Pset 2

Ezra Max

February 3, 2020

[16]:
```
#Import packages
import pandas as pd
import numpy as np
from sklearn import linear_model
import matplotlib.pyplot as plt
import statistics
from matplotlib.lines import Line2D

#Set seed
np.random.seed(33002)
```

[17]:
```
#Import linear regression model and NES data
reg=linear_model.LinearRegression(fit_intercept=True)
df=pd.read_csv("/Users/emax/Downloads/problem-set-2-master/nes2008.csv")
```

[18]:
```
#Split data into features and dependent variable
x_train=df[['female','age','educ','dem','rep']]
y_train=df[['biden']]

#Train linear model
mod=reg.fit(x_train,y_train)

#Save and print beta coefficient, store model in Python function
b0=mod.intercept_[0]
b1,b2,b3,b4,b5=mod.coef_[0]
mod_form=("Y(x)=%s+%s*x1+%s*x2+%s*x3+%s*x4+%s*x5"%(round(b0,2),round(b1,2),round
(b2,2),round(b3,2),round(b4,2),round(b5,2)))

def f(x):
    return(b0+b1*x[0]+b2*x[1]+b3*x[2]+b4*x[3]+b5*x[4])
```

[19]:
```
#Predict dependent variable using model and return MSE
y_pred=[f(x) for x in x_train.values]
y_train=[val[0] for val in y_train.values]
n=len(y_pred)

mse_df=pd.DataFrame({'pred':y_pred,'actual':y_train})
```

```
mse_df['square_err']=mse_df.apply(lambda row: (row['pred']-row['actual'])**2,␣
↪axis=1)

mse_full_set=sum(mse_df['square_err'])/n
mse_full_set, mod_form
```

[19]: `(395.2701692786489, 'Y(x)=58.81+4.1*x1+0.05*x2+-0.35*x3+15.42*x4+-15.85*x5')`

Question 1.
Our model trained on the full set is of the form

$$\hat{Y}(x) = 58.81 + 4.10x_1 + .05x_2 - .35x_3 + 15.42x_4 - 15.85x_5 = (58.81, 4.10.05, .35, 15.42, -15.85) \cdot \overline{x} \tag{1}$$

where $\overline{x} = (1, x_1, \ldots, x_5)$

For the model trained on the full dataset, we find that our full-set MSE is 395.27. This MSE is quite high, and as we will explain below this is largely because of irreducible error in the data.

[20]:
```
def run_one_validation():
    '''
    A function to perform one holdout validation.

    Outputs: (MSE value from model applied to testing set, MSE value from model␣
↪applied to training set)
    '''
    #Randomly split dataframe into testing set (~50% of data) and training set␣
↪(~50% of data)
    msk = np.random.rand(len(df)) < 0.5

    train_data=df[msk]
    test_data=df[~msk]

    #Split testing, training data into features and dependent variable
    x_train=train_data[['female','age','educ','dem','rep']]
    y_train=train_data[['biden']]
    x_test=test_data[['female','age','educ','dem','rep']]
    y_test=test_data[['biden']]

    #Train linear model using training data
    mod=reg.fit(x_train,y_train)

    #Save beta coefficient, store model in Python function
    b0=mod.intercept_[0]
    b1,b2,b3,b4,b5=mod.coef_[0]
    mod_form=("Y(x)=%s+%s*x1+%s*x2+%s*x3+%s*x4+%s*x5"%(
    round(b0,2),round(b1,2),round(b2,2),round(b3,2),round(b4,2),round(b5,2)))

    def g(x):
        return(b0+b1*x[0]+b2*x[1]+b3*x[2]+b4*x[3]+b5*x[4])
```

```
    #Use model to predict on testing data
    y_pred_test=[g(x) for x in x_test.values]
    y_actual_test=[val[0] for val in y_test.values]

    #Compute MSE from training and testing data
    n=len(test_data)

    test_mse_df=pd.DataFrame({'pred':y_pred_test,'actual':y_actual_test})
    test_mse_df['square_err']=test_mse_df.apply(lambda row:␣
↪(row['pred']-row['actual'])**2, axis=1)

    testing_mse=sum(test_mse_df['square_err'])/n

    #Use model to predict on training data
    y_pred_train=[g(x) for x in x_train.values]
    y_actual_train=[val[0] for val in y_train.values]

    #Compute MSE from training data
    k=len(train_data)

    train_mse_df=pd.DataFrame({'pred':y_pred_train,'actual':y_actual_train})
    train_mse_df['square_err']=train_mse_df.apply(lambda row:␣
↪(row['pred']-row['actual'])**2, axis=1)

    training_mse=sum(train_mse_df['square_err'])/k

    total_mse=(n*testing_mse+k*training_mse)/(n+k)
    return([b0,b1,b2,b3,b4,b5],testing_mse,training_mse,total_mse)

#Run one holdout validation and record MSE
mse = run_one_validation()
mse
```

