Problem 2

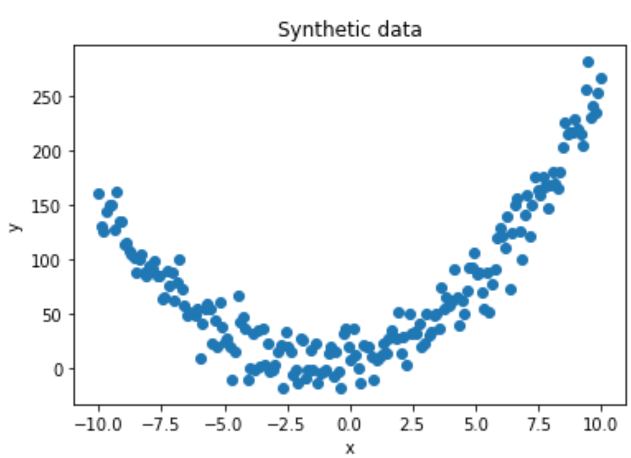
In [3]:
```
# Setup:

import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import KFold

import warnings
warnings.filterwarnings("ignore")
```

Example code using the polyfit and kfold functions

Note: This section is not part of the homework problem, but provides some potentially-helpful example code regarding the usage of numpy.polyfit, numpy.polyval, and sklearn.model_selection.KFold.

First, let's generate some synthetic data: a quadratic function plus some Gaussian noise.

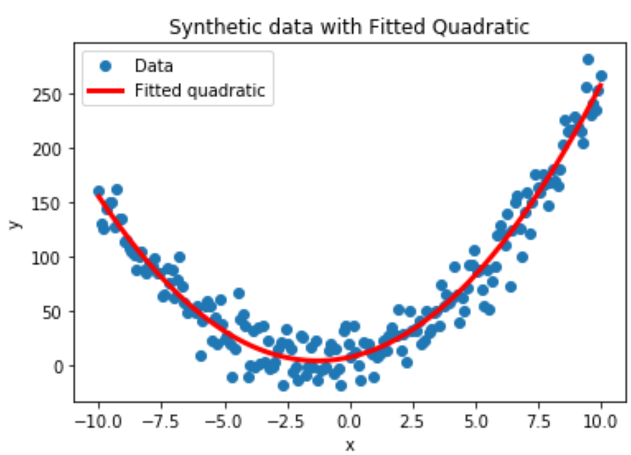
In [4]:
```
# Coefficients of the quadratic function, y(x) = ax^2 + bx + c:
a = 2
b = 5
c = 7

N = 200        # Number of data points
x = np.linspace(-10, 10, num = N)                # x ranges from -10 to 10
# y is the quadratic function of x specified by a, b, and c, plus noise
y = a*x**2 + b*x + c + 15* np.random.randn(N)

# Plot the data:
plt.figure()
plt.plot(x, y, marker = 'o', linewidth = 0)

plt.xlabel('x')
plt.ylabel('y')
plt.title('Synthetic data')

plt.show()
```



Next, we'll use the numpy.polyfit function to fit a quadratic polynomial to this data. We can evaluate the resulting polynomial at arbitrary points.

In [5]:
```
# Fit a degree-2 polynomial to the data:
degree = 2
coefficients = np.polyfit(x, y, degree)

# Print out the resulting quadratic function:
print('We fit the following quadratic function: f(x) = %f*x^2 + %f*x + %f' %  \
      (coefficients[0], coefficients[1], coefficients[2]))

# Evaluate the fitted polynomial at x = 4:
x_test = 4
f_eval = np.polyval(coefficients, x_test)
print('\nf(%i) = %f' % (x_test, f_eval))

# Let's visualize our fitted quadratic:
plt.figure()

plt.plot(x, y, marker = 'o', linewidth = 0)
plt.plot(x, np.polyval(coefficients, x), color = 'red', linewidth = 3)

plt.legend(['Data', 'Fitted quadratic'], loc = 'best')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Synthetic data with Fitted Quadratic')

plt.show()
```

```
We fit the following quadratic function: f(x) = 1.985406*x^2 + 5.113636*x + 7.60
0382

f(4) = 59.821414
```



Finally, assume that we'd like to perform 10-fold cross validation with this dataset. Let's divide it into training and test sets, and print out the test sets. To limit the amount of text that we are printing out, we'll modify the dataset to make it smaller.

In [6]:
```
# Coefficients of the quadratic function, y = ax^2 + bx + c:
a = 2
b = 5
c = 7

N = 80          # Number of points--fewer this time!
x = np.linspace(-10, 10, num = N)               # x ranges from -10 to 10
# y is the quadratic function of x specified by a, b, and c, plus noise
y = a*x**2 + b*x + c + 15* np.random.randn(N)

# Initialize kfold cross-validation object with 10 folds:
num_folds = 10
kf = KFold(n_splits=num_folds)

# Iterate through cross-validation folds:
i = 1
for train_index, test_index in kf.split(x):

    # Print out test indices:
    print('Fold ', i, ' of ', num_folds, ' test indices:', test_index)

    # Training and testing data points for this fold:
    x_train, x_test = x[train_index], x[test_index]
    y_train, y_test = y[train_index], y[test_index]

    i += 1
```

```
Fold  1  of  10  test indices: [0 1 2 3 4 5 6 7]
Fold  2  of  10  test indices: [ 8  9 10 11 12 13 14 15]
Fold  3  of  10  test indices: [16 17 18 19 20 21 22 23]
Fold  4  of  10  test indices: [24 25 26 27 28 29 30 31]
Fold  5  of  10  test indices: [32 33 34 35 36 37 38 39]
Fold  6  of  10  test indices: [40 41 42 43 44 45 46 47]
Fold  7  of  10  test indices: [48 49 50 51 52 53 54 55]
Fold  8  of  10  test indices: [56 57 58 59 60 61 62 63]
Fold  9  of  10  test indices: [64 65 66 67 68 69 70 71]
Fold  10  of  10  test indices: [72 73 74 75 76 77 78 79]
```

Loading the Data for Problem 2

This code loads the data from bv_data.csv using the load_data helper function. Note that data[:, 0] is an array of all the x values in the data and data[:, 1] is an array of the corresponding y values.

