STA 478 Assignment #6

Dr. Robert Buscaglia

October 21, 2025

Exercises

Exercise 1

Produce a simulation of at least than `1e5` redraws from a sample of size `n = 1000` that shows the percent of data retained in a bootstrap sample. For each iteration retain the fraction of original samples retained in each bootstrap resample. Produce a histogram of your results. *On average, how many of the original samples are retained in a bootstrap sample?*

Solution.

To tackle this simulation I produced a function that produces the resampled data and evaluates the fraction of unique samples retained in the bootstrap sample. The function can control the sample size `n` and the number of bootstrap iterations `B`.

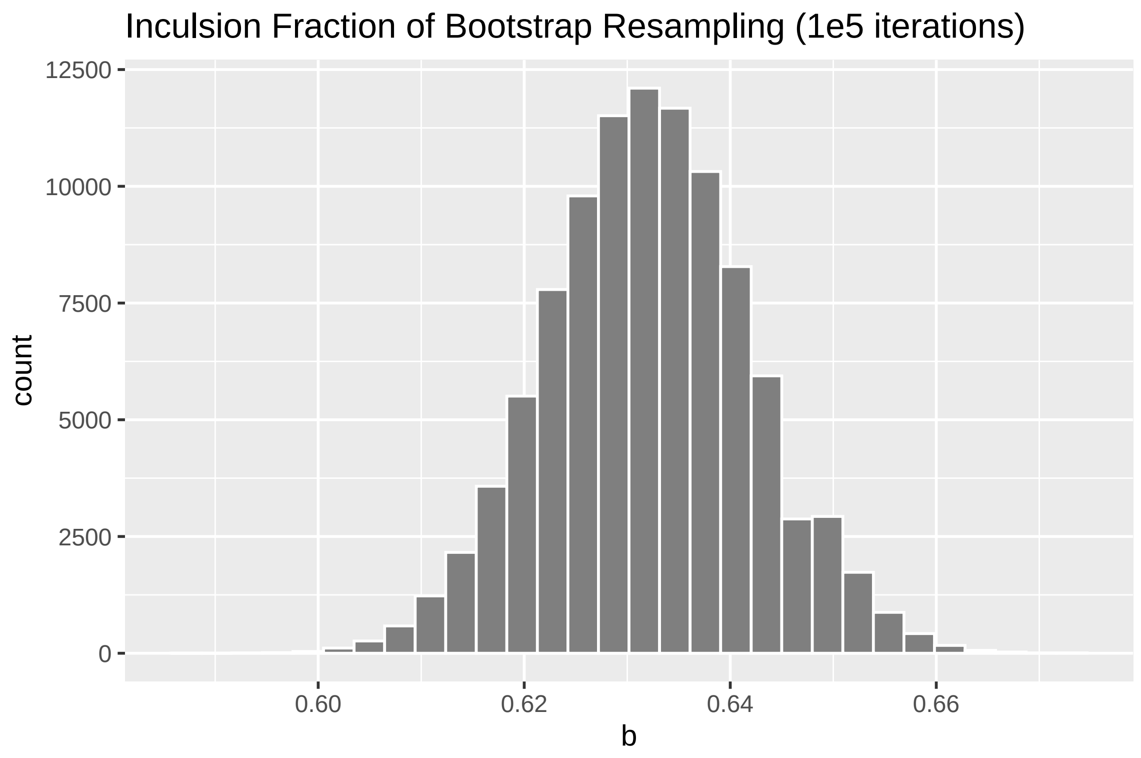
```r
bootstrap.included <- function(n, B)
{
  fraction.included <- numeric()
  for(i in 1:B)
  {
      q <- sample(n, n, replace=TRUE)
      fraction.included[i] <- length(unique(q))/n
  }
  return(fraction.included)
}
```

With the function, I complete the simulation using `n = 1000` samples and `B = 1e5` iterations. Converting the final output to a data.frame.

```r
set.seed(12)
n1000.boot <- bootstrap.included(1000, 1e5)
n1000.boot <- data.frame(b = n1000.boot)
```

A graphical inspection of the fraction retained shows that there is normality in this behavior (specifically we are looking at a large profile of binomial distributed data, which in its limit, has a behavior similar to normality).

```r
ggplot(n1000.boot, aes(x = b)) +
  geom_histogram(color = "white", fill="grey50", bins=30) +
  labs(title = "Inculsion Fraction of Bootstrap Resampling (1e5 iterations)")
```



```
n1000.boot %>% summarize(mean = mean(b))
```

```
##       mean
## 1 0.632342
```

The graph shows that the bootstrap samples typically contain about 60% of the unique observations of a data set. The actual calculated mean inclusion fraction is 0.63, which aligns well with our optimal subset size for cross-validation!

Exercise 2

ISLR Carseats data

We will use the **Carseats** data set to construct a predictive model of total *Sales*.

a

Load the data from `ISLR` and evaluate your predictors. Discuss variable types, correlations, and anything you want to note. You can always learn more about the predictors using the vignette, help, or the `?` command.

