Problem Set 2

By Steven Cao *(written using MATLAB R2018a)*

Preliminary Setup

```
% Load the data.
nes_dataset = readtable('nes2008.csv');

% Specify some stuff about the data.
formula = "biden ~ female + age + educ + dem + rep";
outputVar = {'biden'};
inputVar = {'female', 'age', 'educ', 'dem', 'rep'};
```

Problem 1 - Fit a Model

Fit a multiple linear regression model.

```
simpleModel = fitlm(nes_dataset, formula)
```

```
simpleModel =
Linear regression model:
    biden ~ 1 + female + age + educ + dem + rep

Estimated Coefficients:
                   Estimate       SE        tStat       pValue
                   ________    ________    _______    __________

    (Intercept)      58.811      3.1244     18.823    2.6941e-72
    female           4.1032     0.94823     4.3273    1.5926e-05
    age            0.048259    0.028247     1.7084      0.087727
    educ           -0.34533     0.19478     -1.773      0.076406
    dem              15.424       1.068     14.442    8.1449e-45
    rep              -15.85      1.3114    -12.086    2.1573e-32


Number of observations: 1807, Error degrees of freedom: 1801
Root Mean Squared Error: 19.9
R-squared: 0.282,  Adjusted R-Squared 0.28
F-statistic vs. constant model: 141, p-value = 1.5e-126
```

Get the model's mean-squared error (MSE) value.

```
% (Create a custom function to calculate MSE first)
MSE = @(estimatedValues, actualValues, sampleSize) ...
    sum( (estimatedValues-actualValues).^2 ) / sampleSize;

% Now create the function's corresponding arguments:
% (Use input features to generate the model's predicted values)
EVs = predict( simpleModel , nes_dataset(:,inputVar) );
% (Get all rows/observations for the output feature, 'biden')
```

```
AVs = nes_dataset{:,outputVar};
% (Height of the table == number of observations)
SS = height(nes_dataset);

% Calculate the model's MSE.
simpleModelMSE = MSE(EVs, AVs, SS)
```

```
simpleModelMSE = 395.2702
```

The MSE is a number that's not 0, and in an ideal world with perfectly-straight data (and/or with a perfectly-fitting model), the MSE would be 0.

According to the multiple linear regression model, age and years of education do not have a significant effect on predicting how one would feel about Biden. In contrast, whether the person is female, a Democrat, and(?)/or a Republican, are all significant predictors on how one would feel about Biden. Compared to a male, a female is likely to score Biden 4.1 points high on average on the feeling thermometer. Compared to a person that would presumably be an independent (i.e. neither Democrat nor Republican), a Democrat would score Biden 15.4 points higher on average, whereas a Republican would score Biden 15.9 points lower on average.

Problem 2 - Train-Test Split & Fit

Set the seed.

```
s = RandStream('mt19937ar','Seed',123);
RandStream.setGlobalStream(s);
```

Train-test split on the data (50:50).

```
n = height(nes_dataset);
trainingSize = floor(n * 0.50);

trainingIndices = randsample( 1:n , trainingSize , false );
testingIndices  = setdiff( 1:n , trainingIndices );
```

Now fit a linear model to the training portion of the dataset...

```
trainingSet = nes_dataset(trainingIndices,:);
trainedModel = fitlm(trainingSet, formula);
```

...and use it to get the model's MSE on the test set:

```
testingSet = nes_dataset(testingIndices,:);

estimatedValues = predict( trainedModel , testingSet(:,inputVar) );
actualValues = testingSet{:,outputVar};
sampleSize = height(testingSet);

trainingModel_TestMSE = MSE(estimatedValues, actualValues, sampleSize)
trainingModel_TestMSE = 399.0109
```

```
% Compare the two MSE values:
simpleModelMSE - trainingModel_TestMSE
```

