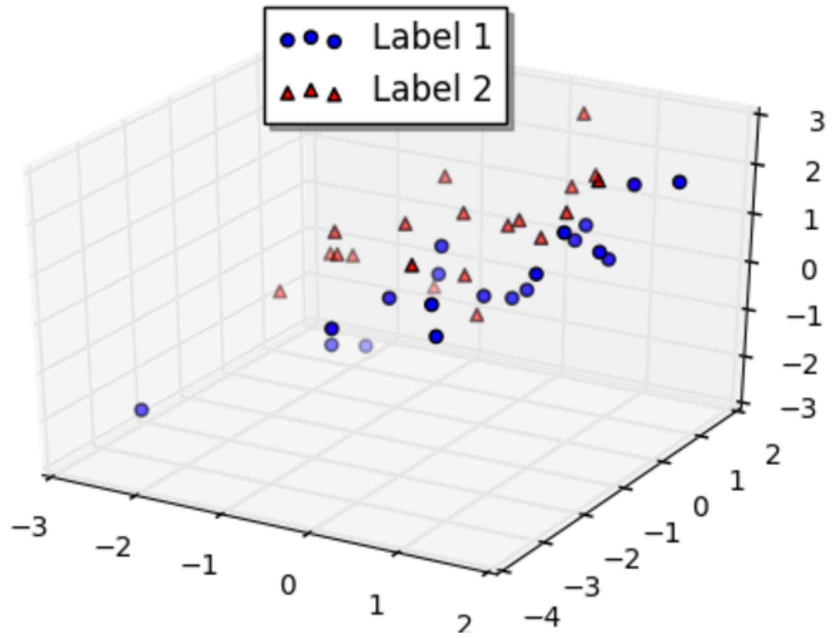
Michael Spearing – MSS3627
Connor Lewis – CSL735

Problem 1: Linear Discriminant Analysis

```
# Problem 1 - Linear Discriminant Analysis
# Question 1
n = 20
d = 3
mean1 = [0,0,0]
cov1 = [[1, 0.9, 0.9], [0.9, 1, 0.9], [0.9, 0.9, 1]]
data1 = np.random.multivariate_normal(mean1, cov1, n)

mean2 = [0, 0, 1]
cov2 = [[1, 0.8, 0.8], [0.8, 1, 0.8], [0.8, 0.8, 1]]
data2 = np.random.multivariate_normal(mean2, cov2, n)

sample_data = [data1, data2]

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(data1[:,0], data1[:,1], data1[:,2], c = 'b', marker='o', label = "Label 1")
ax.scatter(data2[:,0], data2[:,1], data2[:,2], c = 'r', marker='^', label = "Label 2")
legend = ax.legend(loc = 'upper center', shadow = True)
plt.show()
```

