Department of Information Technology

ENCS5341

MACHINE LEARNING AND DATA SCIENCE

Homework #1

Name: Basheer Rjoub

ID: 1180291

Instructor: Dr. Yazan Abu Farha

9 December 2022

Q1)

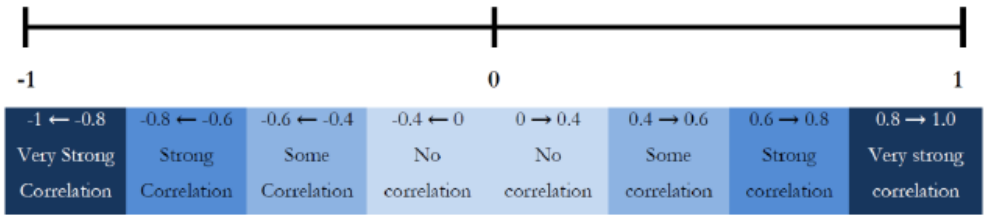
First part using excel and using the formula =ROUND (AVERAGE(Column),0) and rounding the missing values to the average of the current column.

Q2)

Here I used the Pearson Correlation to measure how strong the variables to the final exam's mark and which have the strongest relationship with the final exam.

From the following equation we can calculate the relationship's strength between an independent variable x and a dependent variable y and using the result to compute how strong the relationship is (which is r varies between 0 which indicates no relationship to 1 which is identical strong relationship).

$$r = \frac{N\sum xy - \sum x \sum y}{\sqrt{\left[N\sum x^2 - \left(\sum x\right)^2\right]\left[N\sum y^2 - \left(\sum y\right)^2\right]}},$$



First, using **Scipy** package from Python to compute it, then computing it manually. And from the code below we can see that the midterm has the highest **Pearson's** parameter so it should be the best parameter to represent the relationship to the final.

Code

```
import numpy as np

import matplotlib.pyplot as plt

from scipy import stats


hw1 =

[67,83,63,70,50,43,67,47,97,50,40,57,23,63,77,57,57,63,80,83,100,67,70,17,97,73,80,80,80,80,80,93]
```

```
hw2 =
[78,37,100,93,75,83,47,80,97,67,83,87,100,50,93,83,87,50,97,100,95,73,87,80,97,87,100,90,93,100,100,47]
mid = [32,25,34,35,20,26,22,26,30,21,18,28,29,26,29,28,28,27,40,57,56,34,32,34,44,25,46,43,48,41,47,38]
proj =
[87,91,92,92,76,55,86,94,92,76,94,76,75,75,71,72,72,85,85,98,98,71,95,85,95,85,98,94,94,94,94,92]
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]



print("HW1, Final:", stats.pearsonr(hw1, final).statistic)
print("HW2, Final:", stats.pearsonr(hw2, final).statistic)
print("Mid, Final:", stats.pearsonr(mid, final).statistic)
print("Proj, Final:", stats.pearsonr(proj, final).statistic)
```

Results

HW1, Final: 0.5604545951738554
HW2, Final: 0.5154555963860965
Mid, Final: 0.9177834900015012
Proj, Final: 0.49871770680707844