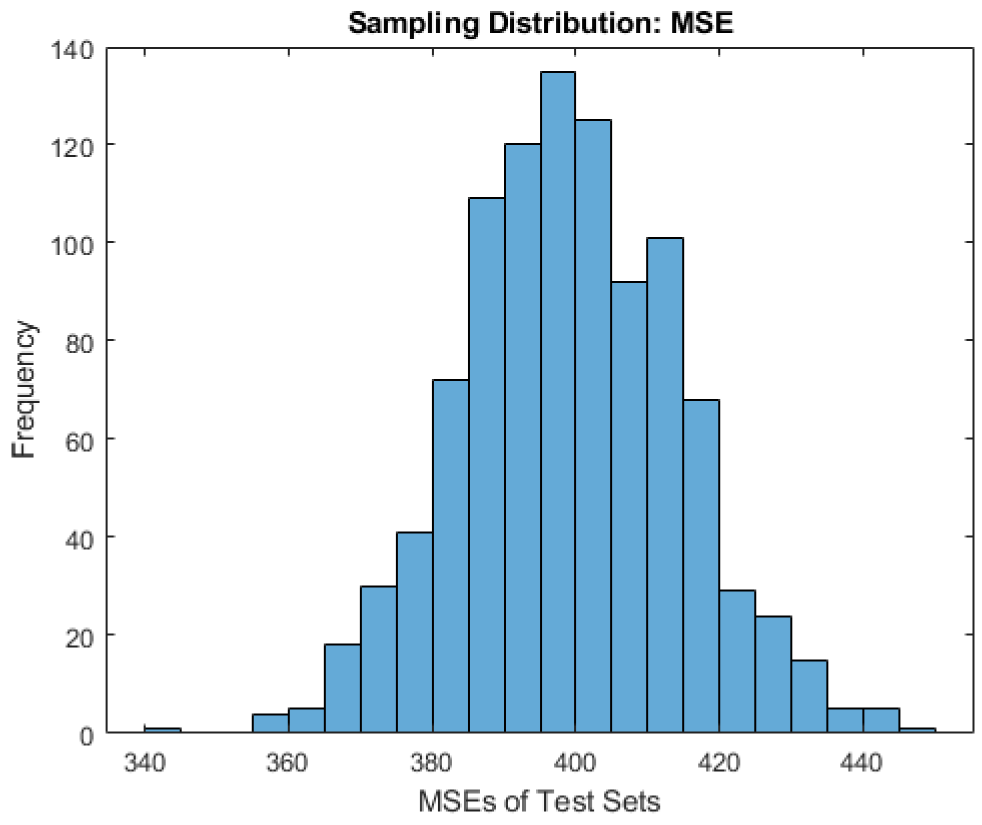
```
ans = -3.7407
```

The model trained on the entire dataset has a lower MSE than the model trained on only the training set (i.e. half of the entire dataset). This makes sense, given that fewer observations means noiser data, which means an increase in MSE.

Problem 3 - Train-Test Split & Fit (1,000 times)

Repeat algorithm from Problem 2 and generate 1000 MSEs.

```
% (Preallocate array)
repeatedModels_TestMSEs = nan(1, 1000);

% Repeat 1000 times
for i = 1:1000

    % Split dataset's indices into training and testing sets.
    trainingIndices = randsample( [1:n] , trainingSize , false );
    testingIndices  = setdiff( [1:n] , trainingIndices );

    % Fit a linear model to the training set.
    trainingSet = nes_dataset(trainingIndices,:);
    trainedModel = fitlm(trainingSet, formula);

    % And get the current model's MSE on the test set.
    testingSet = nes_dataset(testingIndices,:);
    estimatedValues = predict( trainedModel , testingSet(:,inputVar) );
    actualValues = testingSet{:,outputVar};
    sampleSize = height(testingSet);

    repeatedModels_TestMSEs(i) = MSE(estimatedValues, actualValues, sampleSize);

end

figure;
histogram(repeatedModels_TestMSEs);
title("Sampling Distribution: MSE");
xlabel("MSEs of Test Sets");
ylabel("Frequency");
```



```
mean(repeatedModels_TestMSEs)
```

```
ans = 399.0921
```

The test set MSEs generated from the 1,000 models centers around 399.0921, which is very close to the original model's MSE (395.2702), so *overall*, repeating the simple validation set approach seems to do a pretty good job at approximating the parameters of the original model.

Problem 4 - Bootstrapping (1,000 iterations)

Generate a matrix of 1000 rows and 6 columns, representing the 6 beta coefficients for each of the 1000 bootstrapped samples. (6 because the intercept coefficient is included.)

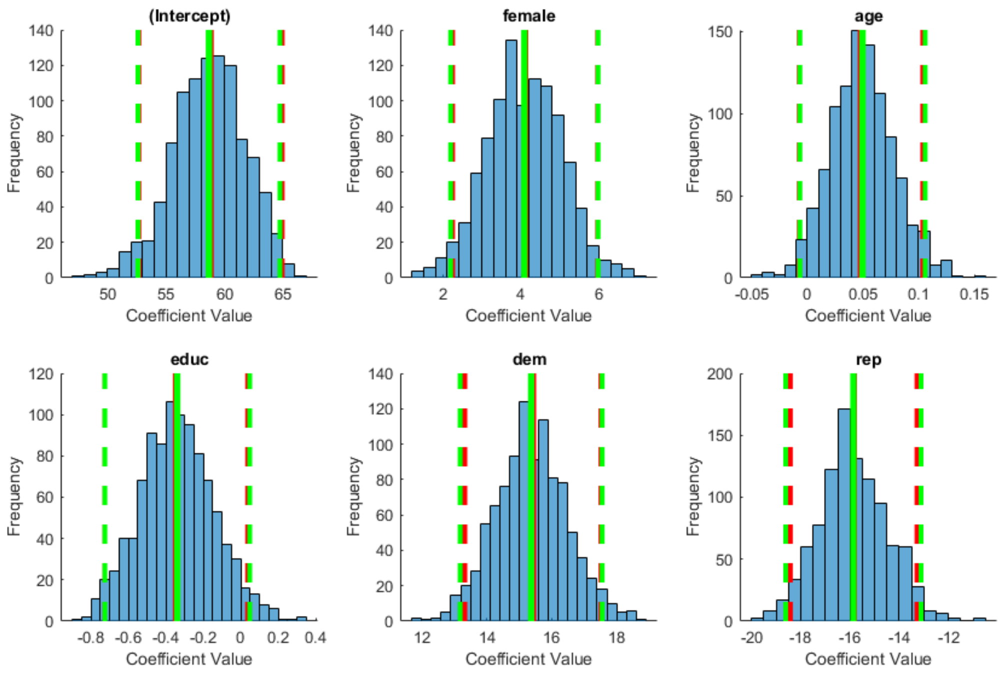
```
% Create another custom function to fetch the beta coefficients of a generated linear model:
get_ModelCoefficients = @(bootstrappedSample, c_formula) ...
    getfield(getfield( fitlm( bootstrappedSample, c_formula{1} ), 'Coefficients'), 'Estimate')
    % Equivalent to calling 'fitlm(sample,model).Coefficients.Estimate' to get a vector of beta

c_formula = {formula}; % needed to deal with a MATLAB-specific limitation
bootstrap_coefficients = bootstrp( 1000, get_ModelCoefficients, nes_dataset, c_formula );

% Get the means for all 6 beta coefficients:
bs_coef_mean = mean(bootstrap_coefficients);
% Get the standard deviations for all 6 beta coefficients:
bs_coef_std = std(bootstrap_coefficients);
```

```
% For easy reference, get the corresponding values from the original model:
sm_coef_mean = simpleModel.Coefficients.Estimate';
sm_coef_std = simpleModel.Coefficients.SE';
```

