Please note this is a prepress version of an accepted manuscript to Advances in Methods and Practices in Psychological Sciences. It is not the copy of record and may not exactly replicate the final, authoritative version of the article. Please do not copy without authors’ permission. The final article will be available, upon publication, via its DOI.

Citation:
Song, Q. C., Tang, C., & Wee, S. (in press) Making sense of model generalizability: A tutorial on cross-validation in R and Shiny. Advances in Methods and Practices in Psychological Science.
Making Sense of Model Generalizability: A Tutorial on Cross-Validation in R and Shiny

Q. Chelsea Song
Purdue University

Chen Tang
University of Illinois at Urbana-Champaign

Serena Wee
University of Western Australia

Author Note

Q. Chelsea Song, Department of Psychological Sciences, Purdue University; Chen Tang, School of Labor and Employment Relations, University of Illinois at Urbana-Champaign; Serena Wee, School of Psychological Science, University of Western Australia.

Correspondence concerning this article should be addressed to Q. Chelsea Song, Department of Psychological Sciences, 703 Third Street, West Lafayette, IN, United States 47907, Contact: qcsong@purdue.edu

Accepted for publication at the Advances in Methods and Practices in Psychological Sciences
Abstract

Model generalizability describes how well the findings from a sample are applicable to other samples in the population. In this tutorial, we explain model generalizability through the statistical concept of model overfitting and its outcome (i.e., validity shrinkage in new samples), and provide a Shiny app to simulate and visualize how it is influenced by three factors: model complexity, sample size, and effect size. We then discuss cross-validation as an approach for evaluating model generalizability, and provide guidelines for implementing this approach. To help researchers understand how to apply cross-validation to their own research, we walk through an example, accompanied by step-by-step illustrations in R. This tutorial is expected to help readers develop the basic knowledge and skills to use cross-validation to evaluate model generalizability in their research and practice.

Keywords: model generalizability, cross-validation, validity shrinkage
Making Sense of Model Generalizability: A Tutorial on Cross-Validation in R and Shiny

A current concern in psychology revolves around our ability to replicate our findings (Open Science Collaboration, 2015). In part, such concerns over replicability reflect an underlying concern with model generalizability (Yarkoni & Westfall, 2017). Model generalizability describes the extent to which statistical models developed in one sample fit other samples from the same population. In general, statistical models tend to not generalize well to a new sample—this is because they capitalize on the unique characteristics of the sample data, and tend to produce overly optimistic results (i.e., effect sizes) that overstate the expected effect size in both the population and new samples (e.g., Wherry, 1931; Lord, 1950). Although model generalizability and the key method to assess it—cross-validation—have been discussed in the early psychometric literature (e.g., Wherry, 1931; Lord, 1950; Mosier, 1951; Rozeboom, 1981), they have been under-emphasized in contemporary psychological training and research (de Rooij, Pratiwi, Fokkema, Dusseldorp, & Kelderman, 2019). As we strive to conduct replicable research, these concepts will become increasingly important for psychological scientists. Thus, our goal in this tutorial is to (re-)introduce to our core training the concepts of model generalizability and cross-validation.

We begin by using a Shiny app to illustrate how statistical models tend to overfit sample data, leading to poor model generalizability. We demonstrate how model generalizability is affected by model complexity, sample size, and effect size. Next, we briefly describe the concept of cross-validation, review the major steps, and discuss two cross-validation methods researchers may use with their own data (k-fold cross-validation and Monte-Carlo cross-validation). We

---

1 Although statistical models commonly overfit the data, it is also possible for them to underfit the data, which could also lead to poor model generalizability. Please refer to “Observation 3: Model Generalizability is Influenced by (a) Model Complexity, (b) Sample Size, and (c) Effect Size” for further discussion.
demonstrate the methods by walking through an empirical example using the easy-to-use and powerful R package, caret (Kuhn, 2008).

A Demonstration of Model Overfit in a Shiny App

Suppose we want to model the relationship between a person’s level of arousal and their performance on a learning task (e.g., how arousal relates to the number of new words memorized). We begin by obtaining a random sample from the population, and then use the arousal and task performance measured in this sample to fit a statistical model. For example, we might fit a regression model to the sample dataset and obtain the regression coefficients that describe the relationship between arousal and task performance. This sample is called the calibration sample, as the process of estimating the regression coefficients is analogous to “calibrating” the model.

Let us visualize the process using an interactive Shiny app (https://qchelseason.shinyapps.io/CrossValTutorial/). We can interact with the Shiny app using the grey control panel on the left-hand side. To draw a random sample of 50 observations from the population, move the “Calibration Sample Size” slider to 50. Let us assume that, in the population, there is a positive linear relationship between arousal and task performance, and that arousal explains 25% of the variation in task performance (population effect size, $\rho^2 = .25$). To specify this in the app, move the “Population Effect Size ($\rho^2$)” slider to .25, and select “Linear” under “Data Generation Procedure”. Then, click the “Generate Calibration Sample” button to generate a random sample. You will obtain a sample of 50 observations, drawn from a population where the true relationship between arousal and task performance is $\rho^2 = .25$. A
scatter plot with 50 black dots shows up in the Shiny app (see Figure 1a). Next, let us fit a regression model to the sample dataset, and obtain the regression coefficients. When there is only one predictor, the fitted regression model has the following general form:

\[ \hat{y} = b_0 + b_1 x + b_2 x^2 + \ldots + b_k x^k \]

where \( \hat{y} \) is the predicted value of the outcome (e.g., task performance) when the predictor (e.g., arousal level), \( x \), takes on a particular value. The regression coefficients, \( b_0, b_1, b_2, \ldots, b_k \), represent the different forms of the relationship between arousal and task performance. For example, \( b_1 \) captures the linear relationship between arousal and task performance, \( b_2 \) captures the quadratic relationship between arousal and task performance, and so on.

In this analysis, we are trying to examine both the form and magnitude of the relationship between arousal and task performance. We start off by estimating a simple linear regression: \( \hat{y} = b_0 + b_1 x \). In this model, the degree of polynomial is 1 (i.e., \( \hat{y} = b_0 + b_1 x^1 \)):

In the Shiny app, use the radio button to set “Degree of Polynomial” to “1”. Then, click on the “Fit the model!” button to fit the model (using the calibration sample). The fitted regression line is shown as a black line in the Shiny app (see Figure 1b). If you select “Show residual errors”, you can see how well the model fits the calibration sample data. This leads us to our first observation:

**Observation 1: The Model Overfits the Calibration Sample**

As shown in Figure 1b, the black line shows the estimated (i.e., expected) task performance, at a given level of arousal. The black vertical dotted lines in Figure 1b represent

---

2 To ensure that you can recreate the same output as shown in Figure 1a, the Shiny app is configured to generate the same initial sample when you start the application. If you already clicked the “Generate Calibration Sample” button multiple times before following the current instruction, your output could look different from Figure 1a. If this is a concern, simply refresh the Shiny app site and closely follow the instructions from the beginning of this section.

3 For example, according to Figure 1b, if a person has an average level of arousal (i.e., mean-centered arousal = 0), our best estimate of that person’s performance is that they memorized 4 words in the learning task.
the residual errors, and indicate the extent to which the expected task performance (i.e., points along the black line) deviates from the observed task performance (i.e., the black dots). In regression, the sum of squared residual errors were minimized to provide the best possible fit to the calibration sample data.

We will use two key metrics to examine how well the model fits the data: $R^2$ and mean squared error ($MSE$). $R^2$ is typically interpreted as the proportion of the variance in the outcome variable (e.g., task performance) that can be accounted for by the predictors (e.g., arousal level); $MSE$ represents the magnitude of the average squared residual, and indicates how much, on average, expected values deviate from observed values. $R^2$ (when calculated by squaring correlation coefficients) captures the extent to which expected values exhibit the same rank-order as observed values, providing a relative measure of model fit; $MSE$ captures the magnitude of the average squared residual, providing an absolute measure of model fit. As $R^2$ and $MSE$ focuses on different aspects of model fit, we recommend reporting both metrics or reporting the metric most relevant to the relationship examined.

Now, in the Shiny app, check the “Show $R$-squared” and “Show MSE” boxes. These values are shown at the top of the scatter plot in Figure 1c: $R^2_{calibration} = .45$ means that, in the calibration sample, the relationship between arousal and task performance (i.e., the fitted model) accounts for 45% of the variance in task performance; $MSE_{calibration} = .48$ means that the model’s predicted task performance will differ from the observed task performance, on average, by a little more than half of a word (i.e., $\sqrt{.48} = .69$ words) per observation.$^4$ The calibration sample effect size ($R^2_{calibration} = .45$) is almost twice as large as the population effect size.

---

$^4$ The square root of $MSE$ (i.e., the root mean squared error, $RMSE$) expresses the model (in)accuracy in the original units of the outcome variable (in this case, number of words memorized).
(\(\rho^2 = .25\)), suggesting that the model overfitted the sample data. Overfit occurred because the model captured variation unique to the calibration sample and unrepresentative of the relationship in the population.

