DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Machine Learning and Data Science

ENCS5341

Homework (1)

Prepared By: Student Name: Ibrahim Nobani

Student ID: 1190278

Instructor: Dr. Yazan Abu Farha

Section: 1

Date: 11/12/2022

First the CSV file was read and appended to a grade list as follows:

```
import csv
grades=[]
with open("./grades.csv", 'r') as file:
  csvreader = csv.reader(file)
  for row in csvreader:
    grades.append(row)
grades.pop(0)
```

Output after reading the file:

```
HW1: [0, 0, 63, 70, 50, 43, 0, 47, 97, 50, 40, 57, 23, 63, 77, 57, 57, 63, 80, 83, 100, 67, 70, 17, 97, 73, 80, 80, 80, 80, 80, 93]
HW2: [78, 37, 100, 93, 75, 0, 47, 80, 97, 67, 0, 87, 100, 50, 93, 83, 87, 50, 97, 100, 95, 73, 87, 80, 97, 87, 100, 90, 93, 100, 100, 47]
Midterm: [32, 25, 34, 35, 20, 26, 22, 26, 30, 21, 18, 28, 29, 26, 29, 28, 28, 27, 40, 57, 56, 34, 32, 0, 44, 25, 46, 43, 48, 41, 47, 38]
Project: [87, 91, 92, 92, 76, 55, 0, 94, 92, 76, 94, 76, 75, 75, 71, 72, 72, 85, 85, 98, 98, 71, 95, 85, 95, 85, 98, 94, 94, 94, 94, 92]
Final: [71, 48, 59, 64, 42, 54, 37, 56, 60, 38, 35, 47, 44, 58, 52, 62, 45, 44, 68, 94, 85, 62, 58, 0, 68, 53, 81, 64, 77, 78, 81, 59]
```

1- Some values are missing (indicated by 0 value). Address all the missing values by using the average of the available values for the corresponding variable.

All missing values were edited as the requested, using the code below:

```
def zeroVar(col,grades):
  avg = 0
  incr = 0
  for i in grades:
    #print(i[col])
    if(i[col]!='0'):
        avg+=int(i[col])
        incr+=1
  for i in grades:
    if (i[col] == '0'):
      i[col]=str(int(avg/incr))
  print(avg/incr)
  return grades

for i in range(0,5):
  (zeroVar(i,grades))
```

Average of each column:

HW1: 66

HW2: 82

Midterm: 33

Project: 85

Final: 59

Output after editing the missing values:

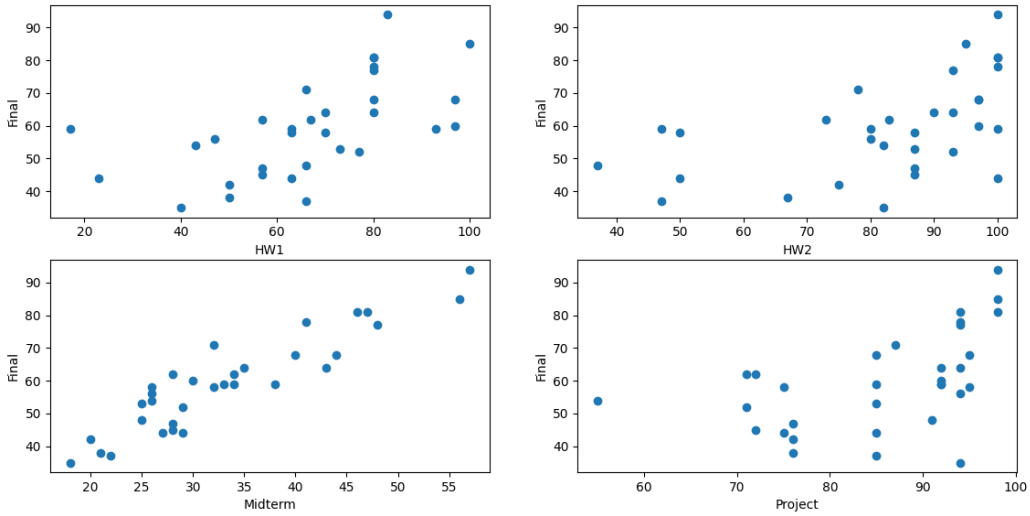
```
HW1: [66, 66, 63, 70, 50, 43, 66, 47, 97, 50, 40, 57, 23, 63, 77, 57, 57, 63, 80, 83, 100, 67, 70, 17, 97, 73, 80, 80, 80, 80, 80, 93]
HW2: [78, 37, 100, 93, 75, 82, 47, 80, 97, 67, 82, 87, 100, 50, 93, 83, 87, 50, 97, 100, 95, 73, 87, 80, 97, 87, 100, 90, 93, 100, 100, 47]
Midterm: [32, 25, 34, 35, 20, 26, 22, 26, 30, 21, 18, 28, 29, 26, 29, 28, 28, 27, 40, 57, 56, 34, 32, 33, 44, 25, 46, 43, 48, 41, 47, 38]
Project: [87, 91, 92, 92, 76, 55, 85, 94, 92, 76, 94, 76, 75, 75, 71, 72, 72, 85, 85, 98, 98, 71, 95, 85, 95, 85, 98, 94, 94, 94, 94, 92]
Final: [71, 48, 59, 64, 42, 54, 37, 56, 60, 38, 35, 47, 44, 58, 52, 62, 45, 44, 68, 94, 85, 62, 58, 59, 68, 53, 81, 64, 77, 78, 81, 59]
```