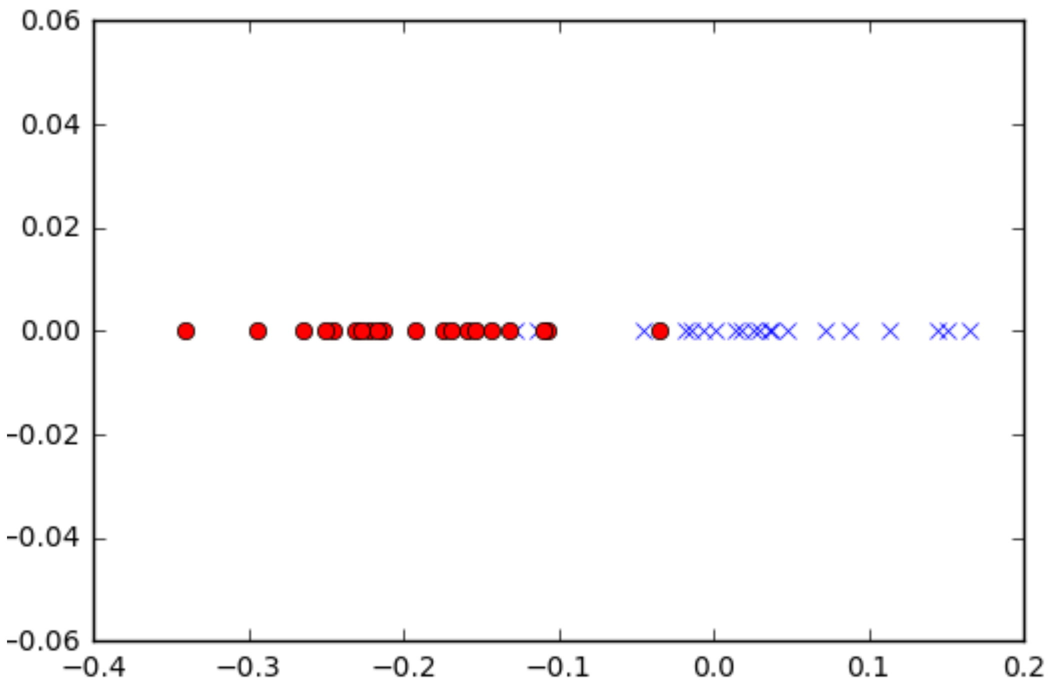


This plot makes sense because it has both distributions that appear to follow the same axis but the points in red, Label 2, are more tightly packed than those in blue, Label 1.

Problem 1. #2
Using Fisher's Linear Discriminant we're able to find w = [0.06773847 0.12832633 -0.17730653]

multiplying the data by this vector we reduce the dimensions of the data to 1 which gives us the plot

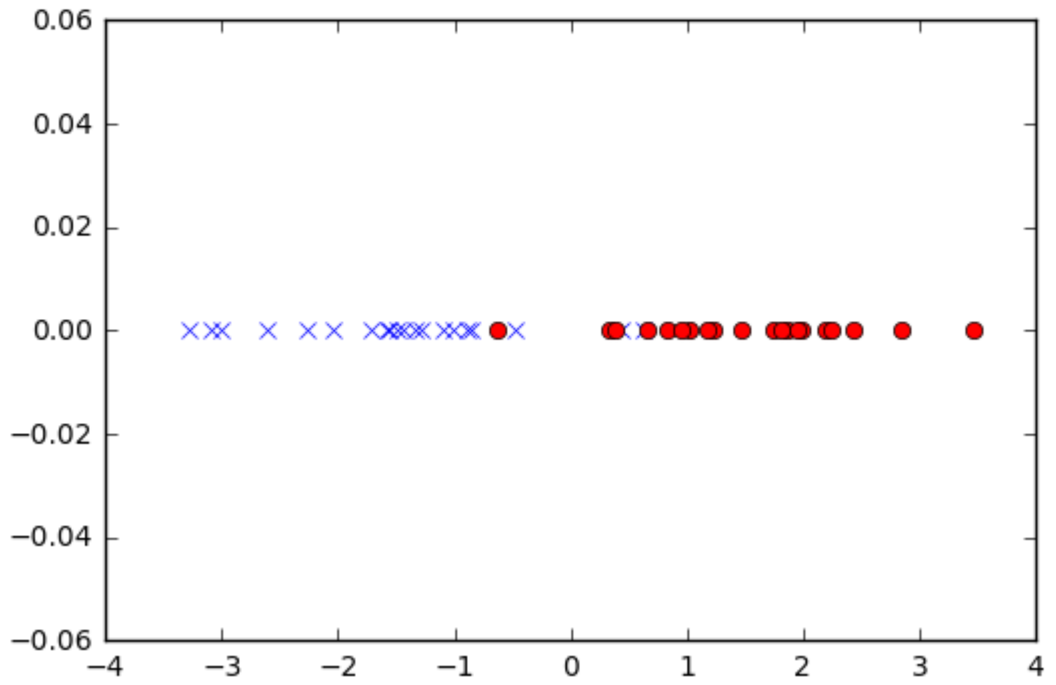
```python
# Problem 1 - Linear Discriminant Analysis
# Question 2: LDA using Fisher's Linear Discriminant
#http://www.csd.uwo.ca/~olga/Courses/CS434a_541a/Lecture8.pdf
mean1 = np.matrix(np.mean(data1, axis = 0))
mean2 = np.matrix(np.mean(data2, axis = 0))
s_1 = len(data1) * np.cov(data1, rowvar = False)
s_2 = len(data2) * np.cov(data2, rowvar = False)
s_w = s_1 + s_2
print(scipy.linalg.inv(s_w))
print(np.linalg.matrix_rank(s_w)) # full rank so can use s_w^-1 * (mean1 - mean2)
mean_diff = mean1 - mean2
w = scipy.linalg.inv(s_w) * np.transpose(mean_diff)
print(w)
new_data1 = np.transpose(w) * np.transpose(data1)
new_data2 = np.transpose(w) * np.transpose(data2)
new_sampledata = [new_data1, new_data2]
fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(new_data1, np.zeros(new_data1.shape[0]), label='Label 1', marker='x', linestyle='')
ax.plot(new_data2, np.zeros(new_data2.shape[0]), c='r', label='Label 2', marker='o', linestyle='')
ax.legend()
plt.show()
```



This plot seems to make sense given the data.