**Observation 2: The Model Obtained from the Calibration Sample Tends to Not Generalize Well to New (Validation) Samples**

In the previous example, we used a statistical model to examine the relationship between arousal and task performance in the calibration sample. Often, we also want to know whether the findings could be replicated in other samples from the same population. That is, we are interested in whether the model generalizes to a new sample (i.e., the validation sample). In the Shiny app, click “Test in a new sample!”: a validation sample of size 1,000 will be randomly drawn from the population, and used to evaluate the calibrated regression model. The updated result is shown in Figure 1d. Specifically, we took the regression model (i.e., the black line)—obtained based on the calibration sample (i.e., the black dots)—and examined how well it predicted task performance from arousal in the validation sample (i.e., the grey dots). That is, we examined the prediction accuracy of the model (as captured by \(R^2_{validation}\) and \(MSE_{validation}\)), which reflects how well the calibrated model is likely to perform in new samples. As shown in Figure 1d, the black line models the relationship between arousal and task performance more poorly in the validation sample than in the calibration sample: Predictions deviated more from the observed values in the validation sample (by \(\sqrt{MSE_{validation}} = \sqrt{.78} = .88\) words) than in the calibration sample (by \(\sqrt{MSE_{calibration}} = \sqrt{.48} = .69\) words).

In addition, as seen in Figure 1d, the model explained less variation in the outcome in the validation sample than in the calibration sample. Whereas \(R^2_{calibration}\) reflects how well the
model fitted the calibration sample data (including its unique characteristic), $R^2_{validation}$ reflects the accuracy with which the calibrated model predicts the outcome variable of the observations not used in fitting the model. This decrease is called validity shrinkage, and reflects the degree to which the model performs less well in the validation sample: The model explained 26% $(R^2_{validation} = .26)$ of the variance in the validation sample, as compared to 45% $(R^2_{calibration} = .45)$ of the variance in the calibration sample.

**Observation 3: Model Generalizability is Influenced by (a) Model Complexity, (b) Sample Size, and (c) Effect Size**

**Observation 3a: The model generalizes less well when it is complex.** Earlier, we modeled the simple linear relationship between arousal and task performance: $\hat{y} = b_0 + b_1 x$. However, other (nonlinear) forms of the relationship might be plausible—in reality, we usually do not know the true form of the relationship between a predictor and an outcome. To examine if a nonlinear relationship between arousal and task performance is plausible, we could fit more complex models to the data and then evaluate their fit. For example, we could fit a quadratic regression model $\hat{y} = b_0 + b_1 x + b_2 x^2$, a cubic regression model $\hat{y} = b_0 + b_1 x + b_2 x^2 + b_3 x^3$, and so forth. Each additional polynomial term increases the model’s complexity, and by doing so, allows the model to fit the calibration sample data more closely.

To illustrate this in the Shiny app, increase “Degree of Polynomial” from 1 to 2 to 3, each time clicking on the “Fit the model!” button to see how well each model fits the calibration sample. As model complexity increases, the regression line fits the data more closely: the cubic regression line fits the calibration sample better than the linear regression line. To examine how well the model performs in the validation sample, keep the “Show R-squared” and “Show MSE” boxes checked, and sequentially increase model complexity (i.e., degree of polynomial)
from 1 to 2 to 3, each time clicking on the “Test in a new sample!” button to see the validation sample results. As the complexity of the model increases, $R^2_{validation}$ decreases and $MSE_{validation}$ increases. In general, as the model becomes more complex, it generalizes less well to other samples in the population.\(^5\)

**Observation 3b: The model generalizes less well when calibration sample size is small.** Aside from the model itself, the calibration sample size is also an important factor. Try varying the calibration sample size: In the Shiny app, set the “Calibration Sample Size” to 30, the “Population Effect Size ($\rho^2$)” to .25, the “Degree of Polynomial” to 1. Then, check the boxes for $R^2$ and $MSE$, and click the “Test in a new sample!” button to see the validation sample results. Repeat this procedure, each time increasing the calibration sample size from 50 to 100, then to 200. In general, as the calibration sample size increases, the difference in terms of $R^2$ and $MSE$ between the calibration and validation sample decreases (i.e., validity shrinkage decreases). The model generalizes better when the calibration sample size is large, because large samples tend to be more representative of the population.

**Observation 3c: The model generalizes less well when the population effect size is small.** Now, we will illustrate what happens if we vary the population effect size. In the Shiny app, set the “Calibration Sample Size” to 50, the “Population Effect Size ($\rho^2$)” to .01, and the “Degree of Polynomial” to 1. Then, check the boxes for $R^2$ and $MSE$, and click “Test in a new sample!”. Repeat this procedure, each time increasing the population effect size from $\rho^2 = .01$ to $\rho^2 = .81$. In general, as the population effect size increases, the difference between $R^2$ in the

---

\(^5\) Due to sampling variation, this general trend might not be observed on every trial. To further illustrate the general trend, we simulated the process in Observation 3a (trials = 1,000). The R code and results of the simulation are shown in Appendix A.
calibration and validation samples decreases, as does the difference in $MSE$. That is, as the population effect size becomes larger, the model generalizes better to new samples.

Together, Observations 3a to 3c show that: model generalizability decreases as (a) model complexity increases, (b) calibration sample size decreases, and (c) population effect size decreases. In the above example, the population relationship was assumed to be linear. However, as suggested by many studies, the population relationship between arousal and task performance is most likely to be quadratic (e.g., Hebb, 1955). To see what happens if the form of the relationship in the population is quadratic, in the Shiny app, select the “quadratic” radio button under “Underlying Relationship in the Population”. Using the “Degree of Polynomial” radio buttons, systematically change the degree of the polynomial of the regression model from “1” to “3” and observe how the $R^2$ and $MSE$ change. In general, we observe patterns consistent with Observations 2, and Observations 3a, 3b, and 3c described above.

It is important to note that, although statistical models commonly overfit the data, it is also possible for them to underfit the data. Model underfit could be caused by sampling variation, as well as model complexity. For example, if the underlying population relationship is quadratic, then fitting a linear regression model to the observed data will likely result in model underfit, as the regression model is too simplistic a representation of the underlying relationship in the population. Underfitted models tend to have low model fit (i.e., low $R^2$ and high $MSE$) in both the calibration and validation sample. Because of this, along with model overfit, model underfit could also lead to poor model generalizability. While a thorough discussion of underfitting is beyond the scope of this tutorial, interested readers could refer to Hastie, Tibshirani, and Friedman (2009).

Statistical Models in Psychological Research
Most psychologists are interested in the population-level relationship between the predictor(s) and the outcome. For instance, in our example above, we estimated the relationship between arousal and task performance using a regression model. As we observed, the regression model overfitted the calibration sample, such that the model fit shrunk in the validation sample. Further, we also observed that the extent to which a model generalized to other samples in the population depended on three factors: model complexity, sample size, and population effect size.

These factors are particularly relevant when considering model generalizability in psychological research. At least until very recently, many psychological studies were based on small sample sizes (Shen, Kiger, Davies, Rasch, Simon, & Ones, 2011). For example, in a recent replication attempt examining 28 classic social psychological studies (Many Labs 2 project; Klein et al., 2018), the median sample size of the original studies was 86.5 (calculated from the raw data: https://osf.io/crz2n/). As we observed earlier (Observation 3b), models fitted on small calibration samples will likely be overfitted. In addition, due to the complexity in human perception and behavior, the phenomena that psychological studies examine also tend to have small effect sizes. For example, in the Many Labs 2 project (Klein et al., 2018), the median Cohen’s $d$s obtained in the replication studies was .15. In fact, a summary of the effect sizes reported in social psychology showed a median effect size of $r = .25$ (Lovakov & Agadullina, 2017; $k = 98$ reporting 13,464 associations [i.e., Pearson $r$ and Hedge’s $g$]); and a summary of the effect sizes reported in industrial-organizational psychology showed a mean effect size of about $r = .22$ (e.g., Bosco, Aguinis, Singh, Field, & Pierce, 2015, based on 147,328 correlations; Paterson, Harms, Steel, & Crede, 2016; $k = 250$). Finally, interaction effects, curvilinear effects, as well as control variables are often included in the models, increasing model complexity.
In order to minimize model overfit and increase model generalizability (decrease validity shrinkage), we need: (a) large samples, (b) not-small effect sizes, and (c) models that are not unnecessarily complex. However, this trifecta is rare in psychological research: Increasing the sample size is often associated with a higher cost; the size of a given effect is not subject to researcher discretion; and the complexity of the model is often guided by theory. Thus, psychological studies are often prone to model overfit, suggesting there is a need for approaches that could provide additional information on how well statistical models are expected to generalize to new samples—cross-validation is such an approach.

**Cross-Validation: An Approach to Assess Model Generalizability**

As demonstrated earlier, one way to evaluate model generalizability is to assess the model on a validation sample. However, obtaining a new sample can be challenging or impractical (e.g., due to limited resources). Cross-validation is an alternative approach to evaluate model generalizability with the sample that we already have in hand (e.g., Hastie et al., 2009).

We could, for example, evenly split the sample data into two sets, then fit (or train) a model in the first set (*training set*), and evaluate (or test) the generalizability of the model in the second set (*test set*). If the model fit is similar between the training and test sets, it provides initial evidence that the model will generalize well to new samples. However, there are caveats to this procedure: Prediction accuracy is still only estimated once, and the issue of sampling error is exacerbated by using only half of the sample for fitting and testing the model. Thus, most cross-validation approaches *repeat* this train-then-test cycle in different splits of the data. Put

---

6 Cross-validation provides an estimate of how likely a model is to replicate in a new sample from the same population. If readers are interested in generalizing to a different population, an alternative method—validity generalization—should be used (Mosier, 1951).
simply, the essence of cross-validation is to generate training and test sets from the data from a single sample, so as to repeatedly train and test the model. Although specific cross-validation methods exist that differ in terms of (a) how the data is split, and (b) how many repetitions of the train-then-test cycles are conducted, these methods share the same underlying process.

