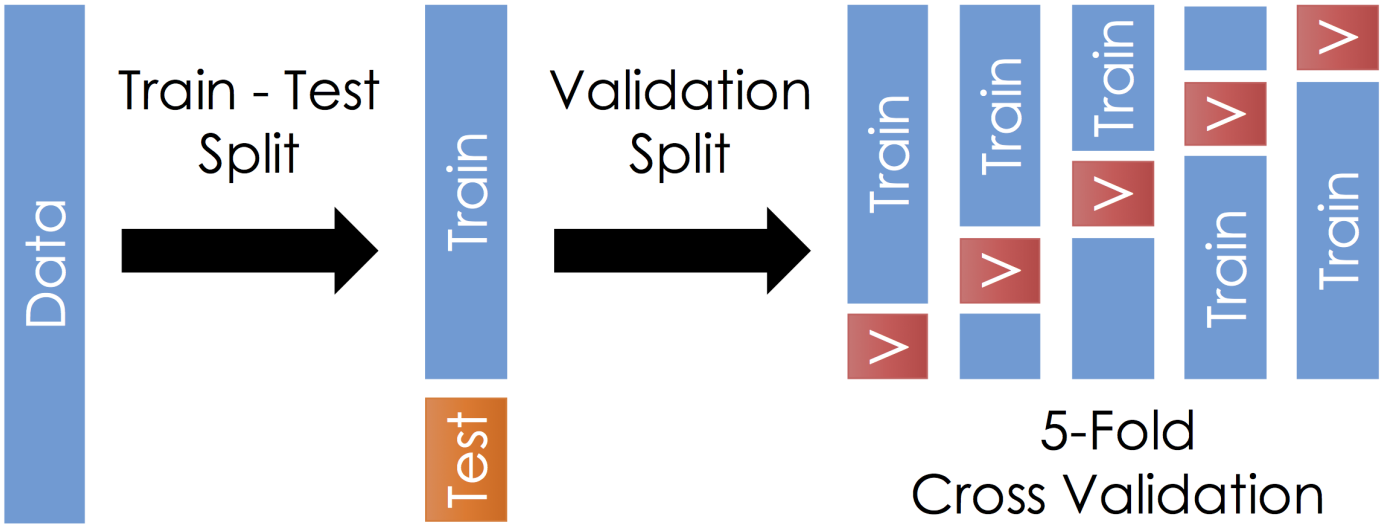
```
fit <- lm(MPG ~ poly(Weight, 5), data = cars_train_val) # note fit to combined train_val data!
test_resids <- cars_test$MPG - predict(fit, newdata = cars_test) # based on predictions for test set
mean(test_resids^2)
```

```
## [1] 4.460012
```

Cross-Validation

A limitation of the validation approach is that it doesn't use all of the available "estimation data" for either training or validation. Above, we made a decision about what model to choose based on performance for a validation set of only 4 observations!

Enter k-fold cross-validation:



- Partition the "estimation data" into k **folds** (groups of approximately equal size; above, $k = 5$)
- For each fold, get an estimate of model performance using that fold as a validation set and the remaining folds put together as a training set
- Overall cross-validated performance estimate is average validation MSE across the k folds

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

Example with cars data

Step 1: Split into training and test sets, obtain validation folds

Note that unlike above, I'm not actually splitting the data according to the validation folds yet.

```
# Set seed for reproducibility
set.seed(7304) # generated at random.org

# Divide into "estimation" (will be used for all parts of estimation) and test sets
# "Estimation set" is not official terminology, but I needed something to call this...
train_val_inds <- caret::createDataPartition(
  y = cars$MPG, # response variable as a vector
  p = 0.8 # approximate proportion of data used for training and validation
)
# any observations not listed below will be part of the test set.
train_val_inds
```

```
## $Resample1
##  [1]  1  4  5  6  7  8  9 10 11 12 13 14 15 16 18 19 20 22 23 24 25 26 27
## [24] 28 29 30 31 33 34 35 36 37
```

```
cars_train_val <- cars %>% slice(train_val_inds[[1]])
cars_test <- cars %>% slice(-train_val_inds[[1]])

# Generate partition of the "estimation" set into 5 folds
# the result is a list of length 5 with indices of obsevations to include in each fold.
num_crossval_folds <- 5
crossval_fold_inds <- caret::createFolds(
  y = cars_train_val$MPG, # response variable as a vector
  k = num_crossval_folds # number of folds for cross-validation
)
```