Problem 1. #3

Next we used LDA through sklearn this gave us w = [-2.70953878 -5.1330531 7.09226126] and the plot



This plot is similar to the plot we obtained through Fisher's Linear Discriminant and seems to verify the results that we got through Fisher's process.

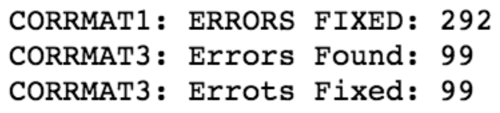
Problem 2: Using Low Rank Structure for Corrupted Entries

First, I plotted each of the features on a scatter plot against a linear index and inspected them. By doing this, it was clear that the value of the corrupted entries was about 10,000 because most of the plots were randomly distributed and then there would be a line of data points at 10,000. By looking at the data, it was easy to see that the actual corrupted value was 9999.

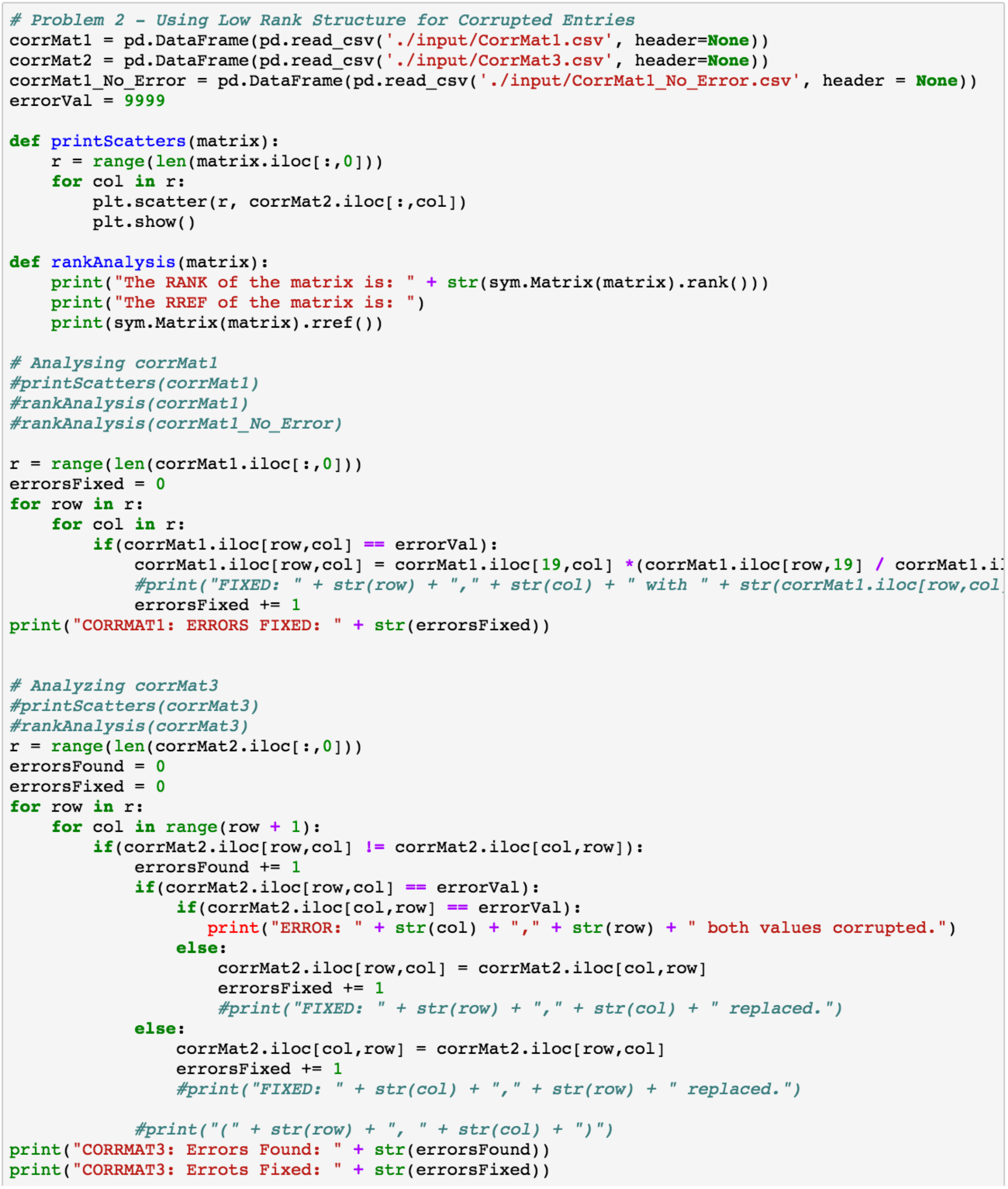
For corrMat1, I first found the rank of the matrix to be 36. This was not helpful. Then, I removed all of the features that contain the corrupted value (9999) and re-ran the rank. This gave a rank of 1 and the RREF was a series of fractions. This indicates that all 75 remaining rows are multiples of each other. I then extrapolated this to the features I had removed to correct the corrupted values. I also ran a similar analysis where I removed all of the rows that contained corrupted entries and the same rank / REEF combination was discovered.