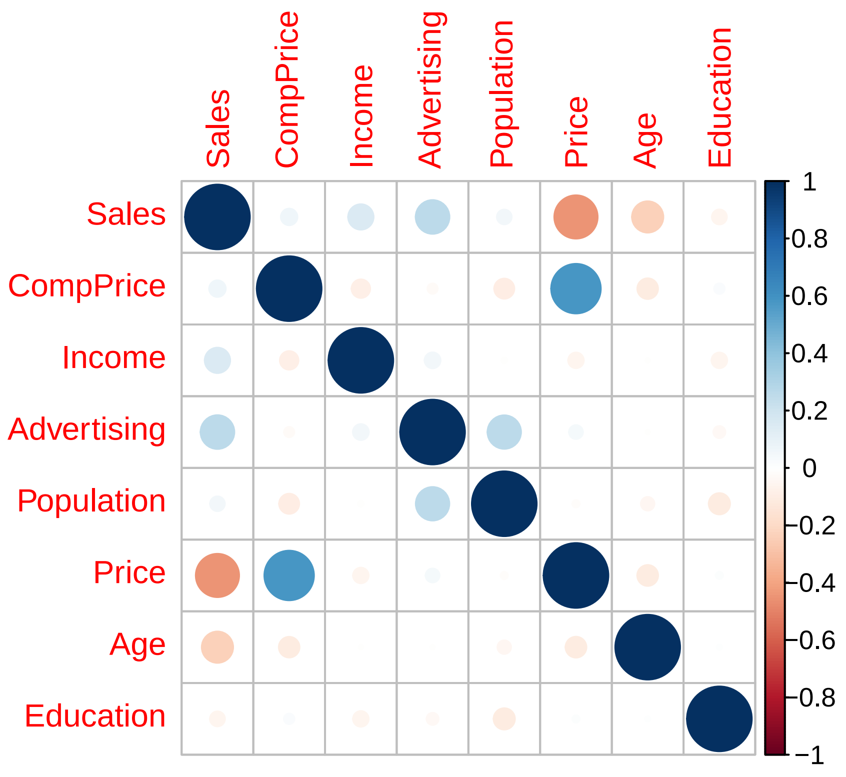
Solution.

```
data(Carseats, package='ISLR')
str(Carseats)
```

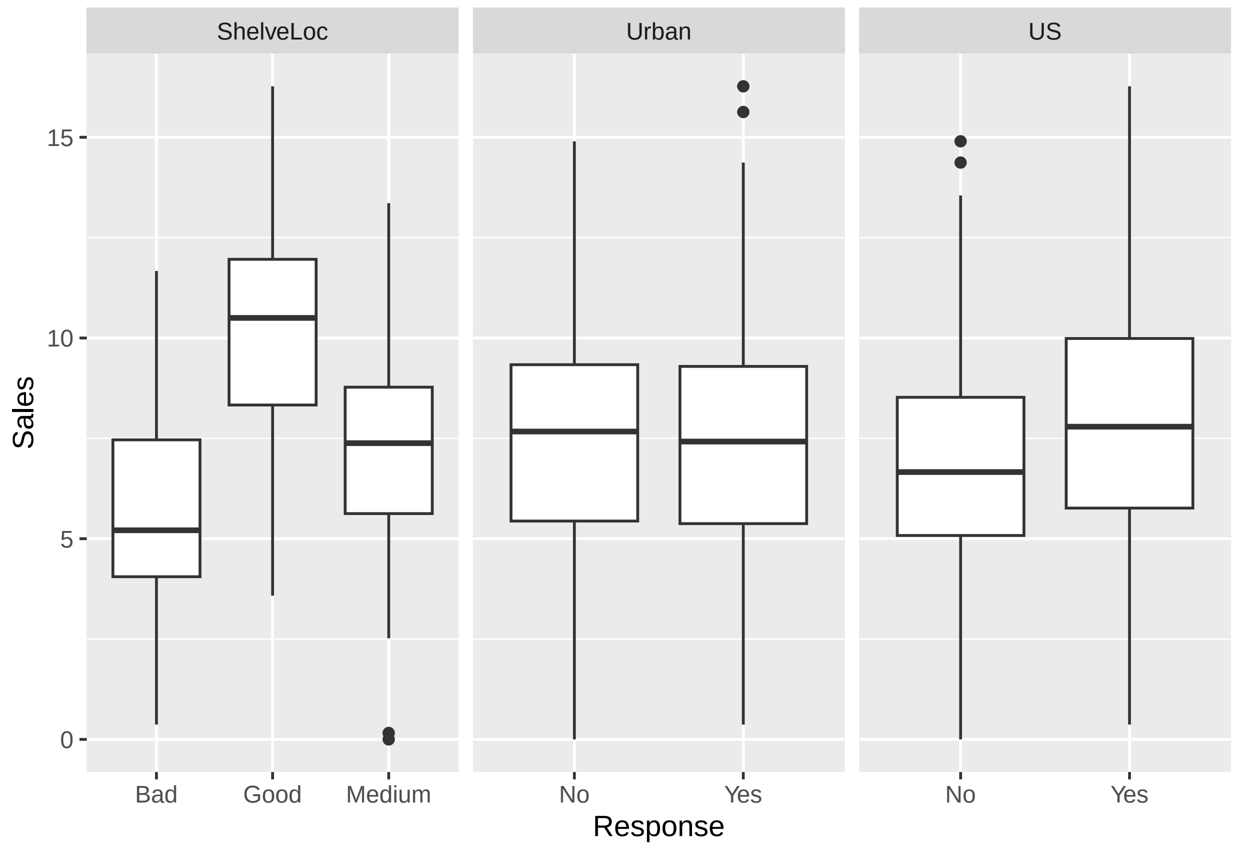
```
## 'data.frame':    400 obs. of  11 variables:
##  $ Sales      : num  9.5 11.22 10.06 7.4 4.15 ...
##  $ CompPrice  : num  138 111 113 117 141 124 115 136 132 132 ...
##  $ Income     : num  73 48 35 100 64 113 105 81 110 113 ...
##  $ Advertising: num  11 16 10 4 3 13 0 15 0 0 ...
##  $ Population : num  276 260 269 466 340 501 45 425 108 131 ...
##  $ Price      : num  120 83 80 97 128 72 108 120 124 124 ...
##  $ ShelveLoc  : Factor w/ 3 levels "Bad","Good","Medium": 1 2 3 3 1 1 3 2 3 3 ...
##  $ Age        : num  42 65 59 55 38 78 71 67 76 76 ...
##  $ Education  : num  17 10 12 14 13 16 15 10 10 17 ...
##  $ Urban      : Factor w/ 2 levels "No","Yes": 2 2 2 2 2 1 2 2 1 1 ...
##  $ US         : Factor w/ 2 levels "No","Yes": 2 2 2 2 1 2 1 2 1 2 ...
```