Step 2: Get performance for each validation fold, using the other folds put together as a training set.

```
train_val_mse <- expand.grid(
  poly_degree = seq_len(7),
  val_fold_num = seq_len(num_crossval_folds),
  train_mse = NA,
  val_mse = NA
)

for(poly_degree in seq_len(7)) {
  for(val_fold_num in seq_len(num_crossval_folds)) {
    # When I'm ready to save results, where should they go?
    results_index <- which(
      train_val_mse$poly_degree == poly_degree &
      train_val_mse$val_fold_num == val_fold_num
    )

    # Assemble training and validation sets as specified by the fold number we're looking at.
    cars_train <- cars_train_val %>% slice(-crossval_fold_inds[[val_fold_num]])
    cars_val <- cars_train_val %>% slice(crossval_fold_inds[[val_fold_num]])

    # Fit to training data
    fit <- lm(MPG ~ poly(Weight, poly_degree), data = cars_train) # note fit to train data!

    # Get training and validation set MSE
    train_resids <- cars_train$MPG - predict(fit) # by default, predictions are for training set
    train_val_mse$train_mse[results_index] <- mean(train_resids^2)

    val_resids <- cars_val$MPG - predict(fit, cars_val) # here, get predictiosn for validation set
    train_val_mse$val_mse[results_index] <- mean(val_resids^2)
  }
}
head(train_val_mse)
```

```
##   poly_degree val_fold_num train_mse  val_mse
## 1           1            1  6.297705 17.43264
## 2           2            1  4.794735 12.30843
## 3           3            1  4.394328 13.18885
## 4           4            1  4.120076 12.39092
## 5           5            1  3.755245 13.58034
## 6           6            1  2.399640 17.16519
```

```
summarized_crossval_mse_results <- train_val_mse %>%
  group_by(poly_degree) %>%
  summarize(
    crossval_mse = mean(val_mse)
  )
summarized_crossval_mse_results
```

```
## # A tibble: 7 x 2
##   poly_degree crossval_mse
##         <int>        <dbl>
## 1           1         9.21
## 2           2         7.02
## 3           3         9.74
## 4           4         7.21
## 5           5        49.9
## 6           6        11.7
## 7           7       170.
```

These results suggest that polynomials of degree 2 and 4, as well as possibly 1 and 3, have similar performance.

Step 3: Find test set performance for chosen model(s)

```
fit <- lm(MPG ~ poly(Weight, 2), data = cars_train_val) # note fit to combined train_val data!
test_resids <- cars_test$MPG - predict(fit, newdata = cars_test) # based on predictions for test set
mean(test_resids^2)
```

```
## [1] 4.036541
```

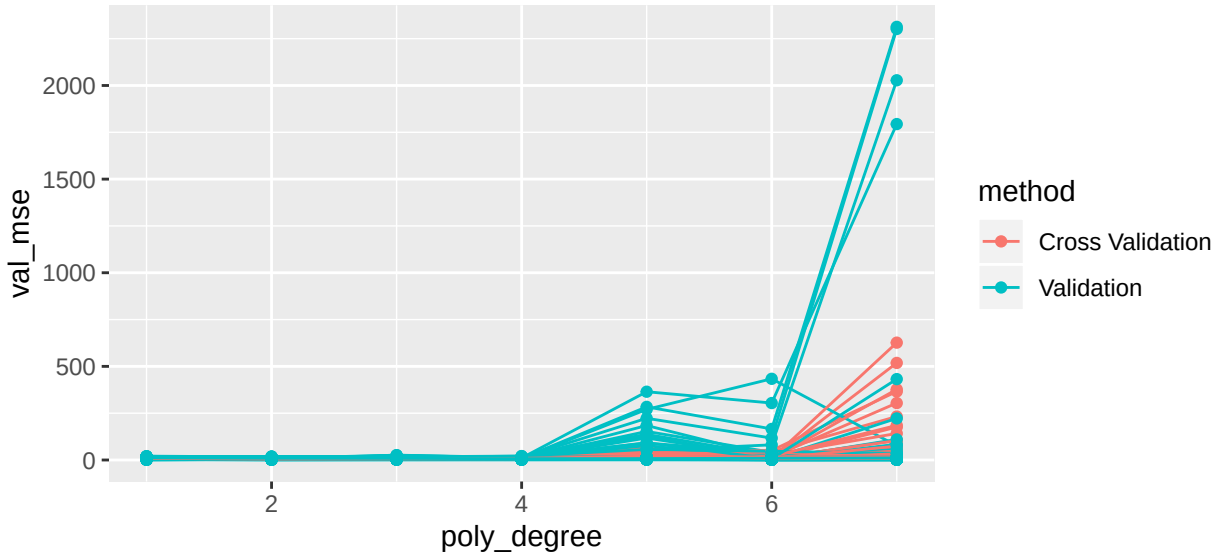
```
fit <- lm(MPG ~ poly(Weight, 4), data = cars_train_val) # note fit to combined train_val data!
test_resids <- cars_test$MPG - predict(fit, newdata = cars_test) # based on predictions for test set
mean(test_resids^2)
```

```
## [1] 2.681682
```

Cross-Validation Gives More Consistent Estimates of Test Set Performance than Just Validation.

Code suppressed, but I did everything above 100 times for different randomly selected partitions of the data into training and validation sets. In the following plot, each line shows results from either

- one split of the data into training and validation sets with an 80/20 split (for validation); or
- one split of the data into 5 cross-validation folds (for cross-validation)



Here is the variance of MSE scores resulting from different train/validation splits, by polynomial degree and method.

```
##   poly_degree Cross Validation Validation
## 1           1             0.11      24.16
## 2           2             0.14      18.30
## 3           3             0.72      28.53
## 4           4             0.86      19.26
## 5           5            69.69    4000.95
## 6           6            55.11    3073.30
## 7           7         10228.40  173975.05
```

How many Cross-Validation folds to use?

Common choices:

- $k = 5$
- $k = 10$
- $k = n$ (also known as leave-one-out cross-validation)

Consider:

- Cross-validation is a procedure for estimating test-set error rate
- Large k means our training sets are more similar in size to our full data set
- Large k can be very computationally expensive.
- An intermediate value like $k = 10$ usually works well enough. $k = 10$ is probably the most common choice.