For corrMat3, I first found that the rank of the matrix which is 100. This means that each feature is linearly independent. Then, I calculated the row reduced echelon form of the matrix. This returned the identity matrix which indicates that corrMat3 is symmetrical along its diagonal. With a simple loop, I replaced each value that was not equal to its counterpart with the value that was not the error value (9999). This will recover all of the data points where the entry is corrupted on only one side of the diagonal. If it is corrupted on both sides, or on the diagonal, then it will be remedied. However, because the matrix is linearly independent and symmetric across the diagonal, having one corrupted entry on the diagonal is not a problem because it is still independent. Having more than one would affect the rank of the matrix.

Results from running the algorithm:

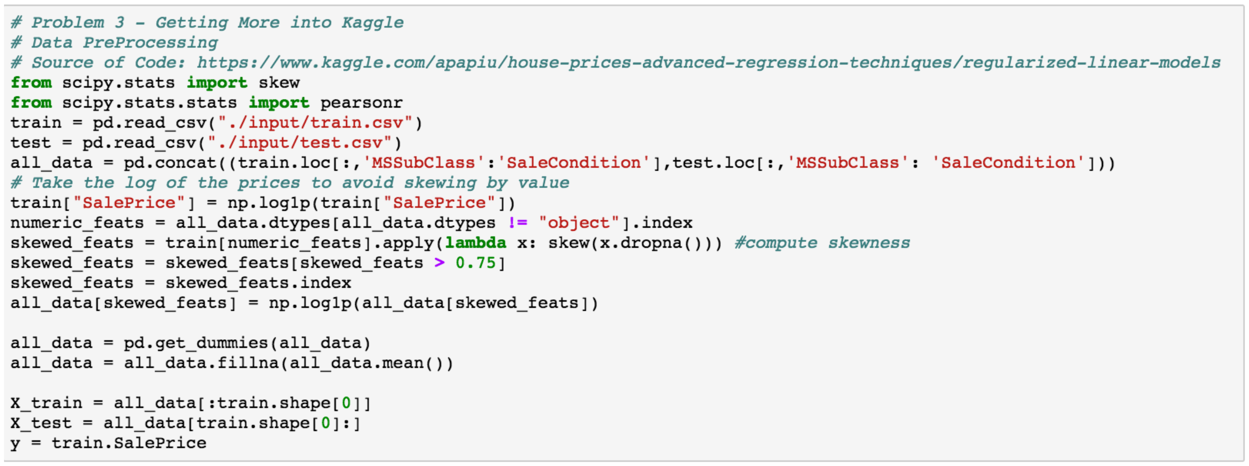


```
CORRMAT1: ERRORS FIXED: 292
CORRMAT3: Errors Found: 99
CORRMAT3: Errots Fixed: 99
```

