Name: Hootan Hosseinzadeganbushehri

[redacted]

UCI Email: hootanh@uci.edu

Problem 1 - Part 1:

In [1]:
```python
import numpy, mltools

data = numpy.genfromtxt("data/curve80.txt")

X = data[:, 0]
X = numpy.atleast_2d(X).T
Y = data[:, 1]
Xtr, Xte, Ytr, Yte = mltools.splitData(X, Y, 0.75)

print()
print("\033[1m" + "Training data for feature value X:", Xtr.shape, '\n')
print("\033[1m" + "Testing data for feature value X:", Xte.shape, '\n')
print("\033[1m" + "Training data for target value Y:", Ytr.shape, '\n')
print("\033[1m" + "Testing data for target value Y:", Yte.shape, '\n')
```

Training data for feature value X: (60, 1)

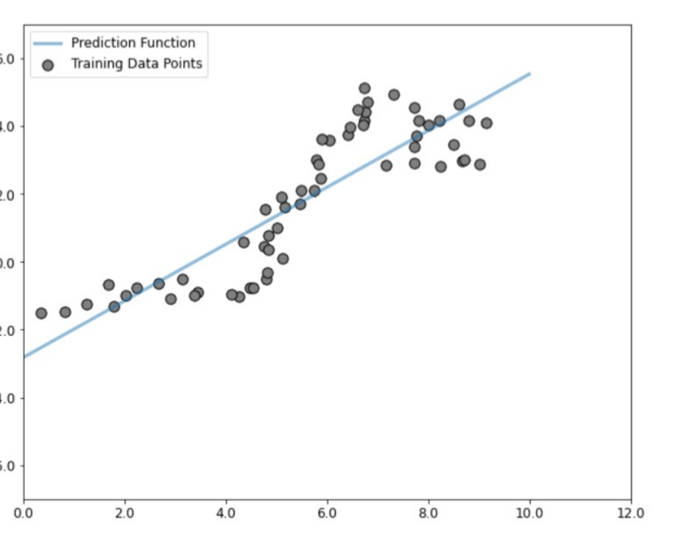
Testing data for feature value X: (20, 1)

Training data for target value Y: (60,)

Testing data for target value Y: (20,)

Problem 1 - Part 2(a):

In [2]:
```python
import matplotlib.pyplot as plt

lr = mltools.linear.linearRegress(Xtr, Ytr)
xs = numpy.linspace(0, 10, 200)
xs = xs[:, numpy.newaxis]
ys = lr.predict(xs)

figure, axes = plt.subplots(1, 1, figsize = (10, 8))
axes.scatter(Xtr, Ytr, s = 90, color = "gray", edgecolors = "black", label = "Training Data Points")
axes.plot(xs, ys, lw = 3, alpha = 0.5, label = "Prediction Function")
axes.set_xlim(0.0, 12.0)
axes.set_ylim(-7.0, 7.0)
axes.set_xticklabels(axes.get_xticks(), fontsize = 12)
axes.set_yticklabels(axes.get_yticks(), fontsize = 12)
axes.legend(fontsize = 12, loc = 2)
print()
plt.show()
print()
```



Problem 1 - Part 2(b):

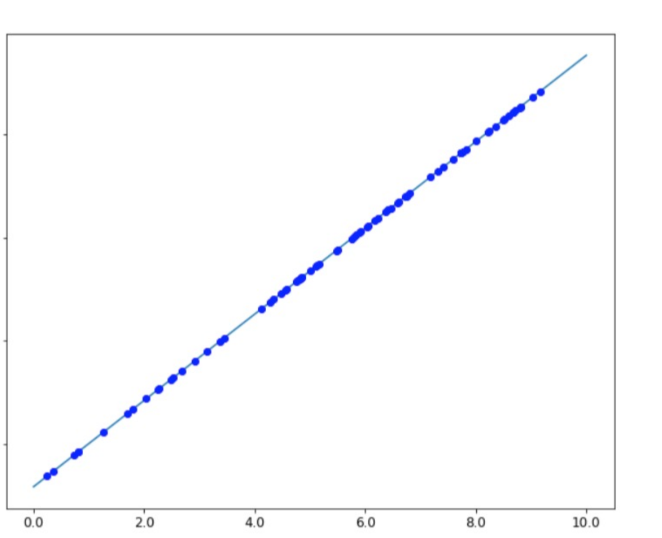
In [3]:
```python
print()
print("\033[1m" + "Linear regression coefficients:", lr.theta)
```