The set contains 11 variables of which we will be interested in using Sales as a numerical response. I believe one of the best ways to investigate correlation is using `corrplot`. It is shown below. It makes it clear that there is nothing alarming about any correlations between predictors. `Price`, `Advertising`, `Age`, and `Income` all seem to have decent correlation with `Sales` and I expect them in my final model. `CompPrice` and `Price` are correlated, but only about 0.6 in magnitude. `Population` and `Eduction` seem to have the lowest correlation with my response, and I expect they will be less significant. The other variables: `ShelveLoc`, `Urban` and `US` need to be investigated in another manner.

```r
corrplot::corrplot(cor(Carseats%>%select_if(is.numeric)))
```



I show boxplots for the categorical variables.



It seems as though `ShelveLoc` could be influential, with `Urban` and `US` showing little difference between the two factors and `Sales`.

b

Construct a model for the prediction of `Sales` using all other variables as predictors. Based on your analysis above and the resulting model, comment on how you believe the model could be improved.

Solution.

Producing the full model and output its summary information.

```r
lm.test <- lm(Sales~., data=Carseats)
summary(lm.test)
```

```
##
## Call:
## lm(formula = Sales ~ ., data = Carseats)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.8692 -0.6908  0.0211  0.6636  3.4115
##
## Coefficients:
##                    Estimate Std. Error t value Pr(>|t|)
## (Intercept)       5.6606231  0.6034487   9.380  < 2e-16 ***
## CompPrice         0.0928153  0.0041477  22.378  < 2e-16 ***
## Income           0.0158028  0.0018451   8.565 2.58e-16 ***
## Advertising      0.1230951  0.0111237  11.066  < 2e-16 ***
## Population       0.0002079  0.0003705   0.561    0.575
## Price           -0.0953579  0.0026711 -35.700  < 2e-16 ***
## ShelveLocGood    4.8501827  0.1531100  31.678  < 2e-16 ***
## ShelveLocMedium  1.9567148  0.1261056  15.516  < 2e-16 ***
## Age             -0.0460452  0.0031817 -14.472  < 2e-16 ***
## Education       -0.0211018  0.0197205  -1.070    0.285
## UrbanYes         0.1228864  0.1129761   1.088    0.277
## USYes           -0.1840928  0.1498423  -1.229    0.220
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.019 on 388 degrees of freedom
## Multiple R-squared:  0.8734, Adjusted R-squared:  0.8698
## F-statistic: 243.4 on 11 and 388 DF,  p-value: < 2.2e-16
```

After producing the model using all available observations, we can then evaluate predictor significance. It seems clear that our analysis above has guided us well. The predictors `Population`, `Education`, `Urban`, and `US` all indicate a lack of significance, in agreement with our assessment above. We would like to evaluate how the predictor significance will change as we remove the least significance predictor from the model. This sounds like a job for backwards step-wise model building.

c

Fit a backward-stepwise model using `step()` trained using the entire data set. Comment on how the resulting model agrees with our assessment above.