In [7]:
```
def load_data(filename):
    """
    Function loads data stored in the file filename and returns it as a numpy ndar
ray.
    Input:
        filename: given as a string.
    Output:
        Data contained in the file, returned as a numpy ndarray
    """
    return np.loadtxt(filename, skiprows=1, delimiter=',')
```

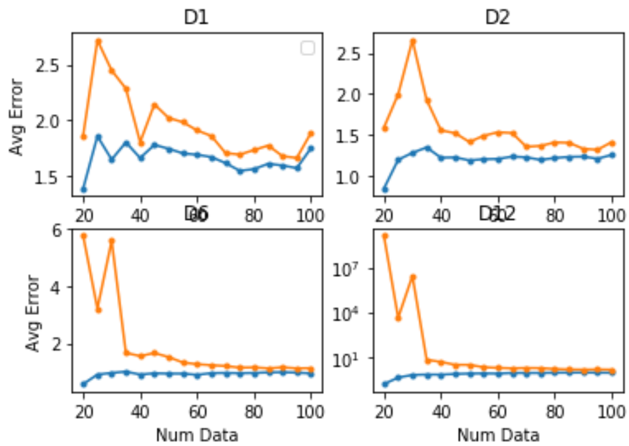
In [8]:
```
data = load_data('data/bv_data.csv')
x = data[:, 0]
y = data[:, 1]
```

Write your code below for solving problem 2 part B:

In [9]:
```
def sqErr(n1, n2):
    # return square error
    return (n1 - n2) ** 2


def getData(data):
    # return arrays representing x and y
    x = []
    y = []

    for i in data:
        x.append(i[0])
        y.append(i[1])

    return np.asarray(x), np.asarray(y)


def retKFoldData(n, numFolds, data):
    # return training data and validation data for one fold
    # fold number = section number that serves as validation number

    foldNumber = int(n)
    part = int(len(data) / numFolds)
    numFolds = int(numFolds)

    lower = (foldNumber - 1) * part
    upper = foldNumber * part

    training = np.concatenate((data[:lower], data[upper:]), axis = 0)
    validation = data[lower : upper]
    return training, validation


def avgErr(data, folds, degreePoly):
    # return avg training & validation errors

    trainingErrs = []
    validationErrs = []

    for k in range(1, folds + 1):
        trainingErr = 0
        validationErr = 0
        training, validation = retKFoldData(k, folds, data)  #Get training and val
idation data from the retKFoldData function

        trainX, trainY = getData(training)
        validationX, validationY = getData(validation)
        z = np.polyfit(trainX, trainY, degreePoly)    #np.polyfit will return coef
ficients

        # use training data to find training error
        for a in range(len(trainX)):
            p1 = np.polyval(z, trainX[a])
            trainingErr += sqErr(p1, trainY[a])

        # use validation data to find validation error
        for b in range(len(validationX)):
            p2 = np.polyval(z, validationX[b])
            validationErr += sqErr(p2, validationY[b])

        trainingErrs.append(trainingErr / len(trainX))
        validationErrs.append(validationErr / len(validationX))

    return sum(trainingErrs) / folds, sum(validationErrs) / folds
```

In [10]:
```
finalData = load_data("data/bv_data.csv")

trainingErr = []
validationErr = []

for d in [1, 2, 6, 12]:
    for size in range(20,101,5):
        t = finalData[:size]
        avgTrain, avgValidation = avgErr(t, 5, d)

        trainingErr.append(avgTrain)
        validationErr.append(avgValidation)

train1 = trainingErr[:17]
valid1 = validationErr[:17]

train2 = trainingErr[17:34]
valid2 = validationErr[17:34]

train6 = trainingErr[34: 51]
valid6 = validationErr[34:51]

train12 = trainingErr[51:]
valid12 = validationErr[51:]


# plotting figures
N = list(range(20,101,5))
x = N

plt.figure(1)
plt.subplot(221)

plt.title('D1')
plt.legend(('Training', 'Validation'))
plt.plot(x, train1, x, valid1, marker = '.')
plt.ylabel('Avg Error')
plt.margins(tight=True)

plt.subplot(222)
plt.title('D2')
plt.plot(x, train2, x, valid2, marker = '.')
plt.margins(tight=True)

plt.subplot(223)
plt.title('D6')
plt.plot(x, train6, x, valid6, marker = '.')
plt.xlabel('Num Data')
plt.ylabel('Avg Error')
plt.margins(tight=True)

plt.subplot(224)
plt.title('D12')
plt.plot(x, train12, x, valid12, marker = '.')
plt.xlabel('Num Data')
# use log scale for fit
plt.yscale('log')
plt.margins(tight=True)

# plt.subplots_adjust(left=.5, bottom=.5, right=.5, top=.5, wspace=.5, hspace=Non
e)
```

Out[10]: (0.05, 0.05)



In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: