Technique Assignment 3: Linear regression

Cogs 109 Spring 2020

Student Name

In [1]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
%matplotlib inline
```

1. (15 points) Datasets and variables

Find a dataset from the UCI machine learning repository (http://archive.ics.uci.edu/ml/index.php) that is suitable for linear regression. Provide a link to your chosen dataset and briefly describe its content.

(9) List the following:

- number of variables
- number of samples
- labels (what is the label?)

(6) Create and report a research question that you could answer using this dataset and some or all of the variables.

Link: http://archive.ics.uci.edu/ml/datasets/Communities+and+Crime (http://archive.ics.uci.edu/ml/datasets/Communities+and+Crime)
This data set contains information about different communities across America. The information for each community is in some way connected to crime. The communities that were not found in census and crime datasets were omitted from the dataset.

- It is a dataset of 1994 samples (communities)
- with 128 variables each (attributes of each community).
- The label is each community.

I think this dataset is can have a lot of different research questions.
First, we can see how each different attribute of the community affects the number of Violent Crimes per Capita.

- ie. How the percentage of unemployed people affects crime
- or how the police budget per capita affects crime

We can also do multivariate regression on this dataset, where we can take two variables that may have dependency on each other like 'Police Budget per Capita' and number of 'Police Cars' and see how these can affect the 'Violent Crimes per Capita'.

2. (20) Arrays and numpy

This tutorial should be very helpful: https://www.numpy.org/devdocs/user/quickstart.html (https://www.numpy.org/devdocs/user/quickstart.html)

a. (10)

Use arange() and reshape() from numpy to create a 4x5 array containing the integers 1 through 20. Append a column of ones to the left of your array to create a 4x6 array. Multiply every element of the array by 2. Print **only** the resulting array.

In [2]:
```
x = np.arange(1, 21).reshape(4, 5)
ones = np.ones(4, dtype=int)
```

In [3]:
```
combined = np.vstack([ones, x.T]).T
combined * 2
```

Out[3]:
```
array([[ 2,  2,  4,  6,  8, 10],
       [ 2, 12, 14, 16, 18, 20],
       [ 2, 22, 24, 26, 28, 30],
       [ 2, 32, 34, 36, 38, 40]])
```

b. (10)

Use linspace() and reshape() to create a 20 x 20 array that contains a smooth range of values between 0 and 1, inclusive.

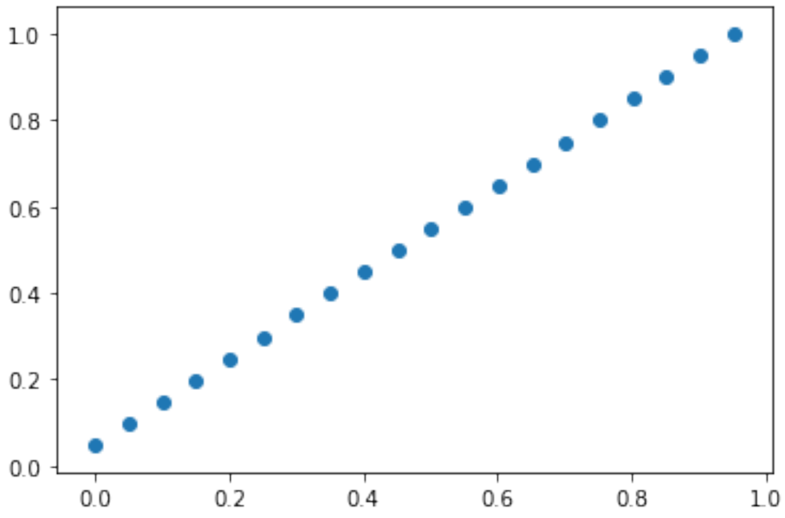
Create a scatter plot using the first (leftmost) column of your array as the x values and the last (rightmost) column of your array as the y values. Use x limits 0 and 1 and y limits 0 and 1 for your plot.