Solution.

We produce a model through `step()` and then evaluate the parameters remaining and their significance.

```r
lm.step <- step(lm.test, k=log(nrow(Carseats)), trace=0)
summary(lm.step)
```

```
##
## Call:
## lm(formula = Sales ~ CompPrice + Income + Advertising + Price +
##     ShelveLoc + Age, data = Carseats)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.7728 -0.6954  0.0282  0.6732  3.3292
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)      5.475226   0.505005   10.84   <2e-16 ***
## CompPrice        0.092571   0.004123   22.45   <2e-16 ***
## Income           0.015785   0.001838    8.59   <2e-16 ***
## Advertising      0.115903   0.007724   15.01   <2e-16 ***
## Price           -0.095319   0.002670  -35.70   <2e-16 ***
## ShelveLocGood    4.835675   0.152499   31.71   <2e-16 ***
## ShelveLocMedium  1.951993   0.125375   15.57   <2e-16 ***
## Age             -0.046128   0.003177  -14.52   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.019 on 392 degrees of freedom
## Multiple R-squared:  0.872,  Adjusted R-squared:  0.8697
## F-statistic: 381.4 on 7 and 392 DF,  p-value: < 2.2e-16
```

The model building continues as expected, with `step()` removing each of the predictors as highlighted in the previous step. We should feel good about our intuition in designing such a model, and could move forward with validation from here. To ensure that this set of predictors is indeed the set that we believe is most significant, we can define a model based on bootstrapping.

d

Using code provided in *Assignment 5 Solutions PDF*, run a bootstrap analysis for parameter inclusion based on backward-stepwise selection using BIC. The final inclusion percentage should be based on at least 1000 bootstrap replicates. Prepare a table with the fraction of time parameters were included.

Solution.

I will run 1000 bootstrap iterations and evaluate parameter inclusion. I cache this simulation as to make it run only once. There are others ways to handle the retention of the coefficients, but I found `merge()` to be an acceptable solution because we need to store the outputs properly, including if variables were not retained in a model, where that value should be retained as `NA` information.

```r
set.seed(10) ### same data splits
Coef.Table <- NULL
samples.n <- nrow(Carseats)
trials = 1000
for(j in 1:trials)
{
  index <- sample(1:samples.n, samples.n, replace=TRUE) ### resamples
  Boot.temp <- Carseats %>% slice(index) ### bootstrap dataframe
  step.test <- step(lm(Sales~., data=Boot.temp), k = log(samples.n), trace=FALSE)
  ifelse(is.null(Coef.Table),
         Coef.Table <- t(as.data.frame(step.test$coefficients)),
         Coef.Table <- merge(Coef.Table, t(as.data.frame(step.test$coefficients)),
                             all=TRUE)
  )
}
```

With the simulation completed, we can calculate the inclusion frequencies using the following apply commands.

```
Perc.Incl<-(trials - apply(apply(Coef.Table, 2, is.na),2,sum))/trials
Perc.Incl
```

```
##     (Intercept)       CompPrice          Income     Advertising           Price
##           1.000           1.000           1.000           1.000           1.000
##   ShelveLocGood ShelveLocMedium             Age       Education        UrbanYes
##           1.000           1.000           1.000           0.106           0.093
##      Population           USYes
##           0.042           0.141
```

We expected these results (although so many coming back 100% of the time is a bit surprising).

e

Based on your work in part (d) decide on a final functional form for your model using a threshold of 90% for parameter inclusion (if it is present in more than 90% of the step models, retain it in your final model).

Solution.

We move forward with a final model that includes each of the 'high inclusion' predictors, specifically those with 90% or higher inclusion over all iterations. Thus, we will work with the model…

$$
\begin{aligned}
y =& \beta_0 + \beta_1 * \text{CompPrice} + \beta_2 * \text{Income} + \beta_3 * \text{Advertising} + \beta_4 * \text{Price} + \\
& \beta_5 * \text{ShelveLocGood} + \beta_6 * \text{ShelveLocMedium} + \beta_7 * \text{Age} + \epsilon, \quad \epsilon \sim N(0, \sigma^2)
\end{aligned}
$$

f

Use 10-repeats of 10-fold cross cross-validation to determine a mean test set MSE.

Solution.

We are asked to implement repeated K-fold cross validation, specifically 10 repeats of 10-fold. We are predicting a numerical response of `Sales` and there does not seem to be any important stratification required, so we will build our folds such that each has an approximately equal number of observations, but we are not concerned about population structure.

I conduct part (g) using the full model in the same loop so that the same fold indexes are used on the both models!

```
set.seed(10)
repeats = 10
Kfolds = 10
test.error.boot <- test.error.full <- numeric()
for(j in 1:repeats){
  folds <- caret::createFolds(Carseats$Sales, k = Kfolds)
  for(i in 1:Kfolds){

    test.index <- folds[[i]] ### fold being tested
    Train <- Carseats %>% slice(-test.index)
    Test <- Carseats %>% slice(test.index)
    lm.boot <- lm(Sales ~ CompPrice + Income + Advertising + Price + ShelveLoc + Age,
                  data=Train)
    lm.full <- lm(Sales ~ ., data=Train)

    boot.test.pred <- predict(lm.boot, Test)
    full.test.pred <- predict(lm.full, Test)

    test.error.boot.calc <- mean((Test$Sales - boot.test.pred)^2)
    test.error.full.calc <- mean((Test$Sales - full.test.pred)^2)

    test.error.boot <- c(test.error.boot, test.error.boot.calc)
    test.error.full <- c(test.error.full, test.error.full.calc)
  }
}
```

With everything stored for further analysis, we complete the remaining subparts for discussion.

g

For a standpoint on how well your bootstrap selected model performs, repeat the same 10-repeats of 10-fold cross-validation using the full model (part (b)).

Solution.

I built this into the work above, cross-validating each model on the same folds at one time.

g

Compare your **final chosen model** to the **full model** basing comparisons on K-fold CV test-set MSE. Discuss what properties have been improved by building the model in this way.

Solution.

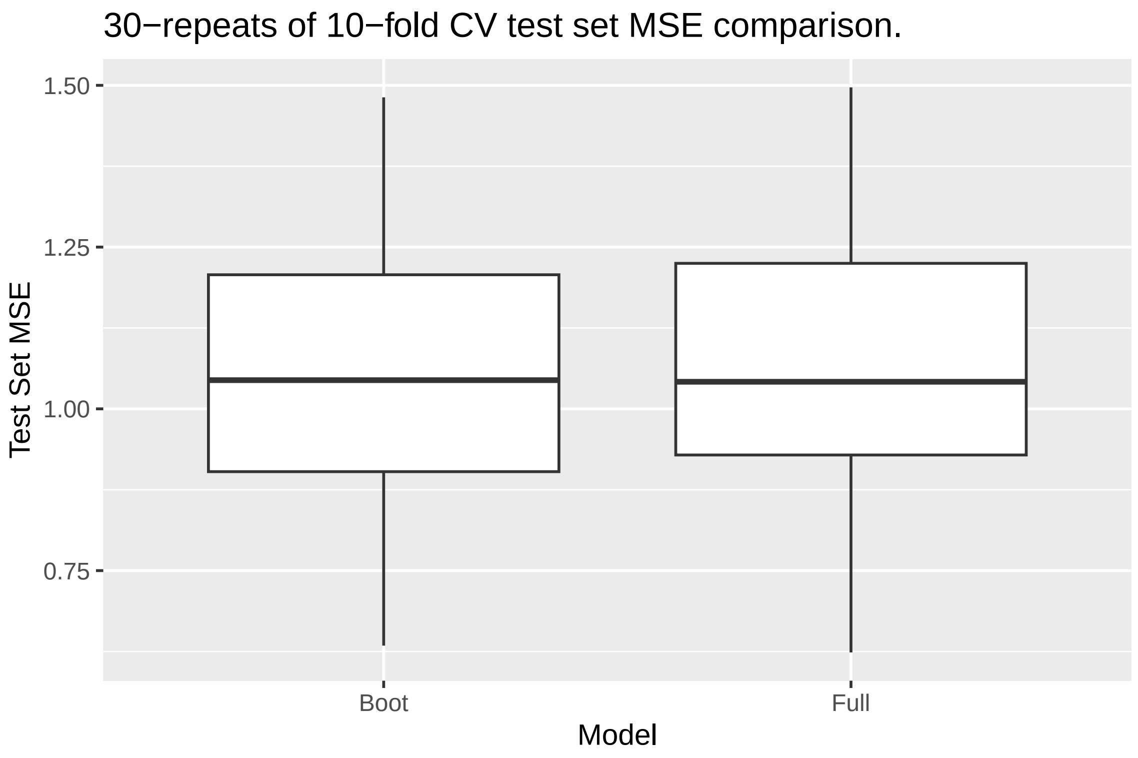
Reworking this problem this year with K-fold cross-validation, we find only a minor increase in the mean test set MSE with the bootstrap selection model compared to the full model. We can start by comparing the mean test set MSE and corresponding standard error.

```
model.compare <- data.frame(Boot = test.error.boot, Full = test.error.full) %>%
  pivot_longer(everything())

model.compare %>%
  group_by(name) %>%
  summarize(Mean = mean(value), SD = sd(value))
```

```
## # A tibble: 2 x 3
##   name   Mean     SD
##   <chr> <dbl>  <dbl>
## 1 Boot   1.061 0.1948
## 2 Full   1.070 0.1980
```