Plot histograms for the beta coefficients; compare linear model and bootstrap approach side-by-side:

(Red lines are from model, green lines are from bootstrap)

```
figure('Position',[0,0,1000,600]);
for i = 1:6

    subplot(2,3,i);
    hold on;
    % Make histogram + make less ugly
    histogram( bootstrap_coefficients(:,i) );
    if i==1
        title("(Intercept)");
    else
        title( nes_dataset.Properties.VariableNames{i} );
    end
    xlabel("Coefficient Value");
    ylabel("Frequency");
    % Draw some vertical lines, showing mean and CI from original model
    meanPos = sm_coef_mean(i);
    line( [meanPos,meanPos] , ylim , 'Color','red' , 'LineStyle','-' , 'LineWidth',4 );
    leftPos = sm_coef_std(i) * -1.96 + meanPos;
    line( [leftPos,leftPos] , ylim , 'Color','red' , 'LineStyle','--' , 'LineWidth',3 );
    rightPos = sm_coef_std(i) * 1.96 + meanPos;
    line( [rightPos,rightPos] , ylim , 'Color','red' , 'LineStyle','--' , 'LineWidth',3 );
    % Draw the vertical lines from bootstrapping
    meanPos = bs_coef_mean(i);
    line( [meanPos,meanPos] , ylim , 'Color','green' , 'LineStyle','-' , 'LineWidth',4 );
    leftPos = bs_coef_std(i) * -1.96 + meanPos;
    line( [leftPos,leftPos] , ylim , 'Color','green' , 'LineStyle','--' , 'LineWidth',3 );
    rightPos = bs_coef_std(i) * 1.96 + meanPos;
    line( [rightPos,rightPos] , ylim , 'Color','green' , 'LineStyle','--' , 'LineWidth',3 );
    hold off;

end
```



Beta coefficients of the original linear model:

```
sm_coef_mean
```

```
sm_coef_mean = 1×6
   58.8113    4.1032    0.0483   -0.3453   15.4243  -15.8495
```

Average beta coefficients of the bootstrap model:

```
bs_coef_mean
```

```
bs_coef_mean = 1×6
   58.6837    4.0842    0.0497   -0.3395   15.3704  -15.8623
```

SE of beta coefficients of the original linear model:

```
sm_coef_std
```

```
sm_coef_std = 1×6
    3.1244    0.9482    0.0282    0.1948    1.0680    1.3114
```

SE of beta coefficients of the bootstrap model:

```
bs_coef_std
```

```
bs_coef_std = 1×6
    3.0954    0.9651    0.0287    0.1974    1.1150    1.4013
```

Compare SEM of bootstrap coefficients against the linear model's coefficients:

```
bs_coef_std - sm_coef_std
```

```
ans = 1×6
   -0.0290    0.0169    0.0004    0.0026    0.0469    0.0899
```

The beta coefficients generated by the linear model and the bootstrap model are nearly identical to each other. With the standard errors, on the other hand, the bootstrap approach generates higher standard errors across the board (i.e. for all *non-intercept* coefficients) than the linear model. This is in line with what we would expect, since using a parametric approach (i.e. making assumptions about the nature of the underlying data) with valid assumptions makes use of more information than a non-parametric approach, thereby making its estimations more precise (i.e. lower SEs for the model's parameters).

In general, bootstrapping is good for use on data with either very low sample size, or on data whose distributional form is unknown, because it attempts to capture the structure of the data itself (i.e. its "empirical distribution") rather than attempting to capture a certain theoretical distribution (which itself estimates a population distribution in the first place). By resampling the sample and fitting a model many times – as opposed to fitting a model once on the whole dataset – the bootstrapped results avoid fitting against the noise and pecularities of some individual sample, such as the way the data may happen to be arranged. On the other hand, when the data has a large sample size and a follows a distribution, then bootstrapping will yield roughly similar results as fitting a parametric model, albeit with slightly wider confidence intervals.