[20]:
```
([59.42798444605631,
  4.479168480438875,
  0.054075952307698974,
  -0.3363807360764138,
  14.963578832999122,
  -17.734748782615192],
 438.9405770803064,
 353.181390126255,
 396.17963217405276)
```

Question 2.

We have created a model by fitting a regression on the training data, and then applied this model separately to the testing data and the training data. This has let us compute three different MSE values: the MSE when applying our model to the testing data (VMSE), the MSE when ap-

plying our model to the training data (TMSE), and the MSE from applying our model to both the training and testing sets.

Our values for VMSE, TMSE, and MSE respectively are 438.94, 353.18, 396.18.

When we trained our model on the whole data set above, our MSE was 395.27. Comparing this to our lower TMSE from the holdout model, the most obvious explanation is that our TMSE here is the result of training our model on a smaller set: our regression in the non-holdout case minimized MSE over a large set, while our regression in the holdout case minimized MSE precsiely on the training set (this is our TMSE), and since it is easier to fit a line on less data, our TMSE for the holdout model was lower than our MSE for the non-holdout model.

This analysis is partly true. But I think it only gives part of the picture. For one thing, our model trained on the training set did not look so different from our model trained on the full set. Numerically, we can compare our models $\hat{Y}_1$ and $\hat{Y}_2$ from the non-holdout and holdout approach, respectively, below:

$$\hat{Y}_1(x) = 58.81 + 4.10x_1 + .05x_2 - .35x_3 + 15.42x_4 - 15.85x_5$$
$$\hat{Y}_2(x) = 59.53 + 4.48x_1 + .05x_2 - .34x_3 + 14.96x_4 - 17.73x_5$$

The similarity of the two models suggests that our much lower TMSE (compared to our MSE from the non-holdout approach above) is not just the result of simple overfitting. In fact, it seems that the line of best for our training data in this example is not so far off from the line of best fit for our set overall. Mathematically, that is also what is suggested by comparing our overall MSE from the holdout model (396.17) with our MSE from the non-holdout model (395.27).

If it's not overfitting, then why would we see such a disparity in our holdout TMSE and our non-holdout MSE/holdout VMSE? The other natural explanation is the irreducible error in our data: it's possible that the data in our testing set here was much noisier than the data in our training set. Messiness in the data captured in the full set and the testing set, but not in the training set, could help explain the high MSE in our non-holdout model above and the higher VMSE than TMSE for our holdout model. It's not that we can simply fit a better line on our training data in this case: it's that this particular random selection of training data is less noisy than the rest of the dataset, and that while the lines we would have fit on the training, testing, or full data sets are all similar, this choice of training set is simply easier to fit a linear model on than the testing set (or the full data set.)

If this is the case, then we would expect to see a wide distribution of VMSE scores that's roughly symmetric around its mean–i.e., randomly drawing new training and testing data and training a model on the training data, we would expect to have similar coefficients for each model but a high variance in VMSE (and in turn a high variance in TMSE, since roughly VMSE $2(\text{MSE} - \text{TMSE}/2)$ $2(395.27 - \text{TMSE}/2)$). Keeping in mind as well our point above about overfitting, and assuming our sample is representative, we would also expect our mean VMSE and TMSE scores to be a bit higher than our baseline MSE from the non-holdout model (though again, this should not have too big an effect seeing as our coefficients in the sample above were quite similar to our original coefficients from the non-holdout case). Finally, it's not chance that we got higher VMSE and lower TMSE here–the distribution of noise between the two sets being equal across many trials, we should still expect to see, on average, lower TMSE scores than VMSE scores, since properly our linear models are minimizing TMSE and not VMSE. Thus the discrepency between the two scors here is not solely the result of noisy data but also of where we train our model.