As seen in Table 1, there are five steps in cross-validation: 1) obtain training and test sets; 2) fit a model in the training set; 3) apply the fitted model to the test set and obtain prediction accuracy from the test set; 4) repeat Steps 1 to 3; and 5) calculate average cross-validated prediction accuracy across all the repetitions. The outcome of the procedure—the average cross-validated prediction accuracy—provides an estimate of how well the model will generalize to new samples. As compared to a single train-then-test cycle, the repeated train-then-test cycle results in a more stable cross-validated prediction accuracy estimate, which is less susceptible to random sampling variation.

In the following sections, we describe two common cross-validation methods (i.e., $k$-fold cross-validation and Monte-Carlo cross-validation), as well as some variations (e.g., repeated $k$-fold cross-validation, stratified $k$-fold cross-validation). The $k$-fold and Monte-Carlo cross-validation methods differ in the procedures used to generate the training and test sets; Table 2 provides a comparison of these methods.

**$k$-fold cross-validation ($k$-fold CV; Geisser, 1975).** In $k$-fold CV, the dataset is first randomly split into $k$ equal-sized subsets. Then, the train-then-test procedure is repeated $k$ times: Each time, one of the $k$ subsets is used as a test set and the rest are used to form the training set. To visualize $k$-fold CV for a regression model, use the Shiny app with the following inputs (see Figure 2): “Calibration Sample Size” is 50; “Population Effect Size” is .25; “Underlying Relationship in the Population” is linear; “Degree of Polynomial” is 3 (i.e., cubic regression).
Then, click on the “5-Fold Cross-Validation” button and watch how each step of the 5-fold cross-validation unfolds!

Figure 2 displays the results of 5-fold CV. Notice that, in addition to the original scatter plot, there are now five additional plots. Each new plot summarizes the results from one repetition of the 5-fold CV. For example, in “Fold 1”, 4/5ths of the original 50 observations (red dots) were used as the training set, with the cubic regression model (red line) fitted to these dots. Then, this model (red line) is evaluated in the test set (blue dots; i.e., the remaining 1/5th of the observations) to obtain an estimate of the cross-validated prediction accuracy ($R^2_{CV}$ and $MSE_{CV}$). The same train-and-test procedure is carried out $k = 5$ times, each time with a 1/5th of the data as the test set and the rest as the training set. The way training sets (red dots) and test sets (blue dots) are partitioned in each fold is also represented visually at the top of the plots, with the red-and-blue bar.

The overall cross-validated prediction accuracy ($R^2_{CV, Avg}$ and $MSE_{CV, Avg}$) is calculated by taking the average across the $k = 5$ folds. This is shown at the top of the Shiny app display (Figure 2, $R^2_{CV, Avg} = .32$ and $MSE_{CV, Avg} = .66$). It suggests that if we obtained a new sample from the population, the model fit is likely to be less than .33 ($R^2_{calibration} = .33$) and closer to .32 and $MSE$ is likely to be larger than .59 ($MSE_{calibration} = .59$) and closer to .66.

Monte-Carlo cross-validation (MCCV; Picard & Cook, 1984). The MCCV procedure follows a similar train-then-test procedure as $k$-fold CV. The key distinction is that, in MCCV, a predefined proportion of the dataset is randomly selected to form the test set in each iteration, with the remaining proportion forming the training set. For example, if the predefined proportion is 20:80, then 20% of the observations will be randomly selected to form the test set, and 80% to form the training set. A model is then fitted to the training set and evaluated on the test set. This
random data draw, together with the train-then-test procedure, is repeated a predetermined number of times (e.g., \( n = 100 \) repetitions). As with \( k \)-fold CV, overall cross-validated prediction accuracy \( (R_{CV, Avg}^2 \) and \( MSE_{CV, Avg} \)) is calculated by averaging across the \( n \) repetitions.

To examine the MCCV procedure in the Shiny app, simply click the “Monte-Carlo Cross-Validation” button at the bottom of the left-hand bar. Similar to the \( k \)-fold CV demonstration, additional plots are added to the original calibration sample result, with each representing a different repetition. However, different from the \( k \)-fold CV demonstration, in the red-and-blue bar at the top of each plot (which demonstrates how training and test sets are partitioned in each repetition), the red lines (representing the training set) are randomly scattered. This is because, in MCCV, the training sets are randomly selected in each repetition, whereas in \( k \)-fold CV, it was selected sequentially. For a demonstration on how to implement \( k \)-fold CV and MCCV in R, please refer to Appendix B.

**Other cross-validation methods.** Over the years, specific extensions to \( k \)-fold CV and MCCV have been developed. We briefly mention some of the most common extensions and provide key citations for interested readers to delve deeper into these specific methods. Leave-one-out cross-validation (LOOCV; Geisser, 1975; Stone, 1974) is a special instance of \( k \)-fold CV, where \( k = n \); this method might be useful when the sample sizes are very small. Repeated \( k \)-fold CV (Molinaro, Simon, & Pfeiffer, 2005; Kim, 2009) extends \( k \)-fold CV by conducting multiple repetitions, where each repetition uses a different \( k \)-fold split; this method can provide a more stable estimate of prediction accuracy, as compared with simple \( k \)-fold CV.

Both LOOCV and repeated \( k \)-fold CV are appropriate for datasets with independent observations. However, many psychological studies use datasets with nested structures that create dependencies in the data—examples include multilevel studies (e.g., students within
schools) and within-subject studies (e.g., repeated measures or longitudinal designs where the same person provides multiple data points). Extensions of the \( k \)-fold CV method have been developed specifically to deal with nested data. For example, if it is important to retain the data dependence, group \( k \)-fold CV should be used, which keeps groups intact when splitting the data (Kuhn, 2019); group \( k \)-fold CV is more suitable when there are many groups and group sizes are small. If it is important to maintain proportionate representation within a group (e.g., the proportion of women or minorities), then stratified \( k \)-fold CV (Kohavi, 1995) is recommended; stratified \( k \)-fold CV is more suitable when there are few groups and group size is large. We note that the above extensions of \( k \)-fold CV could also be applied to MCCV in a similar way. For additional information, see Roberts et al. (2017).

**Step-By-Step Illustrations in R**

In the previous section, we used a Shiny app to visualize the cross-validation procedure, with an aim to help readers develop an intuition about what was happening in each step of the cross-validation process. In this section, we demonstrate how to conduct cross-validation using the `caret` R package (Kuhn, 2008). The `caret` package is a powerful\(^7\) and easy-to-use toolbox that allows users to conduct cross-validation using just a few simple lines of code. For example, a typical modeling and cross-validation procedure only requires two functions: `train()` and `trainControl()`. To illustrate how to conduct cross-validation using `caret`, we walk through an example based on a publicly available dataset, and provide a set of annotated R code that can be easily downloaded and modified for your own use.

**Population and Calibration Sample**

\(^7\) At the time of this writing, the `caret` package provides functionality for fitting 238 distinct statistical models and implementing cross-validation using seven distinct methods.
Our example data set comes from 71,992 participants that completed the online version of the MACH-IV measure of Machiavellianism (Christie & Geis, 1970). The participants also completed the Ten Item Personality Inventory (TIPI; Gosling, Rentfrow, & Swann, 2003) measure of the Big Five personality traits and demographic questions. The original data is available from https://openpsychometrics.org/_rawdata/.

Suppose we are interested in predicting Machiavellianism using Big Five personality, age, and gender. To obtain the population effect size, we treated the 71,992 participants as the population of interest (see Appendix C for details): In the population, the predictor variables explained 28% of the variation in Machiavellianism scores ($R^2_{population} = .28$) and the mean squared distance between the observed Machiavellianism score and fitted score is .45 ($MSE_{population} = .45$). A calibration sample ($N = 300$) was randomly drawn from the population, and was then used to fit a regression model. $R^2$ in the calibration sample was larger than that in the population (i.e., $R^2_{calibration} = .30$ vs. $R^2_{population} = .28$), and $MSE$ in the calibration sample was smaller than that in the population (i.e., $MSE_{calibration} = .43$ vs. $MSE_{population} = .45$). This is because the regression model is capitalizing on chance variation within the calibration sample. Next, to evaluate model generalizability and obtain more realistic estimates of $R^2$ and $MSE$, we conducted $k$-fold CV and MCCV.

**$k$-Fold Cross-Validation**

A 10-fold CV was implemented on the sample dataset using the caret package (see Appendix C). This was done with a few lines of code:

```r
kfold_train_control <- trainControl(method="cv", number=10)
```

---

8 The example dataset ($N = 71,992$) was cleaned based on the original dataset ($N = 73,489$). It is available in a GitHub repository, and downloadable using code in Appendix C.
**CROSS-VALIDATION TUTORIAL**

```r
kfold_cv <- train(mach ~ age + as.factor(gender) + O + C + E + A + N,
                   data=sample_cal, method="lm", trControl=kfold_train_control)
```

Results of the 10-fold CV suggest the cross-validated $R^2$ values are smaller than the calibration sample $R^2$ values (i.e., $R^2_{CV.Avg} = .28$ vs. $R^2_{calibration} = .30$), and thus more closely approximate the population $R^2$ value of .28. Similarly, the cross-validated $MSE$ values are less optimistic (i.e., larger) than the calibration sample $MSE$ values (i.e., $MSE_{CV} = .46$ vs. $MSE_{calibration} = .43$). Here, non-repeated $k$-fold CV was used for demonstration purposes only; other more suitable cross-validation methods are available (e.g., repeated $k$-fold CV, MCCV), and will be discussed in the Discussion section.