The summary output shows a minor improvement in test set MSE with the bootstrap selection model, returning 1.063 with a standard error of 0.2137, improved from the full model with mean test set MSE of 1.075 and standard error 0.2215. There are negligible improvements here, and visualization of the boxplot indicates a nearly equivalent behavior.

```
ggplot(model.compare, aes(x = name, y = value)) +
  geom_boxplot() + labs(y = "Test Set MSE", x = "Model") +
  labs(title = "30-repeats of 10-fold CV test set MSE comparison.")
```



Hard to tell any differences in the above boxplots. So what was improved using this technique? I was hoping the models would improve on unseen data, but this effect is fairly minimal based on the mean test set error. Instead, interpretability and reduced flexibility are improved, which is really the selection of a more parsimonious model. Although the model performance has not changed significantly, we improved the interpretability by having less covariates to discuss as well as reduced flexibility of the model by removing "nuisance" parameters. A good model selection concept is that when models all test equivalently, choose the most parsimonous model, so we do see parsimony here in the reduced predictor set. These changes do not impact the overall performance of the models greatly, but certainly improved other aspects of using and understanding the models produced. *What other ways could we have built the models to see if any other are capable of improved performance?* Maybe using forward-stepwise bootstrapping would elucidate a new model form?

I was a little underwhelmed by the above so I tried bootstrap inclusion using the forward stepwise strategy. Here are the results.

```
set.seed(10) ### same data splits
Coef.Table <- NULL
samples.n <- nrow(Carseats)
trials = 1000
for(j in 1:trials)
```

```
{
  index <- sample(1:samples.n, samples.n, replace=TRUE) ### resamples
  Boot.temp <- Carseats %>% slice(index) ### bootstrap dataframe
  biggest <- formula(lm(Sales~., data=Boot.temp))
  step.test <- step(lm(Sales~1, data=Boot.temp), direction='forward',
                    k = log(samples.n), scope=biggest, trace=FALSE)
  ifelse(is.null(Coef.Table),
         Coef.Table <- t(as.data.frame(step.test$coefficients)),
         Coef.Table <- merge(Coef.Table, t(as.data.frame(step.test$coefficients)),
                             all=TRUE)
  )
}
```

Model inclusion using forward stepwise selection:

```
Perc.Incl<-(trials - apply(apply(Coef.Table, 2, is.na),2,sum))/trials
Perc.Incl
```

```
##     (Intercept)   ShelveLocGood ShelveLocMedium           Price       CompPrice
##           1.000           1.000           1.000           1.000           1.000
##     Advertising             Age          Income       Education        UrbanYes
##           1.000           1.000           1.000           0.104           0.091
##      Population           USYes
##           0.040           0.140
```

Bootstrapping forward stewpwise selection provides the suggested model

$$
\begin{aligned}
y =& \beta_0 + \beta_1 * \text{ShelveLocGood} + \beta_2 * \text{ShelveLocMedium} + \beta_3 * \text{Price} \ + \beta_4 * \text{CompPrice} + \\
& \beta_5 * \text{Advertising} + \beta_6 * \text{Age} + \beta_7 * \text{Income} + \epsilon, \quad \epsilon \sim N(0, \sigma^2)
\end{aligned}
$$

This is the same model tested from bootstrapping backwards selection just output in a different order. Clearly this model is preferred when using BIC! The largest concern I have over this model is the significant correlation between CompPrice and Price variables (see above corrplot). This could be a job for variance inflation factors to see if I could remove one of these variables from the selection process.

Exercise 3

Produce code that will create at least `1e4` bootstrap replicates of your final model chosen in `Exercise 2`. Your code must save the coefficients from each bootstrap model estimated. Produce a histogram for each coefficient. You are not required to calculate any confidence intervals.

Solution.

I use an old bootstrap function that was written for STA 444/445 that accepts a model and runs a bootstrap analysis outputting a data frame of coefficient estimates. The function is given below.

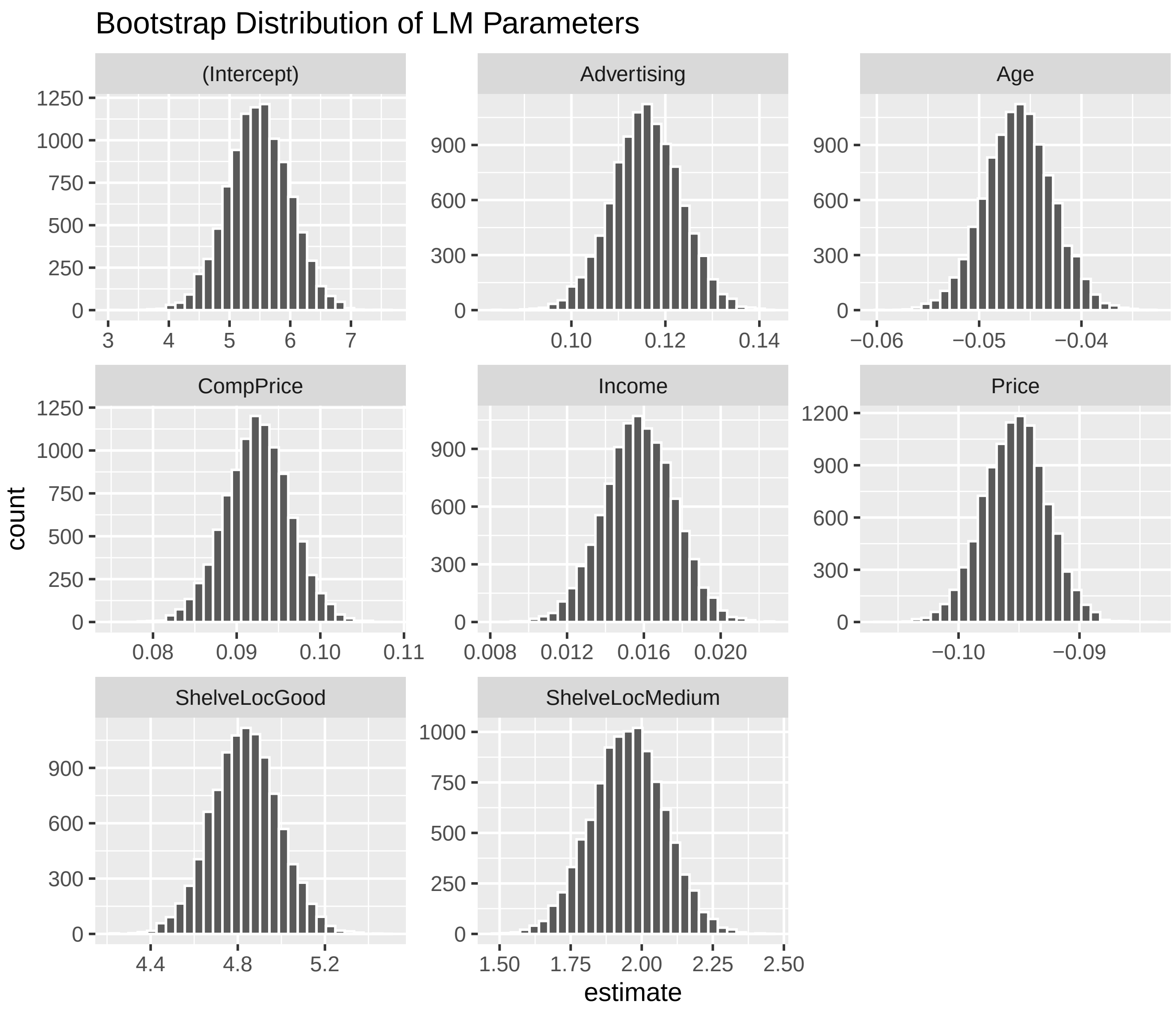
```r
boot.lm <- function(model, N=1000){
  data.full <- model$model  # Extract the original data
  formula <- model$terms  # and model formula used
  # Start the output data frame with the full sample statistic
  output <- broom::tidy(model) %>%
    select(term, estimate) %>%
    pivot_wider(names_from=term, values_from=estimate)

  for( i in 1:N ){
    data <- data.full %>% sample_frac( replace=TRUE )
    model.boot <- lm( formula, data=data)
    coefs <- broom::tidy(model.boot) %>%
      select(term, estimate) %>%
      pivot_wider(names_from=term, values_from=estimate)
    output <- output %>% rbind( coefs )
  }

  return(output)
}
```

Using the bootstrap function, I can then execute the requested `1e4` bootstrap iterations. This is a good time to cache or save the output!

```r
B <- 1e4
# Run the function on a model
m <- lm(Sales ~ CompPrice + Income + Advertising + Price + ShelveLoc + Age, data=Carseats)
boot.dist <- boot.lm(m, B)
```

Wit the bootstrap analysis completed, we output the histograms as a facet wrap.

```r
# If boot.lm() works, then the following produces a nice graph
boot.dist %>% gather('term','estimate') %>%
  ggplot( aes(x=estimate) ) +
  geom_histogram(color="white") +
  facet_wrap(.~term, scales='free') +
  labs(title = "Bootstrap Distribution of LM Parameters")
```



At this point there is many outputs that could be developed. 1) We could use the replicates to determine confidence intervals for the model parameters. 2) We could aggregate the bootstrap estimates to determine a functional model form (Bagging). 3) We could evaluate parameter bias and acceleration. The output looks to confirm well to normality, so some of the more advanced techniques might not be needed, but we could certainly extract a lot of information about the modeling! If time permits, I will include the output of several confidence interval types - no guarantees here!

