Elizabeth Camp CS545 Feb 9 2023

Homework 2: This HW is based on the code for Multiple Variable Linear Regression

Instructions:

Place the answer to your code only in the area specified. Also, make sure to run all your code, meaning, press >> to "Restart Kernel and Run All Cells". This should plot all outputs including your answers to homework questions. After this, go to file (top left) and select "Print". Save your file as a PDF and upload the PDF to Canvas.

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

%matplotlib inline is a magic function that makes plots appear next to code and be stored in notebook: https://stackoverflow.com/questions/43027980/purpose-of-matplotlib-inline

Diabetes Data Example

To illustrate the concepts, we load the well-known diabetes data set. This dataset is included in the `sklearn.datasets` module and can be loaded as follows.

In [2]:
```
from sklearn import datasets, linear_model

# Load the diabetes dataset
diabetes = datasets.load_diabetes()
X = diabetes.data
y = diabetes.target
```

We can print a description of the data as follows:

In [3]:
```
print(diabetes.DESCR)
```

```
.. _diabetes_dataset:

Diabetes dataset
----------------

Ten baseline variables, age, sex, body mass index, average blood
pressure, and six blood serum measurements were obtained for each of n =
442 diabetes patients, as well as the response of interest, a
quantitative measure of disease progression one year after baseline.

**Data Set Characteristics:**

  :Number of Instances: 442

  :Number of Attributes: First 10 columns are numeric predictive values

  :Target: Column 11 is a quantitative measure of disease progression one year
after baseline

  :Attribute Information:
      - age     age in years
      - sex
      - bmi     body mass index
      - bp      average blood pressure
      - s1      tc, total serum cholesterol
      - s2      ldl, low-density lipoproteins
      - s3      hdl, high-density lipoproteins
      - s4      tch, total cholesterol / HDL
      - s5      ltg, possibly log of serum triglycerides level
      - s6      glu, blood sugar level

Note: Each of these 10 feature variables have been mean centered and scaled by
the standard deviation times `n_samples` (i.e. the sum of squares of each colu
mn totals 1).

Source URL:
https://www4.stat.ncsu.edu/~boos/var.select/diabetes.html

For more information see:
Bradley Efron, Trevor Hastie, Iain Johnstone and Robert Tibshirani (2004) "Lea
st Angle Regression," Annals of Statistics (with discussion), 407-499.
(https://web.stanford.edu/~hastie/Papers/LARS/LeastAngle_2002.pdf)
```

The target values are stored in the vector `y`. The attributes for the diabetes data are stored in a data matrix, `X`. The size is is number of samples (442) x number of attributes (10).

In [4]:
```
nsamp, natt = X.shape
print("num samples={0:d}  num attributes={1:d}".format(nsamp,natt))
```

```
num samples=442  num attributes=10
```

In the code above, we use the fromat method to help with output formatting. You use {} to indicate where the output would be substituted and you provide the variable to be used inside the format method, see more: https://docs.python.org/3/tutorial/inputoutput.html

Question 1:

Print the ages of the first five subjects?

In [5]:
```
# first 5 subjects are rows 0-4
# (note that slicing syntax excludes endpoint row index 5),
# age column has index 0
X[:5, 0]
```

Out[5]:
```
array([ 0.03807591, -0.00188202,  0.08529891, -0.08906294,  0.00538306])
```

Question 2:

Print the attributes S1-S3 for subjects 10-15

In [6]:
```
# python is zero indexed so patient 10 has index 9 for row,
# S1 is the 5th column so index is 4 since python is zero-indexed,
# index slicing syntax exludes the endpoint which is column 7
# so only columns 4, 5, 6 are returned
X[9:15, 4:7]
```

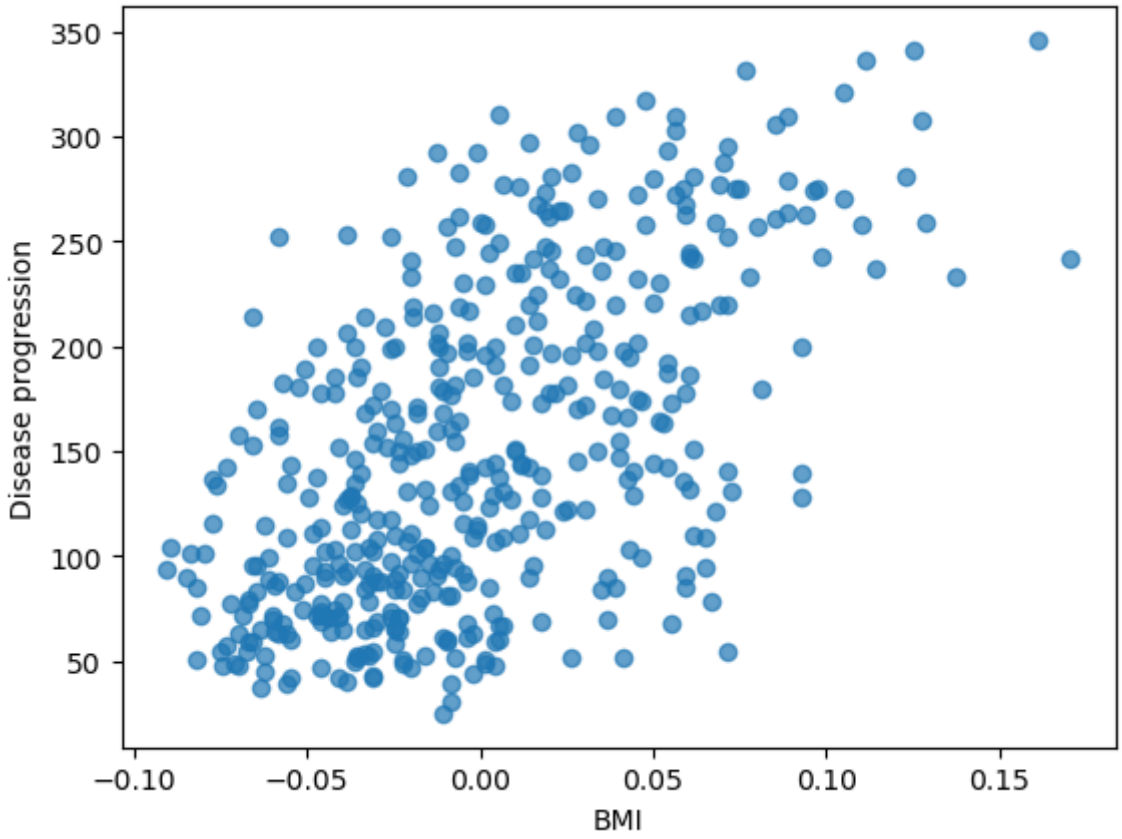
Out[6]:
```
array([[-1.25765827e-02, -3.45076144e-02, -2.49926566e-02],
       [-1.03389471e-01, -9.05611890e-02, -1.39477432e-02],
       [-7.07277125e-03,  4.59715403e-02, -6.54906725e-02],
       [-4.32086554e-03, -9.76888589e-03,  4.49584616e-02],
       [-4.32086554e-03, -1.57187067e-02, -2.90282981e-03],
       [ 1.76943802e-02, -6.12835791e-05,  8.17748397e-02]])
```

Question 3:

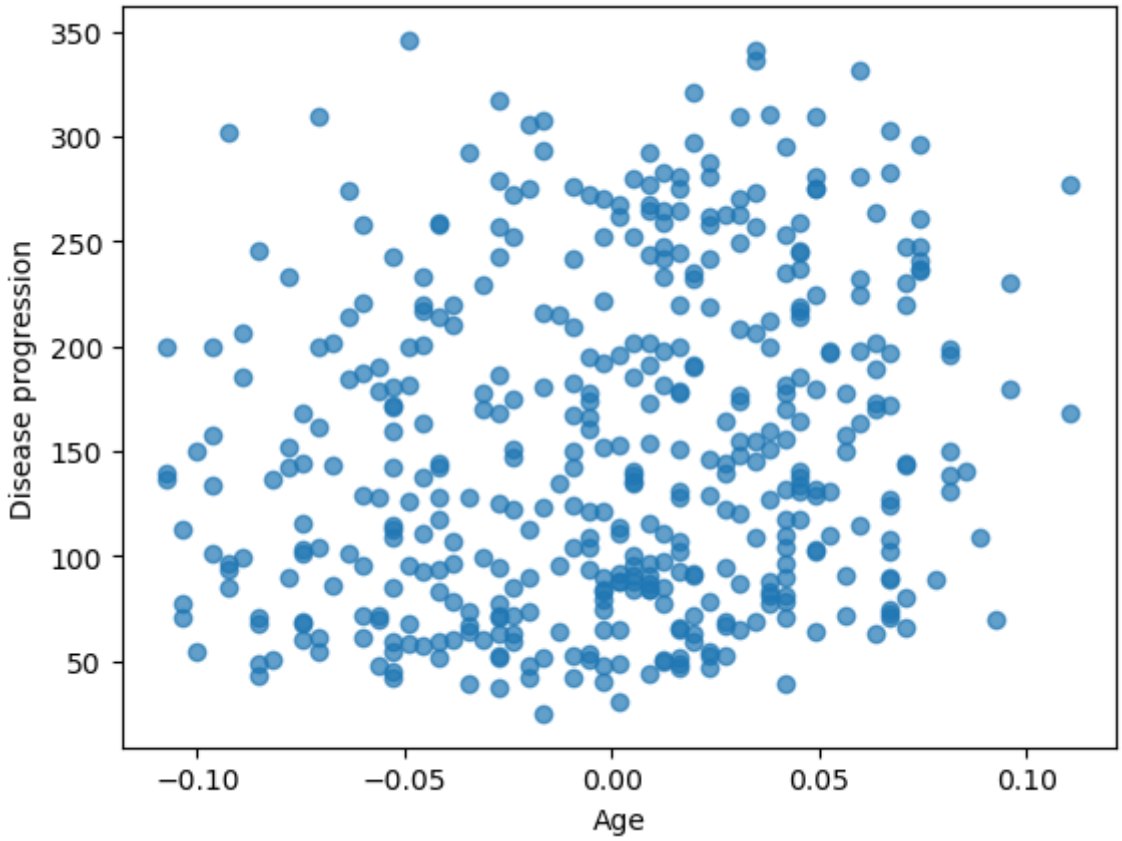
Create a scatter plot of the target variable, `y` vs. the BMI. Does there seem to be a relation? What about `y` vs. the age? Which is a better predictor?

Based on the plots below, there is a more clear linear relationship between `y` and `BMI` whereas there doesn't appear to be a linear trend between `y` and `Age`. So, based on a visual inspection of these plots, `BMI` appears be a better predictor than `Age`.

In [7]:
```
# plot of y (disease progression) vs BMI
plt.scatter(X[:, 2], y,  marker = 'o', alpha=0.7)
plt.xlabel("BMI")
plt.ylabel("Disease progression")
plt.show()
```



In [8]:
```
# plot of y (disease progression) vs age
plt.scatter(X[:, 0], y,  marker = 'o', alpha=0.7)
plt.xlabel("Age")
plt.ylabel("Disease progression")
plt.show()
```



Question 4:

You are given target values `y` and features `x1` and `x2` below. Fit the model on the first 4 data points and test the model on the fifth data point. You may want to use the following steps

- Construct the training training data `X_tr,y_tr`
- Create a regression object `regr = linear_model.LinearRegression()`
- Fit the model with the `regr.fit()` method
- Predict the value on the test value with the `regr.predict()`

In [9]:
```
x1 = np.array([0,1,3,5,4])
x2 = np.array([0,0.7, 4.3, 15.1, 13.2])
y = np.transpose(np.array([-2, -0.9, 1.5, 18, 13]))
print('y shape', y.shape)
```

```
y shape (5,)
```

In [10]:
```
# define matrix X from feature vectors
X = np.transpose(np.array([x1, x2]))
```

In [11]:
```
# Split into training data and test data
X_train = X[:4, :]
X_test = X[4,:].reshape(1, -1)

y_train = y[:4,]
y_test = np.array([y[4,]])
```

In [12]:
```
# check shapes of training & test data
print(X_train.shape)
print(y_train.shape)

print(X_test.shape)
print(y_test.shape)
```

```
(4, 2)
(4,)
(1, 2)
(1,)
```

In [13]:
```
# initialize linear regression model object
lr_model = linear_model.LinearRegression()

# fit the linear regression model on the training data
lr_model.fit(X_train, y_train)
```

Out[13]:
```
LinearRegression()
```

In [14]:
```
# Determine the predicted response y given the test set of predictors
y_predicted = lr_model.predict(X_test)
y_predicted
```

Out[14]:
```
array([15.98170828])
```

In [15]:
```
y_test
```

Out[15]:
```
array([13.])
```

Note that with a single observation in the test dataset, the RSS is undefined since the standard deviation of y_test is zero. So the R^2 value is undefined for the test set. One idea would be to use a different model performance metric such as using a very simple percent difference to assess how well the model generalizes.

In [16]:
```
percent_difference = round(float(np.abs(y_predicted - y_test)/y_test *100),4)
print(f"Percent difference between test observation & corresponding prediction:
```

```
Percent difference between test observation & corresponding prediction: 22.936
2
```

Question 5:

Describe the 1SE rule in cross validation and how the model order is selected based on the value of fitness score, i.e., whether a higher or lower firness score is desired and how the model order is determined.

To use the 1SE rule, we would do K-fold cross-validation over several different model orders to arrive at K observed values of RSS (or whichever error metric is chosen) for each model order. Then we would compute the Standard Error (SE) of the K observed values of RSS at each model order. The standard error of RSS at each model order is calculated as $SE(RSS) = SD(RSS)/\sqrt{K-1}$.

Then we would identify the model order with the lowest mean value of RSS. For the model order with this minimum RSS, we would compute the sum $RSS + SE(RSS)$ for that model order to get an RSS_{target}. We would then find any lower model orders with an $RSS \leq RSS_{target}$, we would then choose the lowest model order with an RSS that obeys the inequality.

Essentially, with the 1SE rule, we would like to find the lowest model order (simplest model) with a performance within 1SE of the best performing model.