```
In [4]: data = np.linspace(0, 1, 20*20).reshape(20,20)
```

```
In [5]: x = data.T[0]
        y = data.T[19]
```

```
In [6]: plt.scatter(x, y)
```

```
Out[6]: <matplotlib.collections.PathCollection at 0x116f69690>
```



3. (40 points) Univariate linear regression

Note: Solutions to this problem must follow the method described in class and the linear regression handout. There is some flexibility in how your solution is coded, but you may not use special functions that automatically perform linear regression for you.

Load in the BodyBrainWeight.csv dataset. Perform linear regression using two different models:

M1: brain_weight = w0 + w1 x body_weight

M2: brain_weight = w0 + w1 x body_weight + w2 x body_weight^2

a. (15)

For each model, follow the steps shown in class to solve for w. Report the model, including w values and variable names for both models.

b. (10)

Use subplots to display two graphs, one for each model. In each graph, include:

- Labeled x and y axes
- Title
- Scatterplot of the dataset
- A smooth line representing the model

c. (10)

For each model, calculate the sum squared error (SSE). Show your 2 SSE values together in a bar plot.

d. (5)

Which model do you think is better? Why? Is there a different model that you think would better represent the data?

Part a: Models

In [7]:
```
## Load the dataset and extract Body weight as X and Brain weight a
s Y
bbdata = pd.read_csv("BodyBrainWeight.csv").values
```

In [8]:
```
X = bbdata[:,0] # body weight
Y = bbdata[:,1] # brain weight
```

In [9]:
```
## Create A, the augmented data array
ones = np.ones(X.size)

A1 = np.vstack([ones, X]).T

## Solve for w, the weight vector
w1 = np.linalg.lstsq(A1, Y, rcond = None)[0]

print("Model M1:")
print("Brain weight = %f + %f x Body_Weight" % tuple(w1))
```

```
Model M1:
Brain weight = 124.928105 + 0.937039 x Body_Weight
```

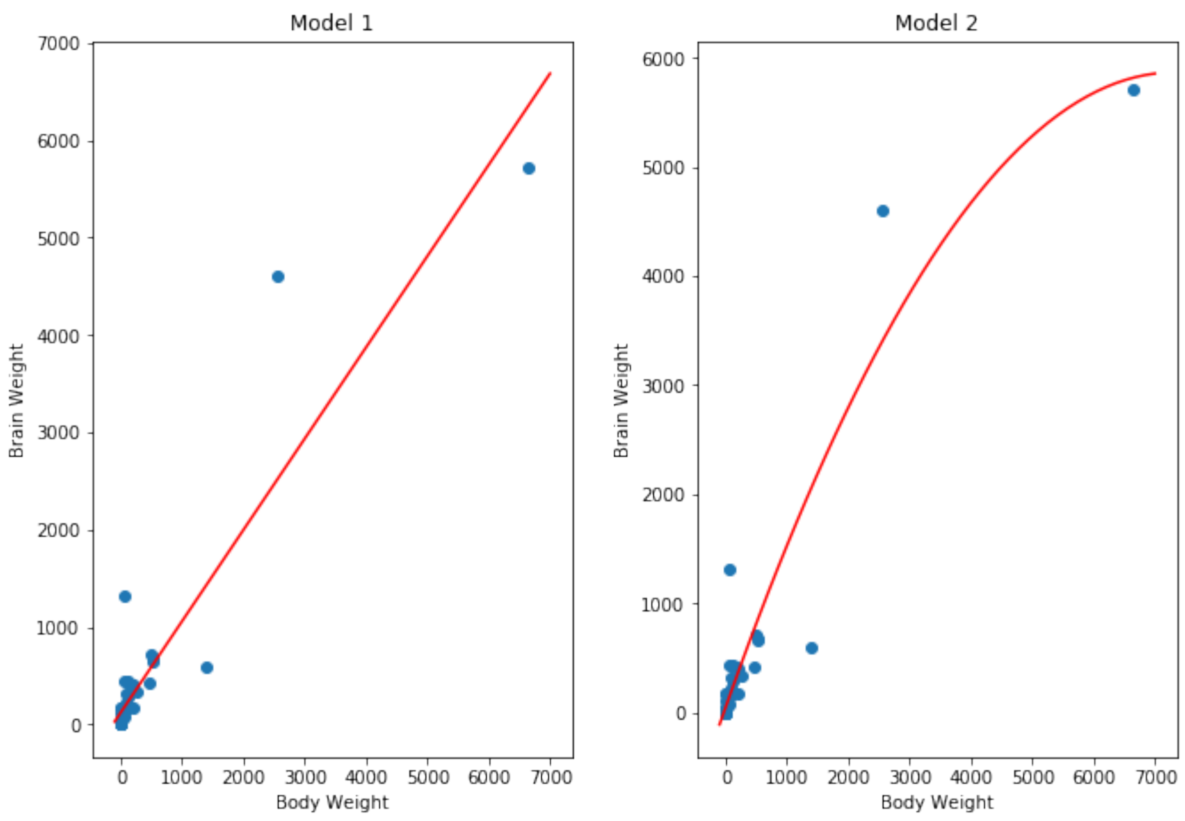
In [10]:
```
## Create A, the augmented data array

ones = np.ones(X.size)
squares = np.square(X)

A2 = np.vstack([ones, X, squares]).T

## Solve for w, the weight vector
w2 = np.linalg.lstsq(A2, Y, rcond = None)[0]

print("Model M2:")
print("Brain weight = %f + %f x Body_Weight + %f x Body_Weight^2" %
tuple(w2))
```

```
Model M2:
Brain weight = 49.091369 + 1.590462 x Body_Weight + -0.000109 x Bo
dy_Weight^2
```

Part b: Plotting

In [11]:
```
fig, axes = plt.subplots(1,2, figsize=(10,7))
fig.tight_layout(pad=4.0)

# set names for model 1
axes[0].set_title('Model 1')
axes[0].set_xlabel("Body Weight")
axes[0].set_ylabel("Brain Weight")

# set names for model 2
axes[1].set_title('Model 2')
axes[1].set_xlabel("Body Weight")
axes[1].set_ylabel("Brain Weight")

# scatter plots
axes[0].scatter(X, Y)
axes[1].scatter(X, Y)

# line for Model 1
x1 = np.linspace(-100, 7000, X.size)
A1 = np.vstack([ones, x1]).T
y1 = np.matmul(A1, w1)
axes[0].plot(x1, y1, color = 'red')

# line for Model 2
x2 = np.linspace(-100, 7000, X.size)
squares = np.square(x2)
A2 = np.vstack([ones, x2, squares]).T
y2 = np.matmul(A2, w2)
axes[1].plot(x2, y2, color = 'red')
```

```
Out[11]: [<matplotlib.lines.Line2D at 0x1171501d0>]
```



Part c: SSE

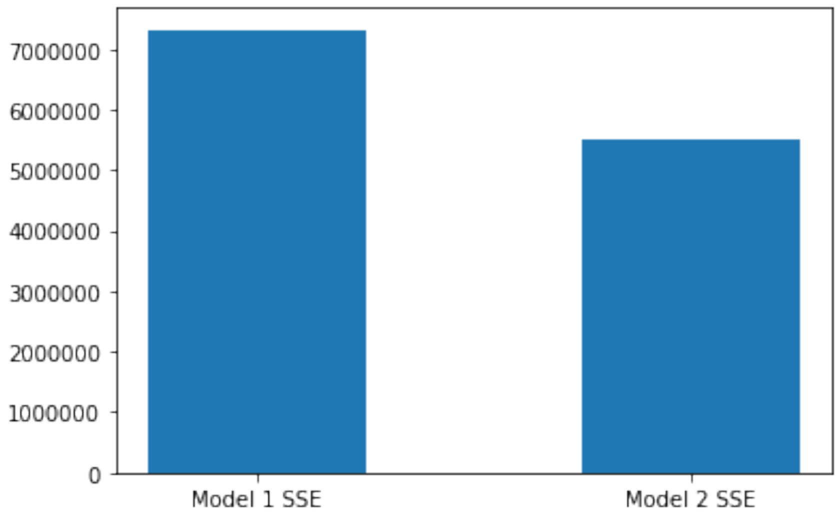
```
In [12]: # function that calculates the SSE of the data given
         def SSE(actual, predicted):
             return np.sum(np.square(predicted - actual))
```