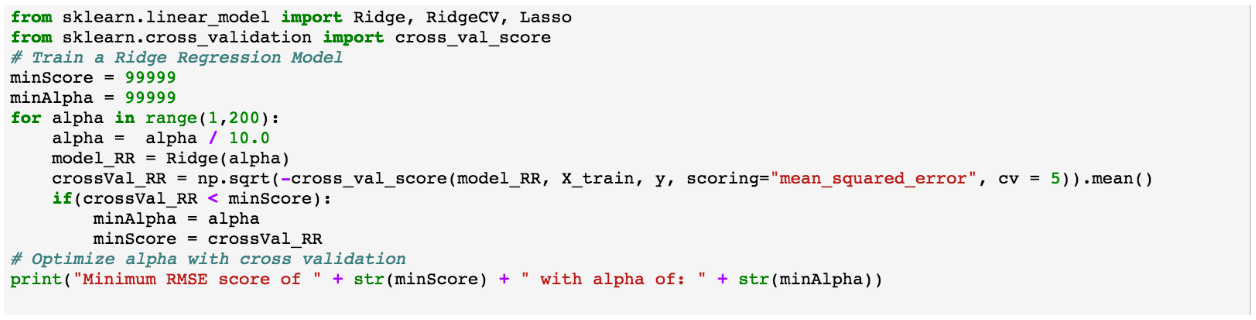


```python
# Problem 2 - Using Low Rank Structure for Corrupted Entries
corrMat1 = pd.DataFrame(pd.read_csv('./input/CorrMat1.csv', header=None))
corrMat2 = pd.DataFrame(pd.read_csv('./input/CorrMat3.csv', header=None))
corrMat1_No_Error = pd.DataFrame(pd.read_csv('./input/CorrMat1_No_Error.csv', header = None))
errorVal = 9999

def printScatters(matrix):
    r = range(len(matrix.iloc[:,0]))
    for col in r:
        plt.scatter(r, corrMat2.iloc[:,col])
        plt.show()

def rankAnalysis(matrix):
    print("The RANK of the matrix is: " + str(sym.Matrix(matrix).rank()))
    print("The RREF of the matrix is: ")
    print(sym.Matrix(matrix).rref())

# Analysing corrMat1
#printScatters(corrMat1)
#rankAnalysis(corrMat1)
#rankAnalysis(corrMat1_No_Error)

r = range(len(corrMat1.iloc[:,0]))
errorsFixed = 0
for row in r:
    for col in r:
        if(corrMat1.iloc[row,col] == errorVal):
            corrMat1.iloc[row,col] = corrMat1.iloc[19,col] *(corrMat1.iloc[row,19] / corrMat1.i
            #print("FIXED:  " + str(row) + "," + str(col) + " with " + str(corrMat1.iloc[row,col
            errorsFixed += 1
print("CORRMAT1: ERRORS FIXED: " + str(errorsFixed))


# Analyzing corrMat3
#printScatters(corrMat3)
#rankAnalysis(corrMat3)
r = range(len(corrMat2.iloc[:,0]))
errorsFound = 0
errorsFixed = 0
for row in r:
    for col in range(row + 1):
        if(corrMat2.iloc[row,col] != corrMat2.iloc[col,row]):
            errorsFound += 1
            if(corrMat2.iloc[row,col] == errorVal):
                if(corrMat2.iloc[col,row] == errorVal):
                   print("ERROR: " + str(col) + "," + str(row) + " both values corrupted.")
                else:
                    corrMat2.iloc[row,col] = corrMat2.iloc[col,row]
                    errorsFixed += 1
                    #print("FIXED:  " + str(row) + "," + str(col) + " replaced.")
            else:
                corrMat2.iloc[col,row] = corrMat2.iloc[row,col]
                errorsFixed += 1
                #print("FIXED:  " + str(col) + "," + str(row) + " replaced.")

            #print("(" + str(row) + ", " + str(col) + ")")
print("CORRMAT3: Errors Found: " + str(errorsFound))
print("CORRMAT3: Errots Fixed: " + str(errorsFixed))
```

Problem 3: Getting more into Kaggle

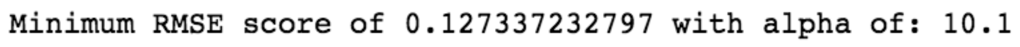


```python
# Problem 3 - Getting More into Kaggle
# Data PreProcessing
# Source of Code: https://www.kaggle.com/apapiu/house-prices-advanced-regression-techniques/regularized-linear-models
from scipy.stats import skew
from scipy.stats.stats import pearsonr
train = pd.read_csv("./input/train.csv")
test = pd.read_csv("./input/test.csv")
all_data = pd.concat((train.loc[:,'MSSubClass':'SaleCondition'],test.loc[:,'MSSubClass': 'SaleCondition']))
# Take the log of the prices to avoid skewing by value
train["SalePrice"] = np.log1p(train["SalePrice"])
numeric_feats = all_data.dtypes[all_data.dtypes != "object"].index
skewed_feats = train[numeric_feats].apply(lambda x: skew(x.dropna())) #compute skewness
skewed_feats = skewed_feats[skewed_feats > 0.75]
skewed_feats = skewed_feats.index
all_data[skewed_feats] = np.log1p(all_data[skewed_feats])

all_data = pd.get_dummies(all_data)
all_data = all_data.fillna(all_data.mean())

X_train = all_data[:train.shape[0]]
X_test = all_data[train.shape[0]:]
y = train.SalePrice
```