**Monte-Carlo Cross-Validation (MCCV)**

Using the `caret` package, MCCV was also implemented on the sample dataset (see Appendix C). The only difference between the R code for MCCV and $k$-fold CV is a different specification in the `trainControl()` function:

```r
mc_train_control <- trainControl(method="LGOCV", p=.8, number=200)
```

The Monte-Carlo cross-validated $R^2$ values are smaller than the calibration sample $R^2$ values (i.e., $R^2_{MCCV} = .28$ vs. $R^2_{calibration} = .30$), and more closely approximate the population $R^2$ value of .28. Similarly, the $MSE$ values from the MCCV are less optimistic than those obtained in the calibration sample (i.e., $MSE_{MCCV} = .46$ vs. $MSE_{calibration} = .43$), and is once again closer to the $MSE$ in the population ($MSE_{population} = .45$).

**Writing Up the Results**

We could summarize the cross-validation results using the following paragraph:

“In order to evaluate the model generalizability of our predicted model, we used the `caret` package (version 6.0-86; Kuhn, 2008) in R (version 3.6.3; R Core Team, 2019) to
perform Monte-Carlo cross-validation (MCCV; using 200 repetitions and holding out 20% of the sample in each repetition). According to the cross-validation result, when generalized to another sample, the prediction accuracy of the regression model is $R^2 = .28$. That is, in a new sample, 28% of the variance in the Machiavellianism scores will likely be accounted for by personality, gender, and age. Additionally, the cross-validated $MSE = .46$ suggests that, on average, the model-predicted Machiavellianism score will likely deviate from the observed scores in the new sample by $\sqrt{.46} = .68$ points on a 5-point scale.”

**Discussion**

**Choosing among Cross-Validation Methods**

As others before us have noted (Arlot & Celisse, 2010; Hastie et al., 2009, Chapter 7; Kuhn & Johnson, 2013), developing clear guidelines for choosing among cross-validation methods is extremely difficult because the choice of specific methods depends on many factors. In practice, these factors include the bias/variance associated with the cross-validation estimates (e.g., $R^2_{CV.Avg}$ and $MSE_{CV.Avg}$), as well as the computational cost of a cross-validation method (see Arlot & Celisse, 2010, p. 68 - 69; James et al., 2013, p. 181 - 184; Kuhn & Johnson, 2013, p. 69 - 70). In the context of cross-validation, *bias* refers to the systematic difference between the population parameter (e.g., $\rho^2$) and the obtained cross-validation estimate (e.g., $R^2_{CV.Avg}$); and *variance* refers to the uncertainty (or expected change) in the cross-validation estimates when different data partitions/training sets are used (e.g., Kuhn & Johnson, 2013, p. 70). For example, if two implementations of simple 5-fold CV are conducted on a dataset, and the obtained cross-validation estimate (e.g., $R^2_{CV.Avg}$) differs substantially across implementations, this would indicate high variance in the cross-validation estimates. Computational cost (also referred to as computational complexity) refers to the computational time and the size of computer memory
required to implement the cross-validation method. It depends on computer specifications (e.g., processing power, RAM), as well as model specifications (e.g., model complexity, number of partitions/repetitions of the dataset, and sample size).

The choice of cross-validation methods is influenced by both bias/variance and computational cost. On the one hand, increasing the number of repetitions for a cross-validation method increases the stability of the estimates (i.e., decreases variance), without increasing bias (Molinaro, Simon, & Pfeiffer, 2005). Thus, repeated $k$-fold CV and MCCV are generally preferred over simple $k$-fold CV (Kim, 2009; see also Zhang & Yang, 2015; Kuhn & Johnson, 2013, p. 70). However, on the other hand, in practice, conducting many repetitions is computationally costly (especially when the statistical model is complex), limiting the choice of cross-validation methods.

The effects of bias/variance and computational cost are further influenced by sample size: when sample size is small, bias/variance is more likely a concern; when sample size is large, computational cost is more likely a concern. Thus, when sample size is small, one could choose repeated $k$-fold CV or MCCV over simple $k$-fold CV, as they yield cross-validated estimates that are less susceptible to high variance (Molinaro et al, 2005). When sample size is large and computational capacity is limited, one could choose simple $k$-fold CV over repeated $k$-fold CV and MCCV, as long as they are willing to accept less accurate cross-validation estimates (e.g., James et al., 2013).\footnote{In addition, when sample size is large, cross-validation estimates generally tend to be more stable and less biased, as each training and test sets are based on a larger number of observations (James et al., 2013, p. 183).}

To sum up, we generally suggest using repeated cross-validation methods (e.g., repeated $k$-fold CV, MCCV) over non-repeated methods (e.g., simple $k$-fold CV). However, when
computational cost becomes a limitation, and especially when sample size is very large, non-
repeated methods could be considered over repeated methods. In such cases, to examine whether
a non-repeated cross-validation method yields stable cross-validation estimates in a particular
study (which involves a specific sample size and model), we suggest readers to run a few
implementations of simple $k$-fold CV to examine the stability of the cross-validation estimates. If
they do not differ much, then a simple $k$-fold CV is likely sufficient. However, if they vary
substantially across implementations (i.e., demonstrate high variance), then the estimates from
the simple $k$-fold CV should be interpreted with caution, and a repeated cross-validation method
should be considered instead. When possible, readers should seek to increase computational
capacity and use repeated cross-validation methods. Below, we provide an example of how these
guidelines could work in practice.

For the Machiavellianism example described earlier, we conducted a simulation to
compare two different cross-validation methods: (a) simple 10-fold CV and (b) repeated 10-fold
CV (with 100 repetitions). 10 different sample size conditions were examined, with sample sizes
varying from $N = 50$ to $N = 30,000$. For each sample size condition, 100 samples (e.g., 100
samples of size 50) were drawn from the population, and for each sample, both (a) simple 10-
fold CV and (b) repeated 10-fold CV were implemented. The variance (i.e., standard deviation)
of the $R^2_{CV}$ and $MSE_{CV}$ and the computational time taken to run each of the cross-validation
procedures were recorded, and averaged by sample size conditions; these results are shown in
Figures 3a to 3c.\textsuperscript{10}

\textsuperscript{10}This simulation was run on a laptop with an average computational capability (a Core-i5-7300U CPU at 2.6GHZ,
8 GB RAM, Windows 10 Operating System). Generally, without parallel computing, a repeated $k$-fold CV (with 100
repetitions) takes about 100 times the running time required for a simple $k$-fold CV. The time difference might differ
when hyperparameter tuning was used to improve the models.
In our example, when sample size was smaller than 500, the variance in the cross-validation estimates (i.e., standard deviations of $R^2_{CV}$ and $MSE_{CV}$) was much higher for simple $k$-fold CV than for repeated $k$-fold CV [see Figure 3, (a) and (b)], whereas the absolute difference in computational time between the two methods was only a few seconds [see Figure 3, (c)]. Thus, when sample sizes were smaller than 500, repeated $k$-fold CV seems to be the clear method of choice. When sample size was between 500 and 5,000, simple- and repeated-$k$-fold CV were similar in terms of the variance of cross-validation estimates and computational time. When sample size was larger than 5,000, simple and repeated $k$-fold CV provided cross-validation estimates with similar variance, but simple $k$-fold CV was much faster to run than the repeated $k$-fold CV.

In the above example, we used simulation to demonstrate how the variance of the cross-validation estimates and computational time differ depending on the cross-validation method and sample size. However, the sample sizes mentioned above are not meant to be universal benchmarks for choosing between simple and repeated $k$-fold CV: Such choice is highly dependent on specific scenarios (i.e., sample and model). As we described earlier, the variance and computational cost associated with each scenario should be taken into account when choosing among cross-validation methods.

The Versatility of Cross-Validation

In this tutorial, we used multiple regression to discuss cross-validation as a method for evaluating model generalizability. Although many indices are already available for assessing model generalizability (e.g., adjusted $R^2$; e.g., Browne, 2000), one advantage of cross-validation is its versatility: It can be adapted for use with many statistical models, and for many different purposes.
Cross-validation does not rely on statistical assumptions (e.g., multivariate normality) and works with almost all types of models. Cross-validation could be used to select the model (or model parameters) that yield the best prediction accuracy. This practice is known as hyperparameter tuning (see Bergstra, Bardenet, Bengio, & Kégl, 2011; Pedregosa et al., 2011; Kuhn & Johnson, 2013, p. 66). Hyperparameter tuning is part of the standard procedure of many machine learning models: It could help optimize for the degree of polynomial terms used in a linear regression model, the maximum depth allowed for in a decision tree model, and the number of neurons used in neural network model, among others. Hyperparameter tuning, or model selection via cross-validation, could be achieved with various statistical tools, such as the caret package, as well as the cv.glmnet() function in the glmnet package in R.