```
In [13]: actual = Y
         ones = np.ones(X.size)
         squares = np.square(X)
```

```
In [14]: # Model 1 SSE
         A = np.vstack([ones, X]).T
         predicted = np.matmul(A, w1)

         SSE1 = SSE(actual, predicted)
```

```
In [15]: # Model 2 SSE
         A = np.vstack([ones, X, squares]).T
         predicted = np.matmul(A, w2)

         SSE2 = SSE(actual, predicted)
```

```
In [16]: plt.bar(['Model 1 SSE', 'Model 2 SSE'], [SSE1, SSE2], width=0.5)

Out[16]: <BarContainer object of 2 artists>
```



Part d: Conclusion

- Model 2
- Because it has considerably lower SSE and it follows the pattern well without overfitting.
- No, I think that higher order models will start overffiting the data.

4. (25 points) Multivariate linear regression with cross validation

Using the dataset found in Housing.csv, build a multivariate model to predict house price using lot size and the number of bedrooms as predictors.

Hint: You may use this as your model:

Price = w0 + w1 x Lot size + w2 x Bedrooms

First, split your data into a training set (80%) and a test set (20%). Then perform linear regression using the **training data** only. Report your model and show the mean squared error (MSE) for your **training** and **test** data using a bar graph.

MSE can be found by dividing SSE by the number of samples in your data.

```
In [113]: ## Load the housing data
          df = pd.read_csv("Housing.csv")
          df = df.sample(frac=1, replace=False) # shuffle the data
```

In [97]:
```
# Extract the variables
Y = df['price'].values
X1 = df['lotsize'].values
X2 = df['bedrooms'].values
```

In [99]:
```
# Get training data, 109 being the 20% cut
Y_train = Y[109:]
X1_train = X1[109:]
X2_train = X2[109:]
```

In [100]:
```
# Create A, augmented array
ones = np.ones(X1_train.size)

A_train = np.vstack([ones, X1_train, X2_train]).T

# Calculate weights
w = np.linalg.lstsq(A_train, Y_train, rcond = None)[0]
```

In [101]:
```
print("Model")
print("Price = %f + %f x Lot size + %f x Bedrooms" % tuple(w))
```

```
Model
Price = 2589.021254 + 6.114866 x Lot size + 11653.785920 x Bedroom
s
```

In [102]:
```
# MSE for training data
predicted = np.matmul(A_train, w)
MSE_train = SSE(Y_train, predicted) / predicted.size
MSE_train
```

Out[102]:
```
476094526.85080856
```

In [103]:
```
# MSE on test data
Y_test = Y[:109]
X1_test = X1[:109]
X2_test = X2[:109]