Linear regression coefficients: [[-2.82765049 0.83606916]]

Here theta 0 is -2.82765049 which is approximately -2.8 and theta 0 is very close and similar to the y-intercept. Also, theta 1 is 0.83606916 and the linear regression line has the slope of theta 1. Therefore, by using y(x) = theta 0 + theta 1 x I can just simply make x = 0 and get y(0) = -2.82765049 + 0.83606916 (0) = -2.82765049 which is approximately -2.8 and very close and similar to the y-intercept as mentioned before and can be used as a verification that the linear regression coefficients and my plot match. In addition, The graph below is showning that my plot line is matching the linear regression line completely.

In [4]:
```python
figure, axes = plt.subplots(1, 1, figsize = (10, 8))
axes.plot(xs, ys, X, -2.82765049 + 0.83606916 * X, "bo")
axes.set_xticklabels(axes.get_xticks(), fontsize = 12)
axes.set_yticklabels(axes.get_yticks(), fontsize = 12)
print()
print("\033[1m" + "This figure verifies that my plot and linear regression coefficients match.")
print("\033[1m" + "By using y(x) = theta 0 + theta 1 x with theta 0 = -2.82765049 and theta 1 = 0.83606916")
print()
plt.show()
print()
```

This figure verifies that my plot and linear regression coefficients match.
By using y(x) = theta 0 + theta 1 x with theta 0 = -2.82765049 and theta 1 = 0.83606916



Problem 1 - Part 2(c):

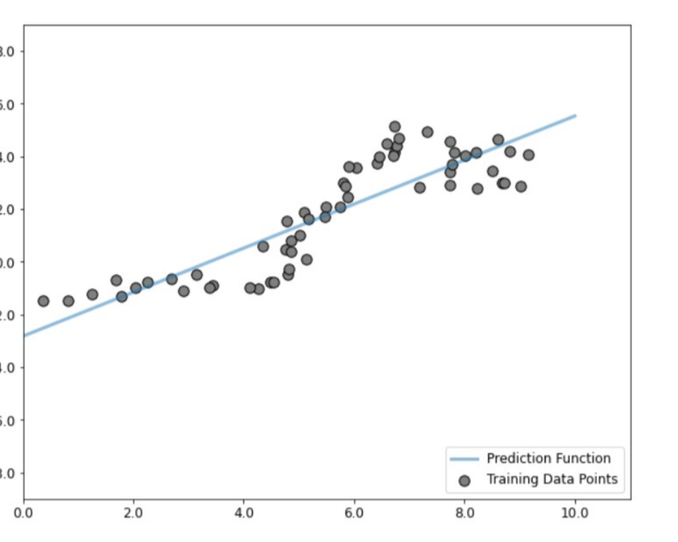
In [5]:
```python
print()
print("\033[1m" + "Mean squared error of the predictions on the training data:", lr.mse(Xtr, Ytr))
print("\033[1m" + "Mean squared error of the predictions on the test data:", lr.mse(Xte, Yte))
print()
```

Mean squared error of the predictions on the training data: 1.127711955609391
Mean squared error of the predictions on the test data: 2.2423492030101246

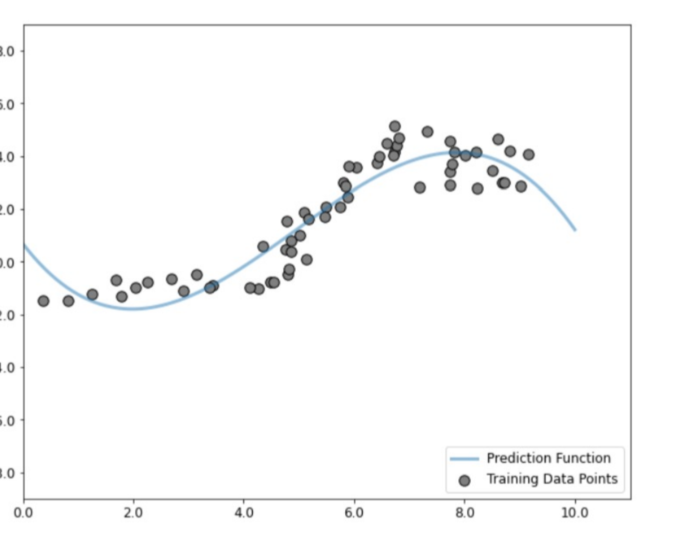
Problem 1 - Part 3(a):