The interesting question is which effect predominates: different lines of fit in our model/bias (reflected in consistently lower TMSE scores and consistently higher VMSE scores) or noisy data/irreducible error (reflected in a more symmetric split of high TMSE/low VMSE cases and

high VMSE/low TMSE cases.) We will argue below looking at the results from a bootstrap that irreducible error predominates, and that it is this irreducible error that explains our high MSEs here rather than model bias.

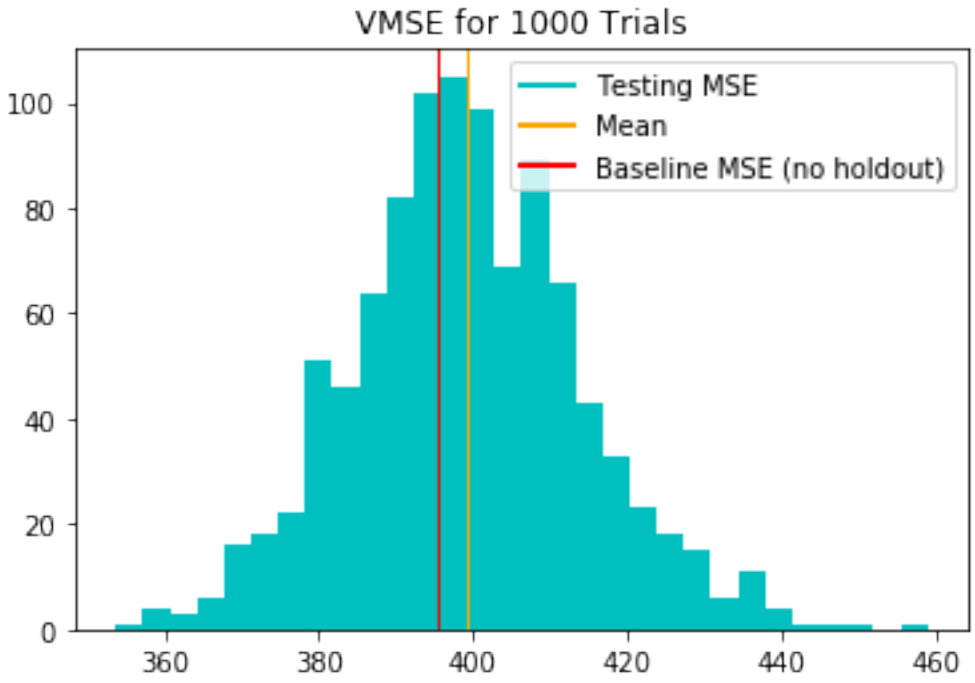
[21]:
```
#Run 1000 validations
mses=[run_one_validation() for i in range(1000)]
mses_test=[val[1] for val in mses]
mses_diff=[val[1]-val[2] for val in mses]
mses_total=[val[3] for val in mses]
avg_mse_holdout=sum(mses_test)/1000
avg_mse_diff=sum(mses_diff)/1000

coeff_list=[val[0] for val in mses]
```

[22]:
```
#Plot validations in histogram against baseline MSE (from full set)
plt.axvline(mse_full_set, color='red', linewidth=1)
plt.axvline(avg_mse_holdout, color='orange', linewidth=1)
plt.hist(mses_test, color='c', bins=30)

custom_lines = [Line2D([0],[0], color='c',lw=2),Line2D([0],[0],␣
 ↪color='orange',lw=2),Line2D([0],[0], color='red',lw=2)]
plt.legend(custom_lines, ['Testing MSE', 'Mean', 'Baseline MSE (no holdout)'])
plt.title('VMSE for 1000 Trials')
```