ones = np.ones(Y_test.size)
```

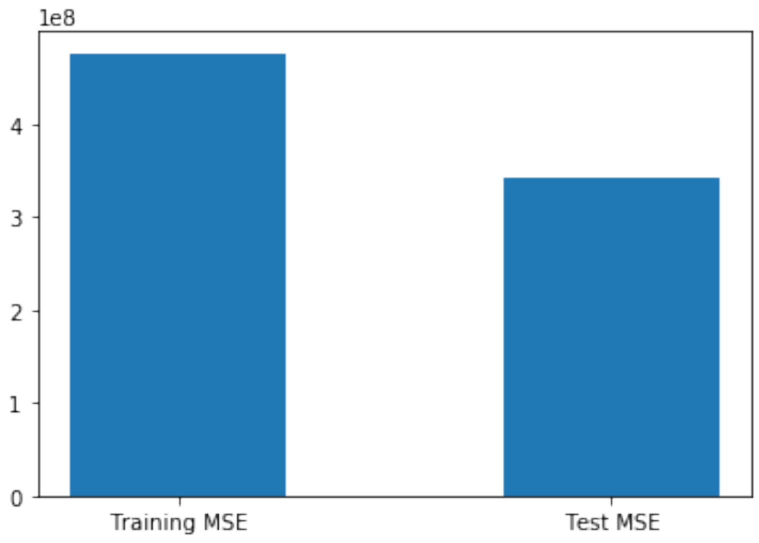
In [104]:
```
A_test = np.vstack([ones, X1_test, X2_test]).T
```

In [105]:
```
predicted = np.matmul(A_test, w)
MSE_test = SSE(Y_test, predicted) / predicted.size
MSE_test
```

Out[105]:
```
341144544.7778806
```

In [106]:
```
plt.bar(['Training MSE', 'Test MSE'], [MSE_train, MSE_test], width=
0.5)
```

Out[106]:
```
<BarContainer object of 2 artists>
```



Extra Credit

I think the following model could also be used:

- Price = w0 + w1 x Lot size + w2 x Lot size x Bedrooms

Since the Lot size and the number of Bedrooms could have some dependency on each other.
Usually bigger houses should have more bedrooms.
The model above allows us to account for variable interaction between lotsize and bedroom.

In [107]:
```
# Create A, augmented array
ones = np.ones(Y_train.size)
combined = X1_train * X2_train

A_train = np.vstack([ones, X1_train, combined]).T

# Calculate weights
w = np.linalg.lstsq(A_train, Y_train, rcond = None)[0]
```

In [108]:
```
print("Model")
print("Price = %f + %f x Lot size + %f x Lot size x Bedrooms " % tu
ple(w))
```

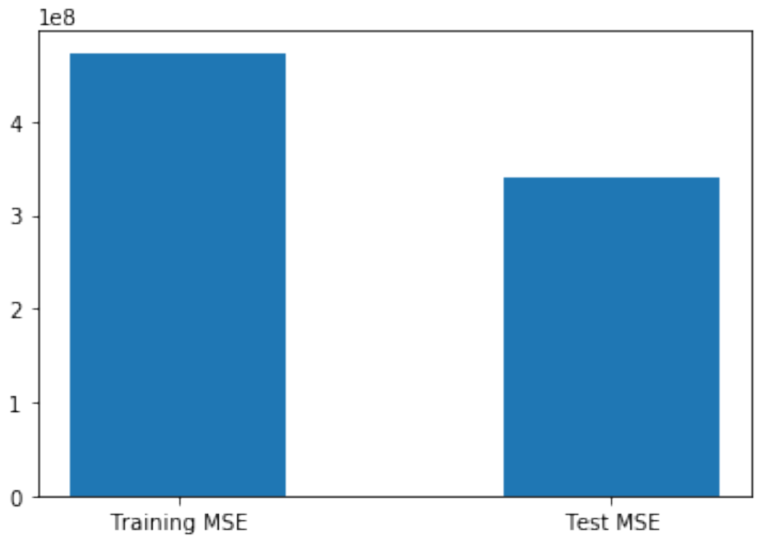
```
Model
Price = 38112.406057 + -0.388698 x Lot size + 2.093167 x Lot size
x Bedrooms
```

```
In [109]: # MSE for training data
          predicted = np.matmul(A_train, w)
          MSE_train = SSE(Y_train, predicted) / predicted.size
          MSE_train

Out[109]: 473354572.0371508
```

```
In [110]: # MSE on test data
          ones = np.ones(Y_test.size)
          combined = X1_test * X2_test

          A_test = np.vstack([ones, X1_test, combined]).T
```

```
In [111]: predicted = np.matmul(A_test, w)
          MSE_test = SSE(Y_test, predicted) / predicted.size
          MSE_test

Out[111]: 339821279.8233979
```

```
In [112]: plt.bar(['Training MSE', 'Test MSE'], [MSE_train, MSE_test], width=
          0.5)

Out[112]: <BarContainer object of 2 artists>
```



The difference in the MSEs is increased using the second model (Test SSE decreased).
But the difference is not very significant (differnce only by 1%).