2- Using data science techniques that we discussed in the course, examine which of the input variables would be a good predictor for the final exam.

Using data visualization a scatter plot was created between each column (HW1+2, Midterm and project) with the final mark to check which one of them is more linear with the final, these diagrams were done with the help of the matplotlib and numpy, using the following code:

```
import matplotlib.pyplot as plt
import numpy as np

f, axis = plt.subplots(nrows = 2, ncols = 2)

axis[0][0].scatter(var1[0],var1[4])
axis[0][0].set_xlabel('HW1')
axis[0][0].set_ylabel('Final')

axis[0][1].scatter(var1[1],var1[4])
axis[0][1].set_xlabel('HW2')
axis[0][1].set_ylabel('Final')

axis[1][0].scatter(var1[2],var1[4])
axis[1][0].set_xlabel('Midterm')
axis[1][0].set_ylabel('Final')

axis[1][1].scatter(var1[3],var1[4])
axis[1][1].set_xlabel('Project')
axis[1][1].set_ylabel('Final')
plt.show()
```

The following figure was obtained after plotting the 4 graphs, by observing these graphs, you can tell the midterm one gives clearer linear curve than the others, thus it is going to be counted as the predictor for the final exam:



3- Implement the closed form solution of linear regression and use it to learn a linear model to predict the final exam from the variable you selected in part 2.

Here we need to find the parameters of the model w0 and w1 using the two following equations which are implemented in python:

$$w_0 = \frac{\sum_{i=1}^{n} y_i}{n} - w_1 \frac{\sum_{i=1}^{n} x_i}{n}$$

$$w_1 = \frac{\sum_{i=1}^{n} y_i x_i - \frac{\sum_{i=1}^{n} y_i \sum_{i=1}^{n} x_i}{n}}{\sum_{i=1}^{n} x_i^2 - \frac{\sum_{i=1}^{n} x_i \sum_{i=1}^{n} x_i}{n}}$$

And then form the linear regression model:

$$y = f(\mathbf{x}) = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_d x_d$$

Which in our case here is:

$$y = f(\mathbf{x}) = w_0 + w_1 x_1$$

To calculate the following sum $\frac{\sum_{i=1}^{n} x_i}{n}$, this function was implemented:

```
def valuesSum (grades,num):
  sum=0
  for i in grades:
    sum+=int(i[num])
  return sum
```

Output for midterm values: 1068

To calculate the following sum $\frac{\sum_{i=1}^{n} y_i \sum_{i=1}^{n} x_i}{}$, this function was implemented:

```
def valuesConvSum (grades,num,num1):
  sum=0
  for i in grades:
    sum+=(int(i[num])*int(i[num1]))
  return sum
```

Output for midterm*final values: 67739

To calculate the following sum $\sum_{i=1}^{n} x_i^2$, this function was implemented:

```
def valuesSquareSum (grades,num):
  sum=0
  for i in grades:
    sum+=(int(i[num])*int(i[num]))
  return sum
```

Output for midterm squared values: 38788

Then a function to calculate W0,W1 was implemented according to the equations above:

```
def calculateW(grades,xn,yn,num):
  Dividend=(valuesConvSum(grades, xn, yn)-
((valuesSum(grades,xn)*valuesSum(grades,yn))/num))
  Diviser=(valuesSquareSum(grades,xn)-
((valuesSum(grades,xn)*valuesSum(grades,xn))/num))
  W1=Dividend/Diviser
  W0=(valuesSum(grades,yn)/num)- (W1*(valuesSum(grades,xn)/num))
  return W0,W1
```

Outputs of W0,W1 respectively: (14.59670749164944, 1.3444806744075075)

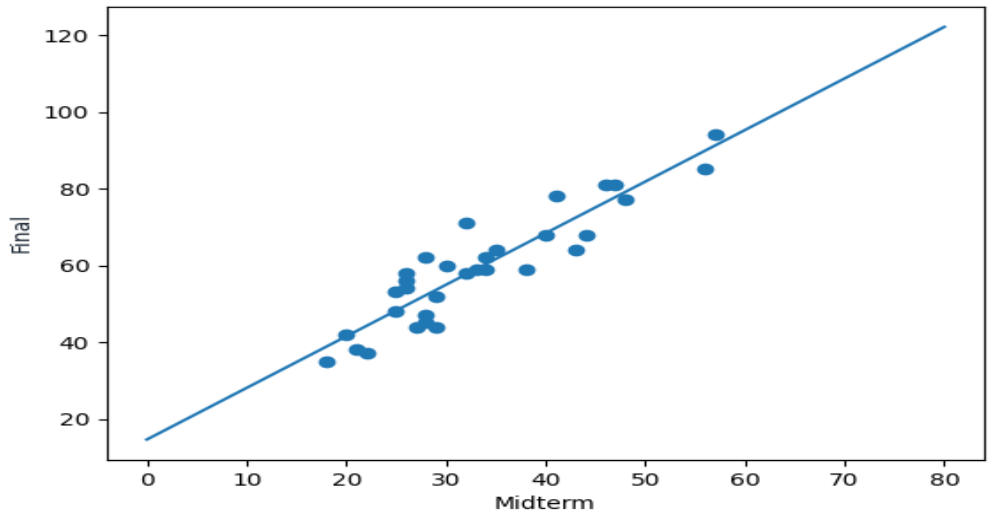
The linear model along with W0,W1 values is obtained as seen here:

```
(14.59670749164944, 1.3444806744075075)
The linear model obtained is F(x)= 14.59670749164944 + 1.3444806744075075 x
```

Then using these values and the obtained linear regression model, it was created and plotted using the following code:

```
x = np.linspace(0,80)
fx=W0+(W1*x)
plt.scatter(var1[int(inp2)-1],var1[4])
plt.plot(x, fx)
plt.xlabel('Midterm')
plt.ylabel('Final', color='#1C2833')
plt.show()
```

The next figure is both the scatter graph and the regression linear model:



4- Repeat part 3 but now by implementing the gradient descent algorithm.

The gradient descent algorithm function is implemented as follows:

```
def gradient_descent(x, y, iterations, n,LearnRate=0.0001):
  W11 = 0.1
  W00 = 0.01
  iterations = iterations
  LearnRate = LearnRate
  for i in range(iterations):
    finalPr = (W11 * x) + W00
    gradient1 = -(2 / n) * sum(x * (y - finalPr))
    gradient2 = -(2 / n) * sum(y - finalPr)
    W11 = W11 - (LearnRate * gradient1)
    W00 = W00 - (LearnRate * gradient2)
    print("Iteration",i,"W1: ",W11,", W0: ",W00)
  return W11, W00
```