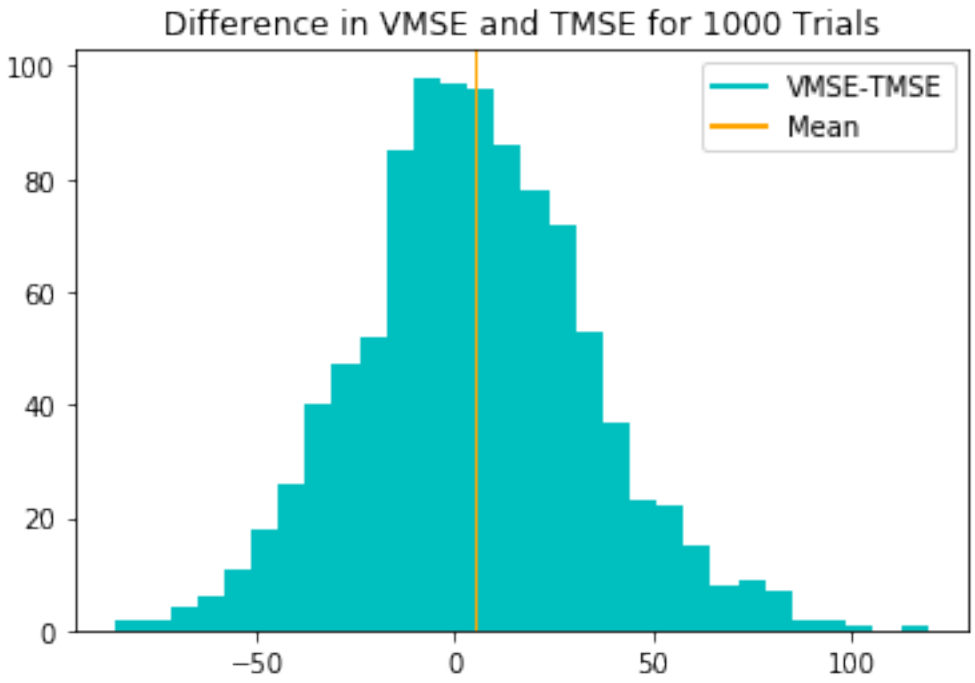
[22]:
```
Text(0.5, 1.0, 'VMSE for 1000 Trials')
```



[23]:
```
#Plot validations in histogram against baseline MSE (from full set)
plt.axvline(avg_mse_diff, color='orange', linewidth=1)
plt.hist(mses_diff, color='c', bins=30)

custom_lines = [Line2D([0],[0], color='c',lw=2),Line2D([0],[0],␣
 ↪color='orange',lw=2)]
plt.legend(custom_lines, ['VMSE-TMSE', 'Mean'])
plt.title('Difference in VMSE and TMSE for 1000 Trials')
```

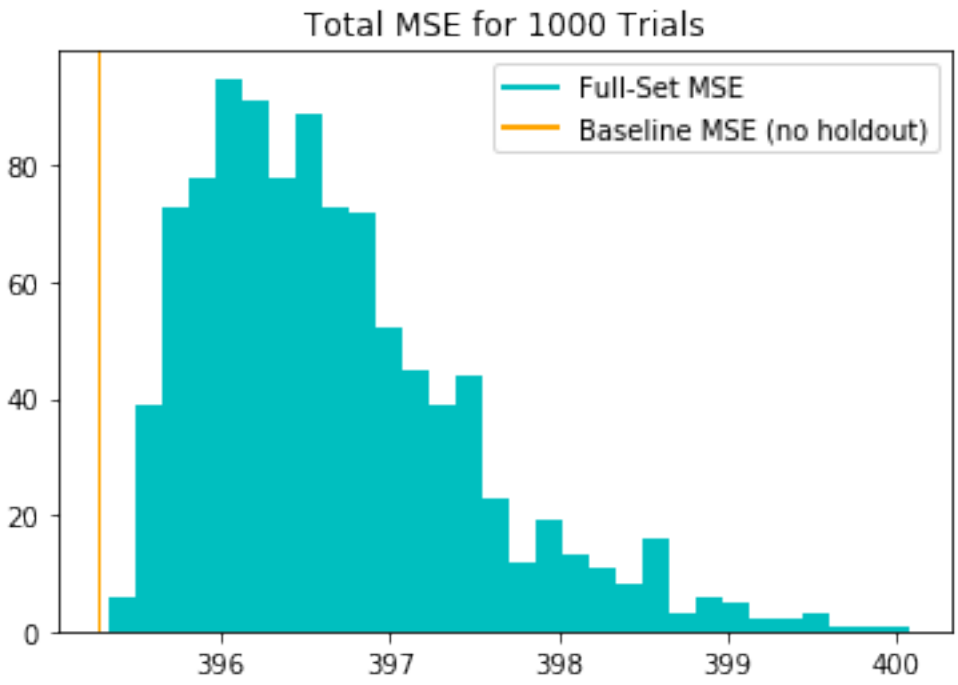
[23]:
```
Text(0.5, 1.0, 'Difference in VMSE and TMSE for 1000 Trials')
```