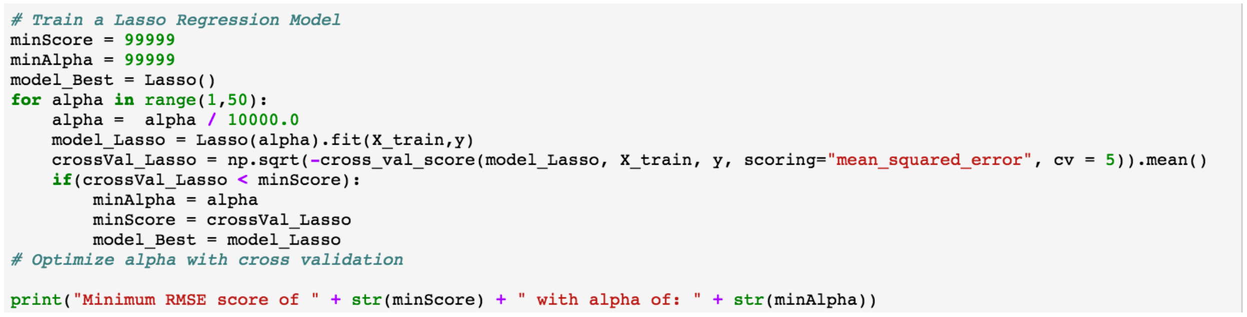


```python
from sklearn.linear_model import Ridge, RidgeCV, Lasso
from sklearn.cross_validation import cross_val_score
# Train a Ridge Regression Model
minScore = 99999
minAlpha = 99999
for alpha in range(1,200):
    alpha =  alpha / 10.0
    model_RR = Ridge(alpha)
    crossVal_RR = np.sqrt(-cross_val_score(model_RR, X_train, y, scoring="mean_squared_error", cv = 5)).mean()
    if(crossVal_RR < minScore):
        minAlpha = alpha
        minScore = crossVal_RR
# Optimize alpha with cross validation
print("Minimum RMSE score of " + str(minScore) + " with alpha of: " + str(minAlpha))
```

The best score from Ridge Regression and results of RMSE cross validation:

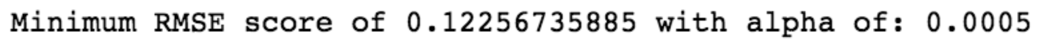


```
Minimum RMSE score of 0.127337232797 with alpha of: 10.1
```

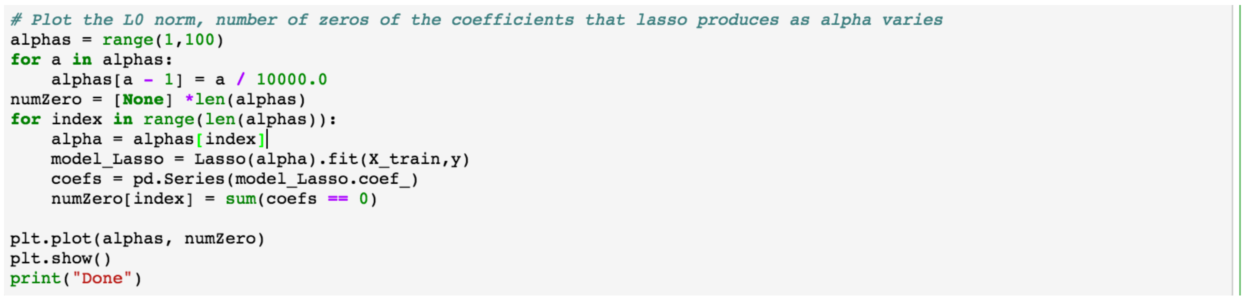


```python
# Train a Lasso Regression Model
minScore = 99999
minAlpha = 99999
model_Best = Lasso()
for alpha in range(1,50):
    alpha =  alpha / 10000.0
    model_Lasso = Lasso(alpha).fit(X_train,y)
    crossVal_Lasso = np.sqrt(-cross_val_score(model_Lasso, X_train, y, scoring="mean_squared_error", cv = 5)).mean()
    if(crossVal_Lasso < minScore):
        minAlpha = alpha
        minScore = crossVal_Lasso
        model_Best = model_Lasso
# Optimize alpha with cross validation

print("Minimum RMSE score of " + str(minScore) + " with alpha of: " + str(minAlpha))
```