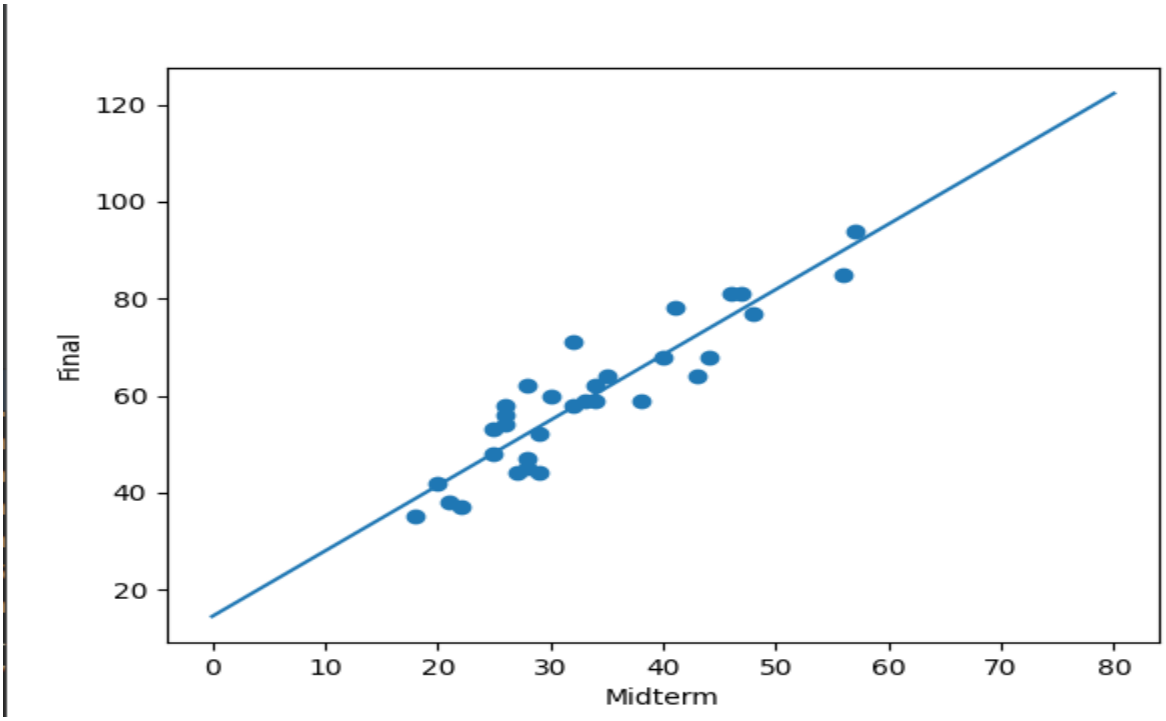
You can enter the number of iterations you wish, or just leave it at 300000, which is the value of iterations I used here:

```
iterations=300000
print("Moving on to the gradient descent, would you like to enter the number
of iterations (Initially set to 300000)?")
print("Press 0 to cancel or the number of iterations you wish to enter")
inpI=input()
if (inpI != '0'):
  iterations=int(inpI)
var1=np.array(Variables[int(inp2)-1])
w1, w0 = gradient_descent(var1, Variables[4],iterations,32)
print(w1,w0)
print("The linear model obtained is F(x)=",w0,"+",w1,"x")
```

This produced the following output after 300000 iterations:

```
Iteration 299998 W1:  1.3475874146745626 , W0:  14.483883343122855
Iteration 299999 W1:  1.3475873643567366 , W0:  14.483885170461278
W1:  1.3475873643567366 W0:  14.483885170461278
The linear model obtained is F(x)= 14.483885170461278 + 1.3475873643567366 x
```

The following graph was then obtained using the linear model from the gradient descent and the scatter plot:



5- Repeat part 3 but now using the linear regression implementation of scikit-learn python library.