[24]:
```
#Plot validations in histogram against baseline MSE (from full set)
plt.axvline(mse_full_set, color='orange', linewidth=1)
plt.hist(mses_total, color='c', bins=30)

custom_lines = [Line2D([0],[0], color='c',lw=2),Line2D([0],[0],␣
 ↪color='orange',lw=2)]
plt.legend(custom_lines, ['Full-Set MSE', 'Baseline MSE (no holdout)'])
plt.title('Total MSE for 1000 Trials')
```

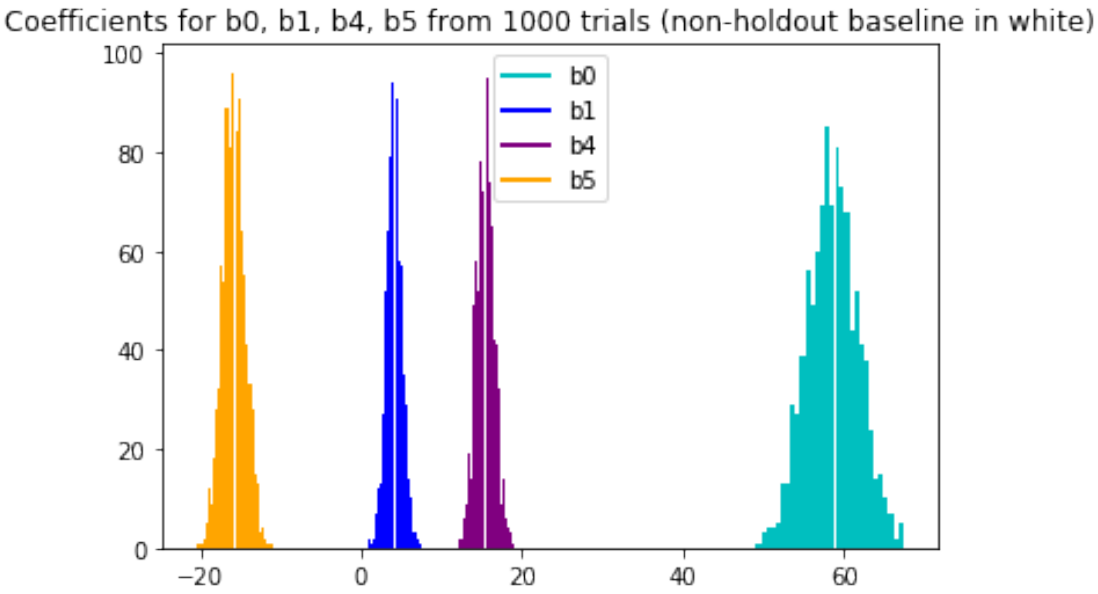
[24]:
```
Text(0.5, 1.0, 'Total MSE for 1000 Trials')
```



[25]:
```
plt.axvline(b0, color='white', linewidth=1)
plt.axvline(b1, color='white', linewidth=1)
plt.axvline(b4, color='white', linewidth=1)
plt.axvline(b5, color='white', linewidth=1)
plt.hist([val[0] for val in coeff_list], color='c', bins=30)
plt.hist([val[1] for val in coeff_list], color='blue', bins=30)
plt.hist([val[4] for val in coeff_list], color='purple', bins=30)
plt.hist([val[5] for val in coeff_list], color='orange', bins=30)

custom_lines = [Line2D([0],[0], color='c',lw=2),Line2D([0],[0],␣
 ↪color='blue',lw=2),Line2D([0],[0], color='purple',lw=2),Line2D([0],[0],␣
 ↪color='orange',lw=2)]
plt.legend(custom_lines, ['b0','b1','b4','b5'])
plt.title('Coefficients for b0, b1, b4, b5 from 1000 trials (non-holdout␣
 ↪baseline in white)')
```