In [6]:
```python
Xtr2 = numpy.zeros((Xtr.shape[0], 2))
Xtr2[:, 0] = Xtr[:, 0]
Xtr2[:, 1] = Xtr[:, 0] ** 2

d = [1, 3, 5, 7, 10, 15, 18]

for n in range(len(d)):
    XtrP = mltools.transforms.fpoly(Xtr, d[n], bias = False)
    XtrP,params = mltools.transforms.rescale(XtrP)
    lr = mltools.linear.linearRegress(XtrP, Ytr)
    figure, axes = plt.subplots(1, 1, figsize = (10, 8))
    xs = numpy.linspace(0, 10, 100)
    xs = xs[:,numpy.newaxis]
    xsP = mltools.transforms.fpoly(xs, d[n], False)
    xsP,_ = mltools.transforms.rescale(xsP, params)
    ys = lr.predict(xsP)
    axes.scatter(Xtr, Ytr, s = 90, color = "gray", edgecolors = "black", label = "Training Data Points")
    axes.plot(xs, ys, lw = 3, alpha = 0.5, label = "Prediction Function")
    axes.set_xlim(0, 11)
    axes.set_ylim(-9, 9)
    axes.set_xticklabels(axes.get_xticks(), fontsize = 12)
    axes.set_yticklabels(axes.get_yticks(), fontsize = 12)
    axes.legend(fontsize = 12, loc = 4)
    print()
    print("\033[1m" + "Polynomial regression model of degree:", d[n], "\n")
    plt.show()
    print()
```

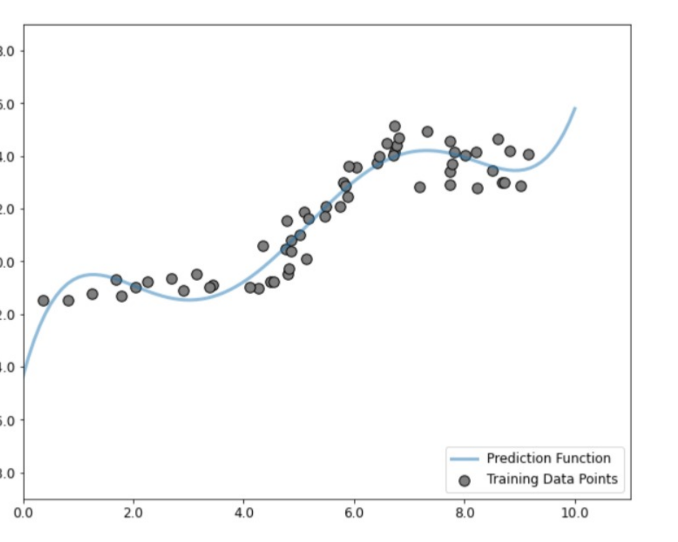
Polynomial regression model of degree: 1



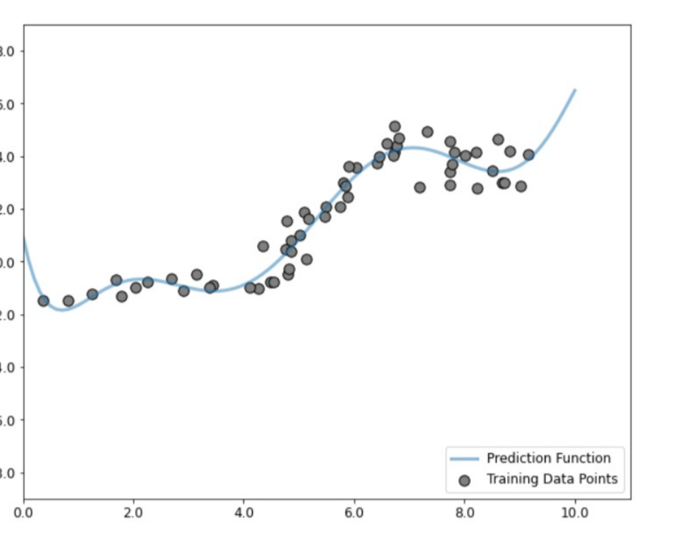
Polynomial regression model of degree: 3