The best score from Lasso Regression and results of RMSE cross validation:

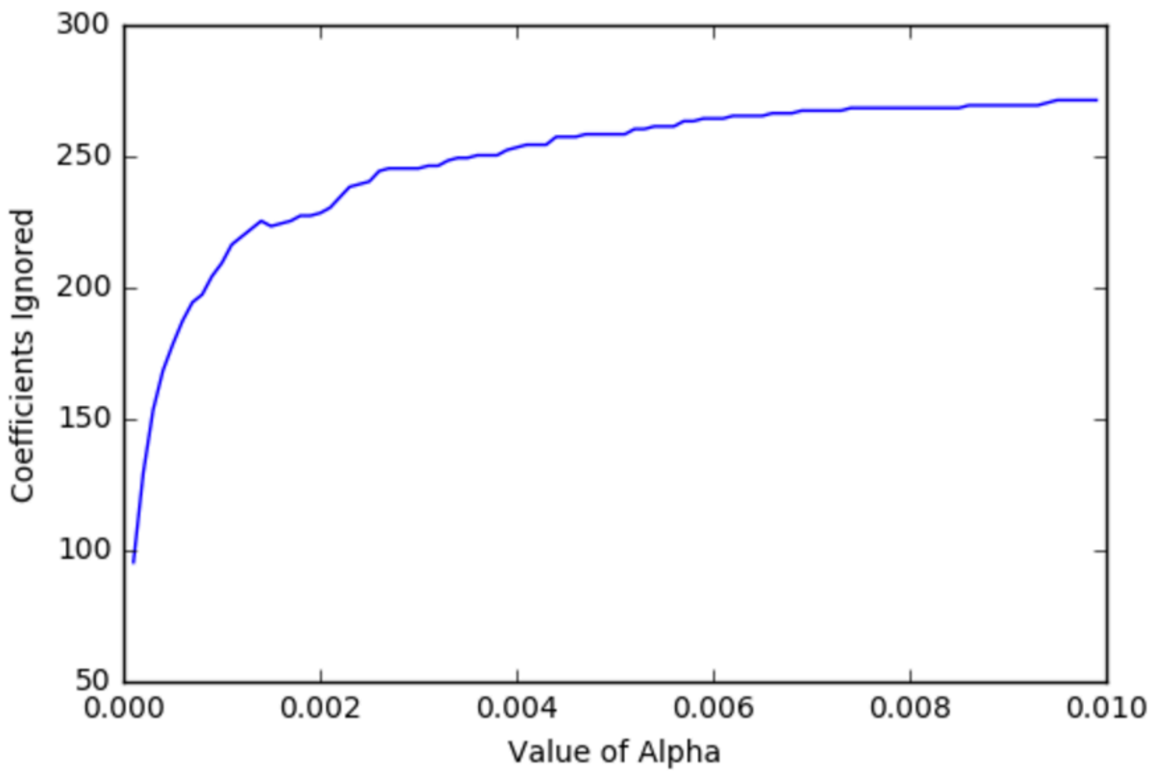


```
Minimum RMSE score of 0.12256735885 with alpha of: 0.0005
```



```
# Plot the L0 norm, number of zeros of the coefficients that lasso produces as alpha varies
alphas = range(1,100)
for a in alphas:
    alphas[a - 1] = a / 10000.0
numZero = [None] *len(alphas)
for index in range(len(alphas)):
    alpha = alphas[index]
    model_Lasso = Lasso(alpha).fit(X_train,y)
    coefs = pd.Series(model_Lasso.coef_)
    numZero[index] = sum(coefs == 0)

plt.plot(alphas, numZero)
plt.show()
print("Done")
```

Plot of coefficients equal to zero as alpha varies:


Coefficients Ignored
Value of Alpha