Cross-validation is particularly useful—and especially important—in high-dimensional situations (e.g., $p \gg n$), where there are many predictors. First, models fitted on high-dimensional data tend to overfit, and thus we recommend using cross-validation to evaluate model generalizability in high-dimensional situations (James et al., 2013; for instructions on how to conduct cross-validation in high-dimensional situations, see Hastie et al., 2009). Second, cross-validation could also be used to minimize model overfit in high-dimensional data. For example, regularization technique, a promising approach to minimize model overfit, often relies on cross-validation (specifically, hyperparameter tuning) to find the best parameters that minimize model overfit.\footnote{Other approaches to minimize model overfit in high-dimensional situations include dimension reduction approaches (e.g., principal components analysis, singular value decomposition) that aim to reduce the $p$ predictors to a smaller number of dimensions/components and feature selection approaches (also known as variable selection, subset selection; e.g., best subset selection) that can also be employed to reduce the number of predictors by identifying only a subset of the $p$ predictors that are most related to the outcome.} In short, cross-validation is a versatile method that helps us evaluate
model generalizability, conduct model selection, and reduce model overfit in high-dimensional situations.

Summary

An ongoing concern in the field of psychology revolves around the difficulty in reproducing results (obtained in an original study) in subsequent replication efforts (e.g., Open Science Collaboration, 2015). Even when the presence or absence of an effect is reproduced in a subsequent study, the effect is often smaller than what was initially reported (e.g., Klein et al., 2018). This is actually less unexpected than it would seem—due to model overfit, the observed effect size in a given sample tends to overstate the true effect size in a population (e.g., Wherry, 1931) and the observed effect size in a new sample (Lord, 1950). In this tutorial, our goal has been to demonstrate cross-validation as a method for obtaining a more accurate estimate of the magnitude of effect sizes in new samples.

In particular, we discussed model generalizability by explaining and demonstrating model overfit, and how it results in validity shrinkage in new samples. Next, we reviewed the basic steps of cross-validation (see Table 1), and discussed two common cross-validation methods (i.e., k-fold cross-validation and Monte-Carlo cross-validation, see Table 2). Finally, we demonstrated the methods using an empirical dataset, and provided a step-by-step demonstration of how to implement the cross-validation methods using the caret R package (see Appendix C).

Cross-validation does not substitute replication efforts; in fact, they are conceptually distinct and complementary in addressing issues related to robust and reliable science (Bollen, Cacioppo, Kaplan, Krosnick, Olds & Dean, 2015): Cross-validation is mainly focused on

---

12 Other factors, such as publication bias, sampling error, and model underfit, could also contribute to poor replicability (Open Science Collaboration, 2015).
whether a particular fitted model performs well in a new sample; replicability efforts often focus on whether we could observe effects that are similar to the original study. Although replication efforts are irreplaceable, not all (in fact, a very small number of) research teams have the resources to conduct large-scale replication studies. Cross-validation could provide important information regarding generalizability that is readily applicable for a broad scope of research, thus providing an invaluable tool to advance reliable and robust science.
References

Arlot, S., & Celisse, A. (2010). A survey of cross-validation procedures for model selection. *Statistics Surveys, 4*, 40-79.

Bergstra, J. S., Bardenet, R., Bengio, Y., & Kégl, B. (2011). Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems* (pp. 2546-2554).

Bollen, K., Cacioppo, J. T., Kaplan, R. M., Krosnick, J. A., Olds, J. L., & Dean, H. (2015). Social, behavioral, and economic sciences perspectives on robust and reliable science. *Report of the Subcommittee on Replicability in Science Advisory Committee to the National Science Foundation Directorate for Social, Behavioral, and Economic Sciences, 3*, 3.

Bosco, F. A., Aguinis, H., Singh, K., Field, J. G., & Pierce, C. A. (2015). Correlational effect size benchmarks. *Journal of Applied Psychology, 100*, 431-449.

Browne, M. W. (2000). Cross-validation methods. *Journal of Mathematical Psychology, 44*(1), 108-132.

Christie, R. & Geis, F. (1970). *Studies in Machiavellianism*. NY: Academic Press.

de Rooij, M., Pratiwi, B. C., Fokkema, M., Dusseldorp, E., & Kelderman, H. (2019). The early roots of statistical learning in the psychometric literature: A review and two new results. *arXiv preprint arXiv:1911.11463*.

Geisser, S. (1975). The predictive sample reuse method with applications. *Journal of the American Statistical Association, 70*, 320-328.

Gosling, S. D., Rentfrow, P. J., & Swann, W. B., Jr. (2003). A very brief measure of the Big Five personality domains. *Journal of Research in Personality, 37*, 504-528.
Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The elements of statistical learning: Data mining, inference, and prediction*. Springer Series in Statistics.

Hebb, D. O. (1955). Drives and the C.N.S. (conceptual nervous system). *Psychological Review, 62*, 243-254.

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An introduction to statistical learning* (Vol. 112, pp. 3-7). New York: Springer.

Kim, J. H. (2009). Estimating classification error rate: Repeated cross-validation, repeated hold-out and bootstrap. *Computational Statistics and Data Analysis, 53*, 3735-3745.

Klein, R. A., Vianello, M., Hasselman, F., Adams, B. G., Adams Jr, R. B., Alper, S., ... & Batra, R. (2018). Many Labs 2: Investigating variation in replicability across samples and settings. *Advances in Methods and Practices in Psychological Science, 1*, 443-490.

Kohavi, R. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. *International Joint Conference on Artificial Intelligence, 1*, 1137-1143.

Koul, A., Becchio, C., & Cavallo, A. (2018). Cross-validation approaches for replicability in psychology. *Frontiers in Psychology, 9*, 1117.

Kuhn, M. (2008). Building predictive models in R using the caret package. *Journal of Statistical Software, 28*, 1-26.

Kuhn (2019). *Simple splitting with important groups*. Retrieved Nov. 3rd, 2019, from http://topepo.github.io/caret/data-splitting.html#simple-splitting-with-important-groups

Kuhn, M., & Johnson, K. (2013). *Applied Predictive Modeling*. New York: Springer.

Larson, S. C. (1931). The shrinkage of the coefficient of multiple correlation. *Journal of Educational Psychology, 22*, 45.
Lord, F. M. (1950). Efficiency of prediction when a regression equation from one sample is used in a new sample. *ETS Research Bulletin Series, 1950*, i-6.

Lovakov, A., & Agadullina, E. R. (2017). Empirically derived guidelines for interpreting effect size in social psychology. Retrieved from https://psyarxiv.com/2epc4/download/?format=pdf

Molinaro, A. M., Simon, R., & Pfeiffer, R. M. (2005). Prediction error estimation: A comparison of resampling methods. *Bioinformatics, 21*, 3301-3307.

Narendra, P. M., & Fukunaga, K. (1977). A branch and bound algorithm for feature subset selection. *IEEE Transactions on Computers, 9*, 917-922.

Mosier, C. I. (1951). I. Problems and designs of cross-validation 1. *Educational and Psychological Measurement, 11*, 5-11.

Open Science Collaboration. (2015). Estimating the reproducibility of psychological science. *Science, 349*(6251), aac4716.

Paterson, T. A., Harms, P. D., Steel, P., & Credé, M. (2016). An assessment of the magnitude of effect sizes: Evidence from 30 years of meta-analysis in management. *Journal of Leadership & Organizational Studies, 23*, 66-81.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Vanderplas, J. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research, 12*(Oct), 2825-2830. Retrieved from: https://scikit-learn.org/stable/modules/grid_search.html#grid-search

Picard, R. R., & Cook, R. D. (1984). Cross-validation of regression models. *Journal of the American Statistical Association, 79*, 575-583.
CROSS-VALIDATION TUTORIAL

R Core Team (2019). R: A language and environment for statistical computing. *R Foundation for Statistical Computing*, Vienna, Austria. URL https://www.R-project.org/.

Roberts, D. R., Bahn, V., Ciuti, S., Boyce, M. S., Elith, J., Guiller-Arroita, G., ... & Warton, D. I. (2017). Cross-validation strategies for data with temporal, spatial, hierarchical, or phylogenetic structure. *Ecography, 40*, 913-929.

Rozeboom, W. W. (1981). The cross-validational accuracy of sample regressions. *Journal of Educational Statistics, 6*, 179-198.

Shen, W., Kiger, T. B., Davies, S. E., Rasch, R. L., Simon, K. M., & Ones, D. S. (2011). Samples in applied psychology: Over a decade of research in review. *Journal of Applied Psychology, 96*, 1055-1064.

Stone, M. (1974). Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society, 36*, 111-133.

Wherry, R. J. (1931). A new formula for predicting the shrinkage of the coefficient of multiple correlation. *Annals of Mathematical Statistics, 2*, 440–457.

Yarkoni, T., & Westfall, J. (2017). Choosing prediction over explanation in psychology: Lessons from machine learning. *Perspectives on Psychological Science, 12*(6), 1100-1122.