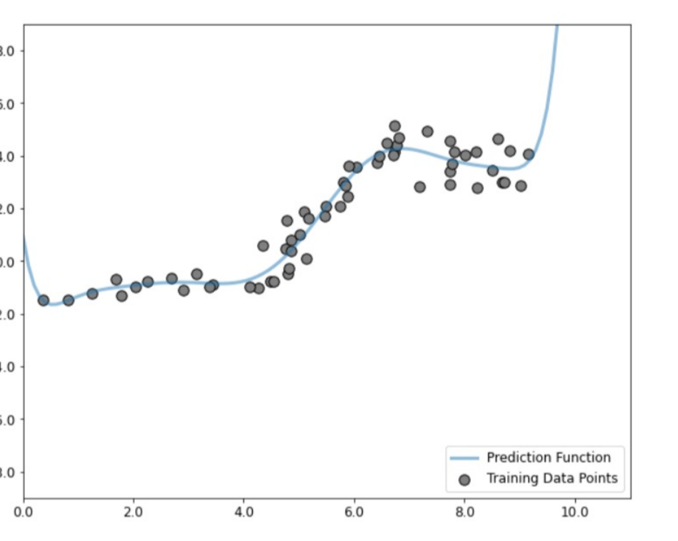
Polynomial regression model of degree: 5



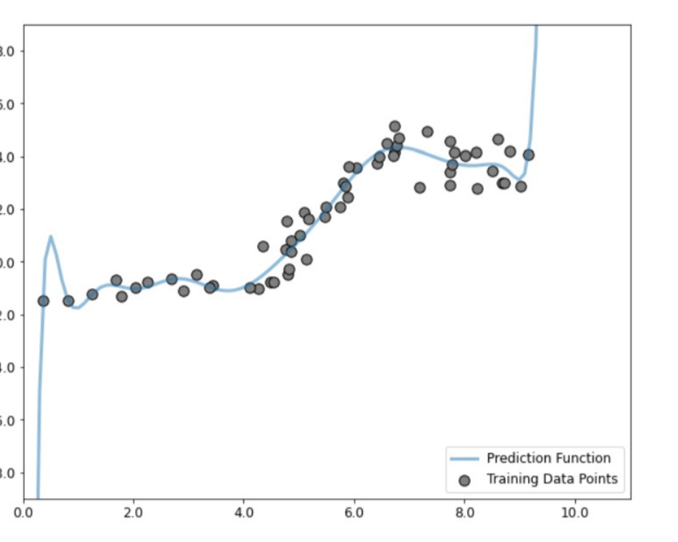
Polynomial regression model of degree: 7



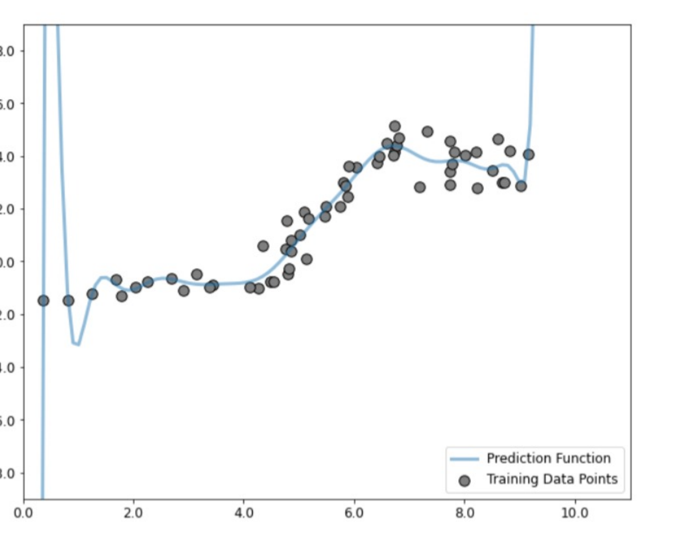
Polynomial regression model of degree: 10