[25]:
```
Text(0.5, 1.0, 'Coefficients for b0, b1, b4, b5 from 1000 trials (non-holdout
baseline in white)')
```

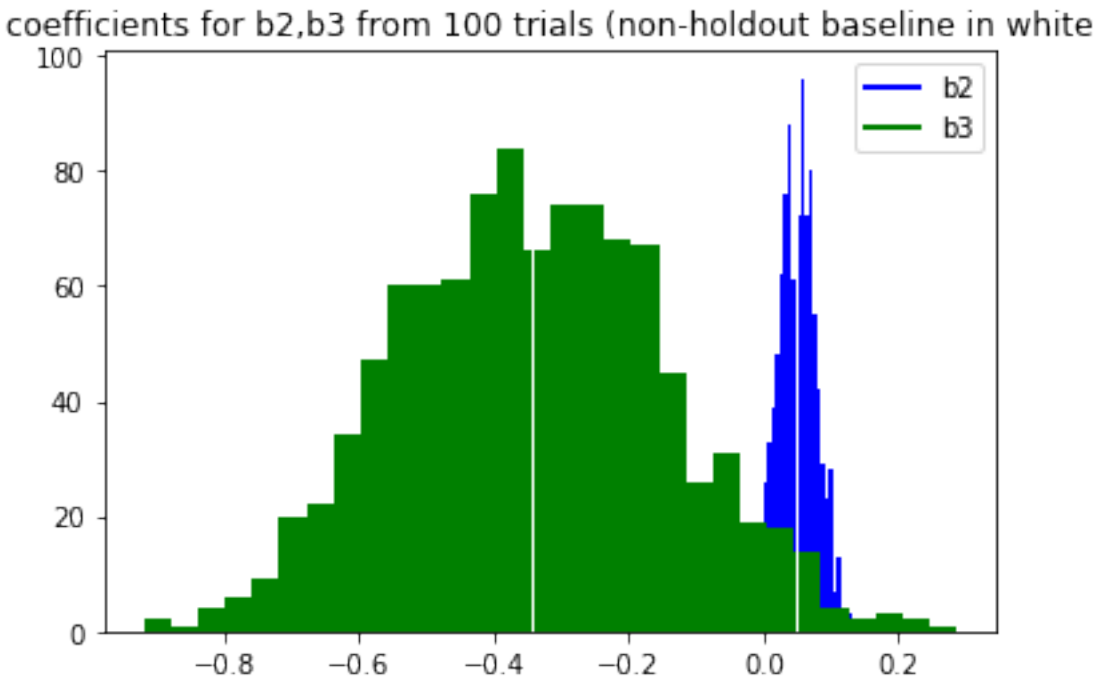


[26]:
```
plt.axvline(b2, color='white', linewidth=1)
plt.axvline(b3, color='white', linewidth=1)

plt.hist([val[2] for val in coeff_list], color='blue', bins=30)
plt.hist([val[3] for val in coeff_list], color='green', bins=30)

custom_lines = [Line2D([0],[0], color='blue',lw=2),Line2D([0],[0],␣
 ↪color='green',lw=2)]
plt.legend(custom_lines, ['b2', 'b3'])
plt.title('coefficients for b2,b3 from 100 trials (non-holdout baseline in␣
 ↪white')
```

[26]:
```
Text(0.5, 1.0, 'coefficients for b2,b3 from 100 trials (non-holdout baseline in
white')
```



Questions 3/4.

We have five histograms above. The first shows the distribution of testing MSEs (VMSEs) from 1000 trials with the holdout approach; the second, the difference VMSE − TMSE between validation and training MSEs from 1000 trials with the holdout approach; the third, the overall MSE from 1000 trials with the holdout approach. The final two show the distributions of different coefficients across the holdout trials, and compare these with the baseline from our original, full-set model.