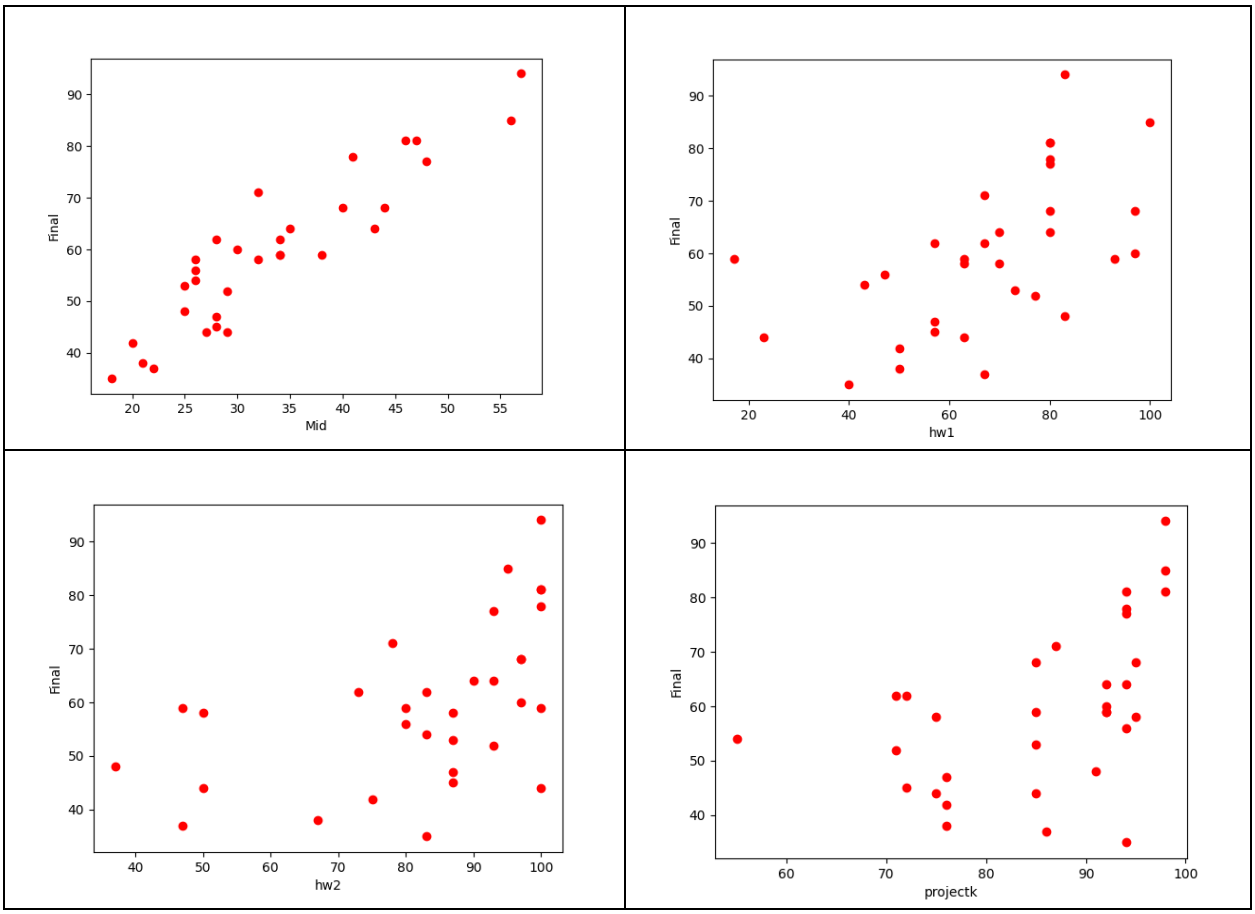
Calculations

For example, calculating the Mid and Final Pearson's correlation using the previous example we can find that:

$$r = \frac{32 * 67798 - (1069 * 1903)}{\sqrt{\left((32 * 38855 - \ 1{,}142{,}761) * (32 * 119913 - 3{,}621{,}409)\right)}}$$

$$r = \frac{135229}{147343} = 0.91778349$$

Also, another way to observe the best variable to predict the final mark, using the visual plots, as in the following plots we can observe clearly that the Mid's grades gives the best linear curve that can be formed with the least square errors.



Q3)

We have to find a linear regression equation that minimizes the square error, meaning that it will give us the least square error, and the equation will be at the form of:

$$f(x) = y = w_0 + w_1 x$$

Using the variable x = midterm mark, as we previously find that it is the best measure to predict the final exam's mark.

$$\underset{w_0,w_1}{\operatorname{argmin}} \sum_{i=1}^{n} \left(y_i - f(x_i)\right)^2$$

Taking the partial derivatives for both W0 and W1 then we have the two listed equations:

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

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

$$W_0 = \frac{1903}{32} - W_1 * \frac{1069}{32}$$

$$W_1 = \frac{67798 - \frac{1903 * 1069}{32}}{38855 - \frac{1069 * 1069}{32}} = \frac{4226}{3143.7} = 1.344$$

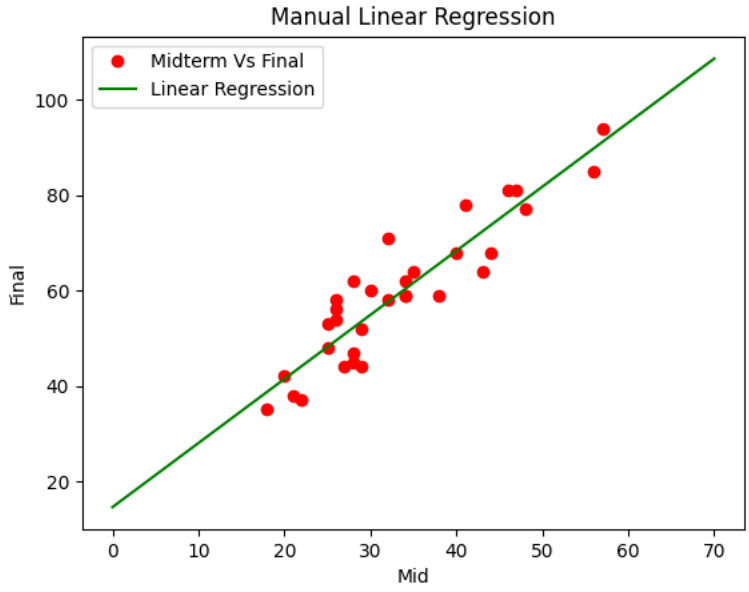
Then substituting W_1 value in W_0 $equation$ we get:

$$W_0 = 14.58$$

Which implies the following Linear regression equation:

$$f(x) = 14.58 + 1.344x$$

The plot for the Linear regression function that we implemented and the real values with the mid and final marks as in the following plot.