Polynomial regression model of degree: 15

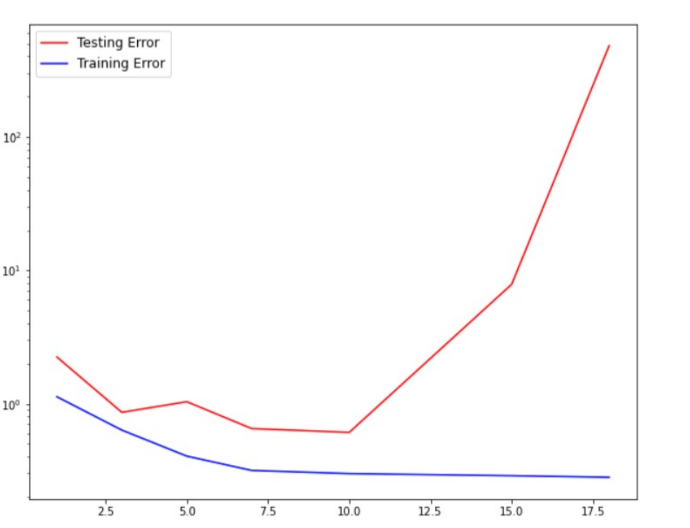


Polynomial regression model of degree: 18



Problem 1 - Part 3(b):

In [7]:
```python
errValue = [None] * len(d)
errTrain = [None] * len(d)

for n in range(len(d)):
    XtrP = mltools.transforms.fpoly(Xtr, d[n], bias = False)
    XtrP,params = mltools.transforms.rescale(XtrP)
    lr = mltools.linear.linearRegress(XtrP, Ytr)
    XteP,_ = mltools.transforms.rescale( mltools.transforms.fpoly(Xte, d[n], False), params)
    errTrain[n] = lr.mse(XtrP, Ytr)
    errValue[n] = lr.mse(XteP, Yte)

figure, axes = plt.subplots(1, 1, figsize=(10, 8))
axes.semilogy(d, errValue, "red", label = "Testing Error")
axes.semilogy(d, errTrain, "blue", label = "Training Error")
axes.legend(fontsize = 12, loc = 0)
print()
plt.show()
print()
```



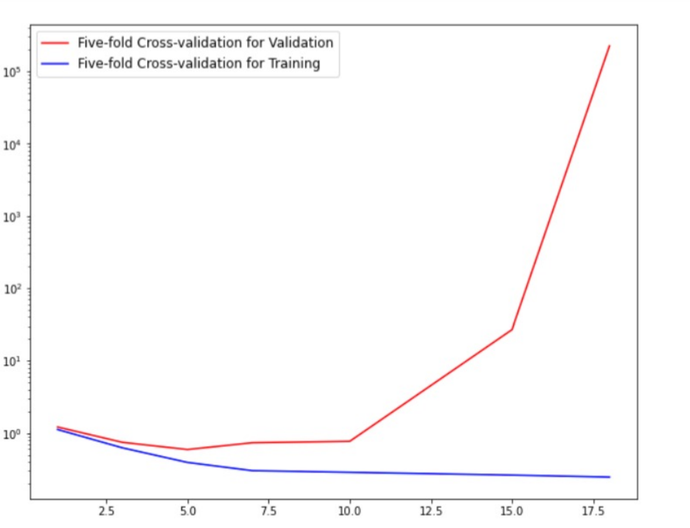
Problem 1 - Part 3(c):

In [8]:
```python
print()
print("\033[1m" + "I recommend", d[numpy.argmin(errValue)], "for the polynomial degree because", d[numpy.argmin(errValue)], "has the smallest value in testing error compare to the other degrees.\n")
```

I recommend 10 for the polynomial degree because 10 has the smallest value in testing error compare to the other degrees.

Problem 2 - Part 1:

In [9]:
```python
d = [1, 3, 5, 7, 10, 15, 18]
nFolds = 5
test_matrix = numpy.zeros((len(d), nFolds))
errTest = [None] * len(d)
train_matrix = numpy.zeros((len(d), nFolds))
errTrain = [None] * len(d)

for k in range(len(d)):
    for iFold in range(nFolds):
        Xti, Xvi, Yti, Yvi = mltools.crossValidate(Xtr, Ytr, nFolds, iFold)
        XtiP = mltools.transforms.fpoly(Xti, d[k], bias = False)
        XtiP, params = mltools.transforms.rescale(XtiP)
        lr = mltools.linear.linearRegress(XtiP, Yti)
        XviP,_ = mltools.transforms.rescale(mltools.transforms.fpoly(Xvi, d[k], False), params)
        test_matrix[k, iFold] = lr.mse(XviP, Yvi)
        train_matrix[k, iFold] = lr.mse(XtiP, Yti)

for n in range(len(d)):
    errTest[n] = numpy.mean(test_matrix[n])
    errTrain[n] = numpy.mean(train_matrix[n])

figure, axes = plt.subplots(1, 1, figsize=(10, 8))
axes.semilogy(d, errTest, "red", label = "Five-fold Cross-validation for Validation")
axes.semilogy(d, errTrain, "blue", label = "Five-fold Cross-validation for Training")
axes.legend(fontsize = 12, loc = 0)
print()
plt.show()
print()
```