The first histogram tells us what we predicted above: due to inherent noisiness in the data, there is a wide variation in VMSE from the holdout approach depending on whether the messy observations are sorted into the validation or training sets. The mean of this variation is slight above the MSE we computed with the non-holdout approach, which makes sense because–all noisiness being equal across 1000 trials–we will come out behind when trying to minimize MSE over one set and then compute it for another, as opposed to minimizing it over the entire set. But the fact that we don't come out so far behind on average (about 5 MSE points) suggests that our model is not very biased, but that there's a wide distribution of VMSE scores due to the wide distribution in the assignment of our noisy data to the training and testing sets over 1000 trials.

The second histogram gives us a bit more color: in very loose terms, from the "irreducible noise" effect described in question 2 above–i.e., the fact that one of the validation and training sets always contains more of the "noise" that drives our high MSE numbers–VMSE-TMSE has a roughly symmetric distribution. The distribution's mean tells us whether bias is a large contributor to our high MSEs; if TMSE is often much lower than VMSE, then bias is a likely explanation, whereas if the two statistics are often close, it's likely not bias which is driving the discrepency between TMSE and VMSE or the high MSE numbers overall. Indeed we see by looking at this histogram that our mean for VMSE-TMSE is just above zero, suggesting that bias plays only a small role here: there is not such a difference in the best line we choose across different training/testing

splits, only in how well each dataset can be modeled with any line. Hence TMSE, since it is properly what we are minimizing in training our models, is usually a bit lower than VMSE, but not much: the high MSE for our model is seemingly the result of low bias and high irreducible error.

The third histogram is a good sanity check: the distribution of full-set MSEs from the holdout approach are not so much worse than our baseline MSE from the non-holdout approach. Again, this lends weight to our guess that there are enough clean observations to make the best line of fit similar between trials, but enough noisy observations to affect MSE numbers. More importantly, it is never the case that the holdout MSE for the full set is better than the baseline MSE from the non-holdout model, which is of course what we would want to see (if one of the holdout MSEs were lower than the baseline, this would mean that the model on the whole set was not actually the model that minimized MSE, which would make no sense since by definition the model trained on the whole set is precisely the model which gives the best overall MSE.)

The final two histograms tell us what we had surmised thus far: the lines of best fit/models do not change very much depending on our choice of training/testing data and our model is not so biased; how well that line of best fit fits does change significantly, due to irreducible error in our data. We can further investigate this claim by looking at the distribution/standard deviation of the β values we get from bootstrapping (below.)

[27]:
```
(statistics.stdev([val[0] for val in coeff_list]),
 statistics.stdev([val[1] for val in coeff_list]),
statistics.stdev([val[2] for val in coeff_list]),
statistics.stdev([val[3] for val in coeff_list]),
statistics.stdev([val[4] for val in coeff_list]),
statistics.stdev([val[5] for val in coeff_list]))
```

[27]:
```
(3.1492585130904103,
 0.9690978059484125,
 0.029072531966820882,
 0.19659153085703135,
 1.1120707155829366,
 1.4031386309187128)
```

We have the following table for the standard errors of our coefficients $\beta = (\beta_0, \beta_1, \beta_2, \beta_3, \beta_4, \beta_5)$:

Feature	Coefficient	Standard Deviation
Intercept	β_0	3.15
Female	β_1	.97
Age	β_2	.029
Education	β_3	.20
Demographic	β_4	1.11
Rep	β_5	1.41

Note that our standard deviations are all fairly small. This confirms what the fourth and fifth histograms above told us–that the results from bootstrapping are all quite close to our original model. This is a good sign that we have robust results for our original model. Boostrapped estimators help us validate the accuracy of models on small datasets or datasets with unknown distributions. Since the bootstrapped estimators here are quite similar to our original estimator, we have good evidence that the original model has high MSE, high irreducible error, and low bias.

As the variation in TMSE and VMSE in our booststrap tell us, a lot of the "badness" in our

original MSE was due to the fact that our data could not be easily fit by any line; as the lack of variation in our coefficients/VMSE-TMSE/overall MSEs above tell us, the high MSE did not primarily come from a biased model.

Thus, thanks to bootstrapping we have been able to better separate out bias and noise from what was otherwise a mysteriously high MSE number, and to understand where our model succeeds and fails (despite our small dataset.)