Exercise 4

We will compute the LOOCV test error estimate using our own code. This can be checked with the `boot` package function `cv.glm()` but be mindful of options within that function (specifically the cost function). I believe it is important that we understand how these things work, and coding is the most practical manner to look inside the *black box*.

a

Load the data `Weekly` from the ISLR package.

Solution.

```r
data(Weekly, package = 'ISLR')
```

b

Fit a logistic regression model that predicts `Direction` using `Lag1` and `Lag2`.

Solution.

```r
Weekly.glm <- glm(Direction ~ Lag1 + Lag2, data=Weekly, family='binomial')
summary(Weekly.glm)
```

```
##
## Call:
## glm(formula = Direction ~ Lag1 + Lag2, family = "binomial", data = Weekly)
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.22122    0.06147   3.599 0.000319 ***
## Lag1        -0.03872    0.02622  -1.477 0.139672
## Lag2         0.06025    0.02655   2.270 0.023232 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1496.2  on 1088  degrees of freedom
## Residual deviance: 1488.2  on 1086  degrees of freedom
## AIC: 1494.2
##
## Number of Fisher Scoring iterations: 4
```

c

Remove the first observation from the `Weekly` data set and fit a new logistic regression model.

Solution.

Performing what is requested. I store the output model rather than display its summary.

```r
Weekly.ws <- Weekly %>% slice(-j)
Model.4c.temp <- glm(Direction ~ Lag1 + Lag2, data=Weekly.ws, family='binomial')
```

d

Using the model from **c.** predict the direction of the first observation. You may use a threshold of 0.5 for the predicted class probability. Ensure you correctly label your prediction. Was your prediction correct?

Solution.

We now need to predict the left-out observation and compare to its known outcome.

```r
### Predict left-out observation returning the "response" (class prob)
prediction.temp <- predict(Model.4c.temp, Weekly[1,], type='response')
### Predict the class based on probability
ifelse(prediction.temp>0.5, 'Up', 'Down')
```

```
##    1
## "Up"
```

```r
### Known class for observation 1
Weekly[1,"Direction"]
```

```
## [1] Down
## Levels: Down Up
```

We predicted *Up* and in reality the direction is *Down.* The way you can determine the class predictions in this case (although you can always look at the help files of these functions as well) is that glm predicts class 1. You can determine which level is Class 1 by looking at the `levels` of a factor. Here, we see that

```r
levels(Weekly$Direction)
```

```
## [1] "Down" "Up"
```

where the ordering of the levels begins at 0. That is, Class 0 is "Down" and Class 1 is "Up".

e

Write a `for`-loop for $i = 1$ to $i = n$ that repeats **c.** and **d.** leaving out each observation sequentially.

Solution.

I produce the loop and save each predicted class to a vector.

```r
trials <- nrow(Weekly)
predictions.loocv <- vector()
for(j in 1:trials)
{
  Weekly.ws <- Weekly %>% slice(-j)
  Model.4c.temp <- glm(Direction ~ Lag1 + Lag2, data=Weekly.ws, family='binomial')
  prediction.temp <- predict(Model.4c.temp, Weekly[j,], type='response')
  predictions.loocv[j] <- ifelse(prediction.temp>0.5, 'Up', 'Down')
}
```

f

Determine the LOOCV test error estimate for the prediction of `Direction` using `Lag1` and `Lag2`.

Solution.

To evaluate the LOOC test set error, we compare the estimated classes to the true classes. We can evaluating the results in a number of ways, I show the confusion matrix.

```r
predictions.loocv <- factor(predictions.loocv)
table(predictions.loocv, Weekly$Direction)
```

```
##
## predictions.loocv Down  Up
##              Down   34  40
##              Up    450 565
```

Then we could calculate classification statistics of the LOOCV analysis. We can also ask how often the elements disagree.

```r
mean(predictions.loocv!=Weekly$Direction)
```

```
## [1] 0.4499541
```

This gives the LOOCV test error as 0.44995. This is slightly better than 50% for a binary problem, but predicting the stock market is about a coin flip.

CANNED CV

Here is how you get the `cv.glm()` function to provide the same result. The function defaults to LOOCV if not provided a K-fold size. The function is not even seem that much faster than our own loop, and the difficulty is providing the correct "Cost" function for proper class predictions.

```r
boot::cv.glm(Weekly, glm(Direction ~ Lag1 + Lag2, family='binomial', data=Weekly),
             function(r, pi = 0) mean(abs(r-pi) > 0.5))$delta[1]
```

```
## [1] 0.4499541
```

If you use default settings, you get garbage because it does not output what you are expecting! That is because the default cost function is the averaged squared error (MSE) - which measures deviance via the link function in this case - but you could easily mistake the below for an error rate!!

```r
boot::cv.glm(Weekly, glm(Direction ~ Lag1 + Lag2, family='binomial', data=Weekly))$delta[1]
```

```
## [1] 0.2464536
```

This is why I suggest using built-in functionality with caution!