Zhang, Y., & Yang, Y. (2015). Cross-validation for selecting a model selection procedure. *Journal of Econometrics, 187*(1), 95–112.
TABLES

Table 1

*General Steps of Cross-Validation*

| Steps | Description | Example: Regression |
|-------|-------------|---------------------|
| **Step 1** | Split the data set into a training set and a test set according to the chosen cross-validation method. | Randomly split a dataset into training and test sets, according to the specific cross-validation method. For example, in a 5-fold cross-validation, in each fold, 4/5 of the dataset is used as the training set, whereas 1/5 of the dataset is used as the test set. |
| **Step 2** | Fit a model to the training set and obtain the model parameters. | Fit a regression model to the training set. Obtain regression weights in the training set. |
| **Step 3** | Apply the fitted model to the test set and obtain prediction accuracy. | Use the regression weights from Step 2 to predict the outcome in the test set. Obtain the cross-validated prediction accuracy metrics (e.g., $R^2$ and $MSE$ in the test set). |
| **Step 4** | Repeat Steps 1 - 3. | For example, in a 5-fold cross-validation, Steps 1 - 3 will be repeated for each of the five folds, resulting in five cross-validated prediction accuracy estimates. |
| **Step 5** | Aggregate all prediction accuracy results from Step 4. | For example, in a 5-fold cross-validation, calculate the average of the five cross-validated prediction accuracy estimates, obtained from each of the five folds. |
## CROSS-VALIDATION TUTORIAL

### Table 2

**Comparison of k-Fold Cross-Validation and Monte-Carlo Cross-Validation**

| Step | Description | Example (sample size \(N = 200\)) |
|------|-------------|----------------------------------|
| **0** | Prepare data. | Randomly shuffle the observations and divide the dataset into 10 roughly equal subsets. |
| **1** | Split the dataset into a training set and a test set. | Select the first subset as the test set (\(200 \times 0.1 = 20\) observations) and use the remaining dataset as the training set (180 observations). |
| **2** | Fit a model to the training set and obtain the model parameters. | The model is fitted to the training set (180 observations). |
|  | Apply the fitted model parameters to the test set and obtain the cross-validated prediction accuracy. | The model obtained from Step 2 is tested in the test set (20 observations). |
| **3** | Repeat Steps 1 - 3. | Repeat Steps 1 - 3 for each of the 10 folds (10 times). |
| **4** | Calculate the average of all prediction accuracy results from Step 4. | There are a total of 10 prediction accuracy results, one for each of the 10 folds. The average of these results is the overall prediction accuracy estimate. |
| **5** | | There are a total of 100 prediction accuracy results, one for each of the 100 repetitions. The average of these results is the overall prediction accuracy estimate. |

**k-fold CV**

\( (k = 10) \)

**MCCV**

\( (p = .8, Rep = 100) \)

- Decide on \(p\) and \(Rep\).
- \(p\) is the proportion of observations randomly sampled as the training set; \(Rep\) is the number of repetitions.

- Randomly sample 160 observations (without replacement; \(200 \times 0.8 = 160\)) from the dataset. This is the training set. The remaining 40 observations are the test set.
Figure 1a. An example of calibration sample in Shiny app
Figure 1b. An example of the simple regression model (calibration sample with residual errors) in the Shiny app
Figure 1c. An example of the simple regression model (calibration sample with residual errors, $R^2$, and MSE) in the Shiny app
Figure 1d. An example of the simple regression model (calibration and validation sample) in the Shiny app
Figure 2. An example of a 5-fold cross-validation with 3rd-order polynomial regression model in the Shiny app
Figure 3. Comparison between simple- and repeated-\(k\)-fold CV methods. For each sample size condition, we ran 100 trials of both simple 10-fold CV and repeated 10-fold CV and recorded the resulting cross-validated \(MSE\) and cross-validated \(R^2\). \(SD\) of \(MSE\) and \(SD\) of \(R^2\) are the standard deviation of the 100 \(MSE\)s and \(R^2\)’s. Time (seconds) is the average time needed to run a cross-validation trial, represented in seconds.
Appendix A

Simulation is used to further demonstrate Observations 1, 2 and 3. Specifically, as each sample draw is influenced by sampling variation, results from the Shiny app may not always be as expected (i.e., by chance, it is possible for the calibrated model to fit better in the validation sample than in the calibration sample). We conducted three simulations, to help illustrate the expected result for each of the three observations. In the simulations, each sample draw and model fit were repeated 1,000 times, then the results were obtained by averaging across the 1,000 trials. Unless otherwise stated, we assume a quadratic relationship between arousal and performance. The overall simulation results support the observations that the model generalizability decreases as (a) the model becomes more complex, (b) calibration sample size decreases, and (c) effect size decreases.

```r
# Define a function to generate samples for simulation
generate_sample <- function(N, R2){
  mean_arousal <- 0
  sd_arousal <- 3
  arousal <- rnorm(n = N, mean = mean_arousal, sd = sd_arousal)
  performance <- (300 - (arousal - 6)^2) / 60

  # add noise to performance according to the value of R2 (R-squared)
  snr <- R2 / (1 - R2) # calculate signal-to-noise ratio
  # (using equation from Hastie et al. (2009); p.401)
  sd_performance <- 0.6364 # this sd is approximated using simulation
  sd_noise <- sd_performance / sqrt(snr)
  noise <- rnorm(n = N, sd = sd_noise) # generate noise around 0
  performance <- performance + noise
  dat <- data.frame(arousal = arousal, performance = performance)
  return(dat)
}
```

13 All code is available at https://github.com/qcsong/CrossValidationTutorial.
Observation 1: The Model Overfits the Calibration Sample

In the following code, the \texttt{lm()} function was used to fit the regression models.

```r
# Set a random seed
set.seed(2020)

# Set population effect size
R2_pop <- 0.25

# Set calibration sample size
n_cal <- 50

# Set model complexity (degree of polynomial)
degree <- 2

# Set validation sample size
n_val <- 1000

# According to this population R-squared value,
# the population MSE can be computed as
MSE_pop <- 0.6364^2 / (R2_pop / (1 - R2_pop))
# Population MSE is the sum of the squared residuals,
# which is equivalent to the variance of noise
# in the generate_sample() function

# Create vectors to store the calibration sample R-squared
# and MSE for each trial
R2 <- NA
MSE <- NA

for(i in 1:1000) {
  # The following procedure is repeated 1000 times.
  data <- generate_sample(N=n_cal, R2=R2_pop)

  # Fit the regression model to the sample and save
  # the resulting model as “ob1_mod”
  ob1_mod <- lm(performance ~ poly(arousal, degree), data=data)

  # Store R-squared and MSE for each trial
  R2[i] <- cor(ob1_mod$fitted.values, data$performance)^2
  MSE[i] <- mean(summary(ob1_mod)$residuals^2)
}

# Calculate the average calibration R-squared and MSE,
# averaged across 1000 trials
R2_cal <- mean(R2)
```
MSE_cal <- mean(MSE)

# Output the results
cat(paste0("R-squared of the population model: ", round(R2_pop, 2), "\n", "R-squared of the current model: ", round(R2_cal, 2), "\n", "MSE of the population model: ", round(MSE_pop, 2), "\n", "MSE of the current model: ", round(MSE_cal, 2), "\n"))

## R-squared of the population model: 0.25
## R-squared of the current model: 0.27
## MSE of the population model: 1.22
## MSE of the current model: 1.13

The output showed that the $R^2$ for the current model is larger than the population $R^2$. Meanwhile, $MSE$ for the current model is smaller than the population $MSE$. This is because the model is capturing sample-specific variations that are unrepresentative of the population. As a result, the model is overfitted to the calibration sample.

**Observation 2: The Model Obtained from the Calibration Sample Tends to Not Generalize Well to New (Validation) Samples**

Apply the model obtained from the calibration sample to a new (validation) sample and check the prediction accuracy, $R^2_{\text{validation}}$ and $MSE_{\text{validation}}$.

# Set a random seed
set.seed(2020)

# Set population effect size
R2_pop <- .25

# Set calibration sample size
n_cal <- 50

# Set model complexity (degree of polynomial)
degree <- 2

# Set validation sample size
n_val <- 1000

# Create vectors to store the results for each trial
CROSS-VALIDATION TUTORIAL

R2_cal <- NA
MSE_cal <- NA
R2_val <- NA
MSE_val <- NA

for(i in 1:1000) {

  # The following procedure is repeated 1000 times.
  data_cal <- generate_sample(N=n_cal, R2=R2_pop)
  data_val <- generate_sample(N=n_val, R2=R2_pop)
  ob2_mod <- lm(performance ~ poly(arousal, degree), data=data_cal)
  R2_cal[i] <- cor(ob2_mod$fitted.values, data_cal$performance)^2
  MSE_cal[i] <- mean(summary(ob2_mod)$residuals^2)

  # Use the fitted regression model to predict task performance
  # from arousal in the validation sample
  yhat_val <- predict(ob2_mod, data_val)

  # Calculate validation R-squared and MSE
  R2_val[i] <- cor(yhat_val, data_val$performance)^2
  MSE_val[i] <- mean((yhat_val - data_val$performance)^2)
}

# Calculate the average validation sample R-squared and MSE,
# averaged across 1000 trials
R2_cal_mean <- mean(R2_cal)
R2_val_mean <- mean(R2_val)
MSE_cal_mean <- mean(MSE_cal)
MSE_val_mean <- mean(MSE_val)

# Output the results
cat(paste0("Average Calibration R-squared: ", round(R2_cal_mean, 2), "\n",
  "Average Validation R-squared: ", round(R2_val_mean, 2), "\n",
  "Average Calibration MSE: ", round(MSE_cal_mean, 2), "\n",
  "Average Validation MSE: ", round(MSE_val_mean, 2), "\n"))

## Average Calibration R-squared: 0.27
## Average Validation R-squared: 0.23
## Average Calibration MSE: 1.14
## Average Validation MSE: 1.31

Notice that $R^2_{validation}$ is smaller than $R^2_{calibration}$: there is a 15% reduction in $R^2$ ($\frac{0.27 - 0.23}{0.27} = 0.15$). Also, $MSE_{validation}$ is larger than $MSE_{calibration}$: there is a 0.17 increase in $MSE$ ($1.31 - 1.14 = 0.17$). This suggests that when a model fitted on one (calibration) sample is
used to make predictions in a new (validation) sample, the model performs less well in the new sample.