Problem 2 - Part 2:

Here we can choose five which is a better polynomial degree than ten that we got in the problem one since the learner function trains with the various training data points in this case. Therefore, the fold cross-validation leads to a better or the best training data model which can make the results more precise. In addition, the fold cross-validation error is smaller compare to the error which we got by using the MSE.

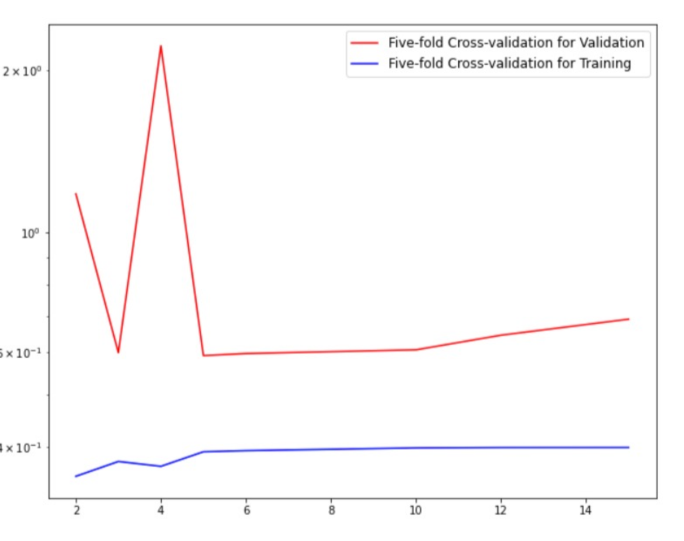
Problem 2 - Part3:

In [10]:
```python
print()
print("\033[1m" + "Based on five-fold cross-validation error I recommend polynomial degree of:", d[numpy.argmin(errTest)], "\n")
```

Based on five-fold cross-validation error I recommend polynomial degree of: 5

Problem 2 - Part4:

In [11]:
```python
nFolds = [2, 3, 4, 5, 6, 10, 12, 15]
test_matrix = numpy.zeros(len(nFolds))
train_matrix = numpy.zeros(len(nFolds))

for k in range(len(nFolds)):
    for iFold in range(nFolds[k]):
        Xti, Xvi, Yti, Yvi = mltools.crossValidate(Xtr, Ytr, nFolds[k], iFold)
        XtiP = mltools.transforms.fpoly(Xti, 5, bias = False)
        XtiP, params = mltools.transforms.rescale(XtiP)
        lr = mltools.linear.linearRegress(XtiP, Yti)
        XviP, _ = mltools.transforms.rescale(mltools.transforms.fpoly(Xvi, 5, False), params)
        test_matrix[k] += lr.mse(XviP, Yvi)
        train_matrix[k] += lr.mse(XtiP, Yti)
    test_matrix[k] /= nFolds[k]
    train_matrix[k] /= nFolds[k]

figure, axes = plt.subplots(1, 1, figsize=(10, 8))
axes.semilogy(nFolds, test_matrix, "red", label = "Five-fold Cross-validation for Validation")
axes.semilogy(nFolds, train_matrix, "blue", label = "Five-fold Cross-validation for Training")
axes.legend(fontsize = 12, loc = 0)
print()
plt.show()
print()
```



I can understand from the pattern that a good and ideal model can be obtained by having the fold cross-validation with more folds in order to teach the machine with a better training data. In other words, we can have more precise training data on the specific part of the data by breaking the data into the more number of pieces. Furthermore, MSE plot can be very helpful and essential to cross-validation in order to assist in choosing a better number of folds for the cross-validation.

Problem 3:

I completed HW2 entirely on my own by attending the lectures and the discussions. In addition, I used the lecture notes and read through the base codes which was provided in the zip file. Also, I read the documentation for the python libraries which we are using more closely such as numpy and matplot. Finally, I completely followed the academic honesty guidelines which is on our canvas website and I did not discuss my homework with anyone in-person.