This was done using the following code:

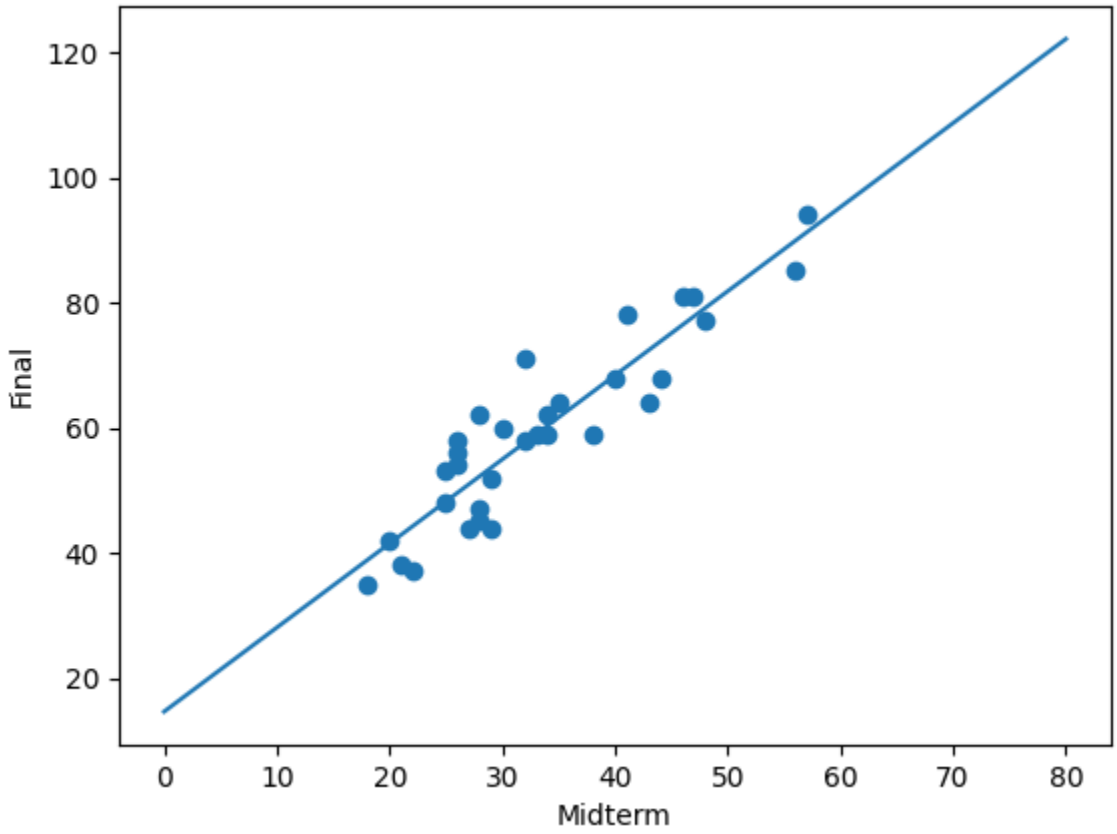
```
from sklearn import linear_model
print("The following is the linear regression using the sci-kit library...")
regr = linear_model.LinearRegression()
var1=(np.array(Variables[int(inp2)-1]))[..., np.newaxis]
var2=(np.array(Variables[4]))[..., np.newaxis]
regr.fit(var1, var2)
Y_axis_predict=regr.predict(var1)
#print("final:",var2)
#print(Y_axis_predict)
print("W1: ", regr.coef_)
print("W0: ", regr.intercept_)
print("The linear model obtained is
F(x)=",regr.intercept_[0],"+",regr.coef_[0][0],"x")
```

The output obtained is:

```
The following is the linear regression using the sci-kit library...
W1:  [[1.34448067]]
W0:  [14.59670749]
The linear model obtained is F(x)= 14.596707491649447 + 1.3444806744075073 x
```

You can notice that the values are close to the ones obtained above.

Plotting this linear model from scikit with the scatter plot the following graph was generated:



Then at the end, computing the Error for each model using the following code:

```
def error(pred, Final,n):
    error = pow((Final - pred), 2)
    return np.sum(error)/n
print("For normal linear regression: ",error(FLR,np.array(Variables[4]),32))
print("For Gradient Descent: ",error(FGD,np.array(Variables[4]),32))
print("For SCIkit learn linear regression:
",error(FSK,np.array(Variables[4]),32))
```

The error outputs after running each model:

```
Now computing the error for each one of the above methods.
For normal linear regression:  33.17778874065533
For Gradient Descent:  33.17882032836877
For SCIkit learn linear regression:  33.17778874065533
```

You can notice that the linear regression and Scikit are identical and slightly better than the gradient descent.