**Observation 3: Model Generalizability is Influenced by (a) Model Complexity, (b) Sample Size, and (c) Effect Size**

In Observation 3, we will vary (a) model complexity, (b) calibration sample size, and (c) effect size and compare the results with Observation 2.

**Observation 3a: The model generalizes less well when the model is complex**

Now let us explore how model complexity influences model generalizability. Increase the model complexity by specifying a cubic (instead of quadratic) regression model. Then, apply the calibrated models to new samples and obtain prediction accuracy.

```r
# Set a random seed
set.seed(2020)

# Set population effect size
R2_pop <- .25

# Set calibration sample size
n_cal <- 50

# Set model complexity (degree of polynomial)
degree <- 3 # Cubic regression, a more complex model

# Set validation sample size
n_val <- 1000

# Create vectors to store the calibration and validation sample R-squared for each trial
R2_cal <- NA
MSE_cal <- NA
R2_val <- NA
MSE_val <- NA

for(i in 1:1000) {
  # The following procedures are repeated 1000 times
data_cal <- generate_sample(N=n_cal, R2=R2_pop)
data_val <- generate_sample(N=n_val, R2=R2_pop)
  ```
ob3a_mod <- lm(performance ~ poly(arousal, degree), data=data_cal)
R2_cal[i] <- cor(ob3a_mod$fitted.values, data_cal$performance)^2
MSE_cal[i] <- mean(summary(ob3a_mod)$residuals^2)
# Use the fitted regression model to predict task performance
# from arousal in the validation sample
yhat_val <- predict(ob3a_mod, data_val)

# Calculate validation R-squared and MSE
R2_val[i] <- cor(yhat_val, data_val$performance)^2
MSE_val[i] <- mean((yhat_val - data_val$performance)^2)
}

# Calculate the average validation sample R-squared,
# averaged across 1000 trials
R2_cal_mean <- mean(R2_cal)
R2_val_mean <- mean(R2_val)
MSE_cal_mean <- mean(MSE_cal)
MSE_val_mean <- mean(MSE_val)

# Output the results
cat(paste0(
  "Average Calibration R-squared: ", round(R2_cal_mean, 2), "\n",
  "Average Validation R-squared: ", round(R2_val_mean, 2), "\n",
  "Average Calibration MSE: ", round(MSE_cal_mean, 2), "\n",
  "Average Validation MSE: ", round(MSE_val_mean, 2), "\n"))

## Average Calibration R-squared: 0.28
## Average Validation R-squared: 0.2
## Average Calibration MSE: 1.11
## Average Validation MSE: 1.43

Across 1,000 trials, $R^2_{validation}$ is smaller for the cubic regression model ($R^2_{validation} = .20$) than for the quadratic regression model ($R^2_{validation} = .23$, see Observation 2). Similarly, $MSE_{validation}$ is larger for the cubic regression model ($MSE_{validation} = 1.43$) than for the quadratic regression model ($MSE_{validation} = 1.31$, see Observation 2). Consistent with Observation 3a, a complex model generalizes less well than a simpler model when applied to a new sample.

Observation 3b: The model generalizes less well when calibration sample size is small
Examine model generalizability for a smaller calibration sample size of 30 (as compared to 50). Apply the model obtained from the calibration sample to new samples (validation samples) and check the magnitude of prediction accuracy.

```r
# Set a random seed
set.seed(2020)

# Set population effect size
R2_pop <- .25

# Set calibration sample size
n_cal <- 30 # smaller sample size

# Set model complexity (degree of polynomial)
degree <- 2

# Set validation sample size
n_val <- 1000

# Create vectors to store the calibration and validation sample R-squared and MSE for each trial
R2_cal <- NA
MSE_cal <- NA
R2_val <- NA
MSE_val <- NA

for(i in 1:1000) {
  # The following procedures are repeated 1000 times.
  data_cal <- generate_sample(N=n_cal, R2=R2_pop)
  data_val <- generate_sample(N=n_val, R2=R2_pop)
  ob3b_mod <- lm(performance ~ poly(arousal, degree), data=data_cal)
  R2_cal[i] <- cor(ob3b_mod$fitted.values, data_cal$performance)^2
  MSE_cal[i] <- mean(summary(ob3b_mod)$residuals^2)

  # Use the fitted regression model to predict task performance from arousal in the validation sample
  yhat_val <- predict(ob3b_mod, data_val)

  # Calculate validation R-squared and MSE
  R2_val[i] <- cor(yhat_val, data_val$performance)^2
  MSE_val[i] <- mean((yhat_val - data_val$performance)^2)
}

# Calculate the average calibration and validation sample R-squared and MSE,
Notice that, across 1,000 trials, the average $R^2_{validation} = .22$ is smaller when the same quadratic regression model was fitted on a small sample size ($n = 30$), as compared to when the quadratic regression model was fitted on a larger sample size ($n = 50$; average $R^2_{validation} = .23$, see Observation 2). Similarly, the average $MSE_{validation} = 1.38$ is larger when the same quadratic regression model was fitted on a small sample size ($n = 30$), as compared to when the quadratic regression model was fitted on a larger sample size ($n = 50$; average $MSE_{validation} = 1.31$, see Observation 2). This suggests that a model fitted on a smaller calibration sample tends to generalize less well in a new sample, as compared to a model fitted on a larger calibration sample.

**Observation 3c: The model generalizes less well when the population effect size is small**

Examine model generalizability when the population effect size is $\rho^2 = .04$ (as compared to $\rho^2 = .25$). Apply the model obtained from the calibration sample to new samples and check the magnitude of prediction accuracy.
# Set population effect size
R2_pop <- .04 # smaller effect size

# Set calibration sample size
n_cal <- 50

# Set model complexity (degree of polynomial)
degree <- 2

# Set validation sample size
n_val <- 1000

# Create vectors to store the calibration
# and validation sample R-squared for each trial
R2_cal <- NA
MSE_cal <- NA
R2_val <- NA
MSE_val <- NA

for(i in 1:1000) {

  # The following procedures are repeated 1000 times.
  data_cal <- generate_sample(N=n_cal, R2=R2_pop)
  data_val <- generate_sample(N=n_val, R2=R2_pop)
  ob3c_mod <- lm(performance ~ poly(arousal, degree), data=data_cal)
  R2_cal[i] <- cor(ob3c_mod$fitted.values, data_cal$performance)^2
  MSE_cal[i] <- mean(summary(ob3c_mod)$residuals^2)

  # Use the fitted regression model to predict task performance
  # from arousal in the validation sample
  yhat_val <- predict(ob3c_mod, data_val)

  # Calculate validation R-squared and MSE
  R2_val[i] <- cor(yhat_val, data_val$performance)^2
  MSE_val[i] <- mean((yhat_val - data_val$performance)^2)
}

# Calculate the average validation sample R-squared,
# averaged across 1000 trials
R2_cal_mean <- mean(R2_cal)
R2_val_mean <- mean(R2_val)
MSE_cal_mean <- mean(MSE_cal)
MSE_val_mean <- mean(MSE_val)

# Output the results
cat(paste0(
  "Average Calibration R-squared: ", round(R2_cal_mean, 2), "\n",
  "Average Validation R-squared: ", round(R2_val_mean, 2), "\n",})
"Average Calibration MSE: \( \text{round(MSE_cal_mean, 2), } \backslash n\) \" 
"Average Validation MSE: \( \text{round(MSE_val_mean, 2), } \backslash n\)\)  

## Average Calibration R-squared: 0.08 
## Average Validation R-squared: 0.03 
## Average Calibration MSE: 9.1 
## Average Validation MSE: 10.48 

Notice that, when the population effect size is .04, across 1,000 trials, the average 
\[ R_{validation}^2 = .03 \] is less than half the average \[ R_{calibration}^2 = .08 \] (63% reduction versus 15% 
reduction in Observation 2), and the average \[ MSE_{validation} = 10.48 \] is less than the average 
\[ MSE_{calibration} = 9.10 \] (1.39 increase versus 0.17 increase in Observation 2). This suggests that, 
when the population effect size is small, the model tends to generalize less well in a new sample.
Appendix B

$k$-Fold Cross-Validation

The R code below provides a step-by-step demonstration of a 5-fold cross-validation process.

```r
# Set random seed
set.seed(2020)

# Sample size
n <- 300

# Set population effect size
R2_pop <- .16

cv_data <- generate_sample(N=n, R2=R2_pop)

# Specify a cubic regression model
degree <- 3

# Fit the cubic regression model using the complete dataset and save the resulting model as "cv_original_mod"
cv_original_mod <- lm(performance ~ poly(arousal, degree), data=cv_data)

# Obtain R-squared and MSE of the model fitted to the complete dataset
R2 <- cor(cv_original_mod$fitted.values, cv_data$performance)^2
MSE <- mean(summary(cv_original_mod)$residuals^2)

# Specify the number of folds
k <- 5

# Split the index of each observation into k equal subsets
subsets <- split(x=1:n, f=sort(rep_len(x=1:k, length.out=n)))

# Iterate through each fold
R2_test <- NA
MSE_test <- NA

for(i in 1:k) {

    # Split the dataset into k equal subsets
    ind_test <- subsets[[i]]

    # For example, when i=1, the first subset of data
```
# (1/5 of the original dataset) is specified as the test set
data_test <- cv_data[ind_test, ]

# Specify the rest of the data (4/5 of the original data)
# as the training set
data_train <- cv_data[-ind_test, ]

# Fit the cubic regression model to the training set
mod_tmp <- lm(performance ~ poly(arousal, degree), data=data_train)

# Test the model on the test set and calculate
# R-squared and MSE for this fold (iteration)
yhat_test <- predict(mod_tmp, data_test)
R2_test[i] <- cor(yhat_test, data_test$performance)^2
MSE_test[i] <- mean((yhat_test - data_test$performance)^2)
}

# Average Cross-validated R-squared and MSE (averaged across 5 folds)
R2_kfold <- mean(R2_test)
MSE_kfold <- mean(MSE_test)

# Output the results
cat(paste0(
  "Model R-squared: ", round(R2, 2), "\n",
  "Model MSE: ", round(MSE, 2), "\n",
  "Cross-validated R-squared: ", round(R2_kfold, 2), "\n",
  "Cross-validated MSE: ", round(MSE_kfold, 2), "\n")))

## Model R-squared: 0.25
## Model MSE: 1.97
## Cross-validated R-squared: 0.24
## Cross-validated MSE: 2.01
Monte Carlo Cross-Validation (MCCV)

The R code below provides a step-by-step demonstration of a Monte Carlo cross-validation process. In this example, we use 80% of the sample to train the model and the remaining 20% to test the model. This is repeated 100 times to obtain an estimate of the cross-validated $R^2$ and $MSE$.

```r
# Using the same dataset and model as the above k-fold example

set.seed(2020)

# Specify to repeat 100 times
rep <- 100

# Set the test set size to 1/5 (20%) of the total sample size
n_test <- ceiling(n / 5) # n_test = 60 for sample size 300

R2_test <- NA
MSE_test <- NA

for(i in 1:rep) {

  # Repeat 100 times, each time randomly drawing 60 observations 
  # as the test set and the rest as the training set
  ind_test <- sample(1:n, n_test) # randomly draw 60 numbers from 1 to 300

  # use the 60 numbers draw as row indices and get the test set
  data_test <- cv_data[ind_test, ]

  # Specify the remaining 240 observations as the training set
  data_train <- cv_data[-ind_test, ]

  # Fit the model with the training set
  mod_temp <- lm(performance ~ poly(arousal, degree), data=data_train)

  # Use the fitted model to make predictions in test set
  yhat_test <- predict(mod_temp, data_test)

  # Save cross-validated R-squared and MSE for each repetition
  R2_test[i] <- cor(yhat_test, data_test$performance)^2
  MSE_test[i] <- mean((yhat_test - data_test$performance)^2)
}
```
# Average cross-validated R-squared (averaged across 100 repetitions)
R2_mccv <- mean(R2_test)
MSE_mccv <- mean(MSE_test)

# Output the results
cat(paste0(
  "Model R-squared: ", round(R2, 2), "\n",
  "Model MSE: ", round(MSE, 2), "\n",
  "Cross-validated R-squared: ", round(R2_mccv, 2), "\n",
  "Cross-validated MSE: ", round(MSE_mccv, 2), "\n"))

## Model R-squared: 0.25
## Model MSE: 1.97
## Cross-validated R-squared: 0.24
## Cross-validated MSE: 2.02
Appendix C

Empirical Example Using the Machiavellianism Dataset

In this example, we treat the 71,992 observations as the population and fit a regression model to obtain the population estimates (see R code below). As a result, in the population,

\[ R^2_{\text{population}} = 0.28, \quad \text{and} \quad MSE_{\text{population}} = 0.45. \]

The R code in this appendix is also available at https://github.com/qcsong/CrossValidationTutorial.

```r
# If the caret package is not installed run below line to install it
# install.packages("caret")

# Load the package
caret

# The following dataset (from https://openpsychometrics.org/_rawdata/) # is used by the current overfitting and cross-validation example. # R Code for data preparation is available from the authors.

# Import data
data <- read.csv("https://git.io/JfsiA")

# Fit the model on the complete dataset
mod <- lm(mach ~ age + as.factor(gender) + O + C + E + A + N, data=data)

# Summary of the regression model fitted on the complete dataset
R2_pop <- round(summary(mod)$r.squared, 2)
MSE_pop <- round(mean(mod$residuals^2), 2)
cat(paste0("R2 of the population model is ", R2_pop, ";n", 
"MSE of the population model is ", MSE_pop, "."))

## R2 of the population model is 0.28;
## MSE of the population model is 0.45.

Model Overfitting

Let us use a (calibration) sample from this population and fit the same model.

sample_cal <- read.csv("https://git.io/JfsPt")

# This is a random sample drawn from the whole Mach dataset using below code.
```

---

14 This example dataset was cleaned based on the original dataset, and it is available in the GitHub repository: https://github.com/qcsong/CrossValidationTutorial.
# Interested readers can vary the random seed to test with different samples.

```r
# set.seed(2020)
# n <- 300
# sample_cal <- data[sample(nrow(data), n), ]

mod_cal <- lm(mach ~ age + as.factor(gender) + O + C + E + A + N, 
              data=sample_cal)

# Calculate and save results
R2_cal <- round(summary(mod_cal)$r.squared, 2)
MSE_cal <- round(mean(mod_cal$residuals^2), 2)

# Print results
cat(paste0("R2 of the calibration model is ", R2_cal, ";\n", 
            "MSE of the calibration model is ", MSE_cal, ".\n"))

## R2 of the calibration model is 0.3;
## MSE of the calibration model is 0.43.
```

We notice that in the sample (N = 300), the regression model has a larger $R^2$ and a smaller $MSE$ than the population model.

**Cross-Validation**

**k-fold cross-validation.** We will carry out a 10-fold cross-validation. First, call the `trainControl()` function, and specify the method as k-fold cross-validation (i.e., “cv”), where the number of folds is equal to 10. Save the specified information in an object named `kfold_train_control`. Then, call the `train()` function to implement the k-fold cross-validation. The cross-validation results are saved in the object, “kfold_cv”.

```r
set.seed(2020) # Set a random seed to replicate results

k <- 10 # Number of folds

kfold_train_control <- trainControl(method="cv", number=k)

kfold_cv <- train(mach ~ age + as.factor(gender) + O + C + E + A + N, 
                   data=sample_cal, method="lm", trControl=kfold_train_control)

# Calculate and save results
```

```r
```
CROSS-VALIDATION TUTORIAL

MSE_kfold <- round(mean(kfold_cv$resample$RMSE^2), 2)
R2_kfold <- round(kfold_cv$results$Rsquared, 2)

# Print results
cat(paste0("k-fold cross-validated R2 is ", R2_kfold, ";
"k-fold cross-validated MSE is ", MSE_kfold, "."))

## k-fold cross-validated R2 is 0.28;
## k-fold cross-validated MSE is 0.46.

It looks like there is a lot going on in this code, so let us talk about each of the input arguments (e.g., data, method, trControl) that are being specified. First, data = sample_cal specifies that the sample_cal dataset, the dataset that was used in the original analysis, is used to conduct the cross-validation. Next, method = “lm” specifies the statistical model as we need to indicate the model that will be used. In this case, we are using a linear (regression) model, and so we specify that method = “lm” and provide the actual form of the regression model that should be fitted. This is exactly the same model as the one fitted earlier. Lastly, to specify the cross-validation method to be used, trControl = kfold_train_control tells the function that should correspond to the 10-fold cross-validation method that we specified earlier.

All the results are saved in the object called kfold_cv, we retrieved $R^2$ and $MSE$ from the kfold_cv object. Note that caret calculates $RMSE$ by default, so $MSE$ is calculated as the mean of squares the $RMSE$’s for all folds.

**Monte Carlo cross-validation (MCCV).** To conduct MCCV, we specify that method = “LGOCV” (i.e., leave-group-out cross-validation, which is another term for Monte-Carlo cross-validation) in the trainControl() function. We set the number of repetitions to be 200 (number = 200; i.e., we ask for the train-then-test procedure to be conducted 200 times). In addition, we specify the proportion of data that should be randomly held out as a test set in each
of the 200 repetitions. For example, \( p = .8 \) means that 80% of the data set will be used as the \textit{training} set, and therefore 20% (i.e., 1 - \( p \)) of the data set will be used as the \textit{test} set. These pre-
specifications will once again be saved in an object named \texttt{mc_train_control}.

Next, the \texttt{train()} function will again be used to implement MCCV. As with
conducting \textit{k}-fold cross-validation using \texttt{train()}, we specify the data, analytic method, and
linear regression equation to be used when conducting the MCCV. Similar to the \textit{k}-fold example,
results are retrieved and calculated from the final \texttt{mc_cv} object.

```r
set.seed(2020) # Set a random seed to replicate results

R <- 200 # Number of repetitions

mc_train_control <- trainControl(method="LGOCV", p=.8, number=R)

mc_cv <- train(mach ~ age + as.factor(gender) + O + C + E + A + N,
               data=sample_cal, method="lm", trControl=mc_train_control)

# Calculate and save results
MSE_mc <- round(mean(mc_cv$resample$RMSE^2), 2)
R2_mc <- round(mc_cv$results$Rsquared, 2)

# Print results
cat(paste0("Monte Carlo cross-validated R2 is ", R2_mc, ";\n", "Monte Carlo cross-validated MSE is ", MSE_mc, "."))
```

## Monte Carlo cross-validated R2 is 0.28;
## Monte Carlo cross-validated MSE is 0.46.