4) For the gradient decent we need to minimize the costs (errors) for every iteration the error will have the least squares error between the predicted and the actual value as in the following formula.

$$error = \frac{1}{n}\sum_{i=1}^{n}(y_i - (w_1 x_i + w_0))^2$$

Then taking the partial differential equations for the bias w_0 and weight w_1:

$$\frac{d(error)}{dw1} = \frac{-2}{n}\sum_{i=1}^{n} x_i(y_i - (w_1 x_i + w_0))$$

$$\frac{d(error)}{dw0} = \frac{-2}{n}\sum_{i=1}^{n}(y_i - (w_1 x_i + w_0))$$

Here we have n=32.

The following code implements the equations above and do many iterations to get the values for the weights w_0 and w_1:

Code

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


def gd(x, y, iterations, learning_rate = 0.0001):
```

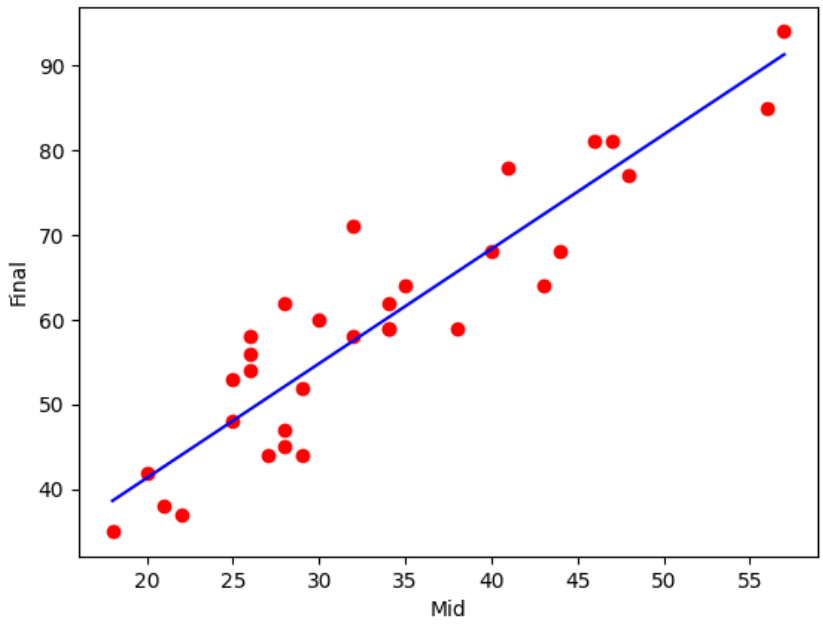
```
    w1 = 0.1
    w0 = 0.01
    costs = []
    weights = []
    prev_error = None

    for i in range(iterations):

        predict = (w1 * x) + w0
        err  = np.sum((y-predict)**2) / 32

        if prev_error and abs(prev_error-err)<=0.0000001:
            break

        prev_error = err
        costs.append(err)
        weights.append(w1)

        w1_der = -(2/32) * sum(x * (y-predict))
        w0_der = -(2/32) * sum(y-predict)

        w1 = w1 - (learning_rate * w1_der)
        w0 = w0 - (learning_rate * w0_der)

    return w1, w0

X =
np.array([32,25,34,35,20,26,22,26,30,21,18,28,29,26,29,28,28,27,40,57,56,34,32,34,4
4,25,46,43,48,41,47,38])
Y =
np.array([71,48,59,64,42,54,37,56,60,38,35,47,44,58,52,62,45,44,68,94,85,62,58,59,6
8,53,81,64,77,78,81,59])

w1, w0 = gd(X, Y, iterations=10000000)
```

```
print(f"W1: {w1}\nW0: {w0}")

F = w1*X + w0

plt.scatter(X, Y, marker='o', color='red')
plt.plot([min(X), max(X)], [min(F), max(F)], color='blue',markerfacecolor='green')
plt.xlabel("Mid")
plt.ylabel("Final")
plt.show()
```

Results

For 10m iteration

W1: 1.349616842038272
W0: 14.367307313681252



Q5)

Using a 0.1 of data to test the model and 0.9 to train it, we conclude to the following results:

W0: [14.57784825] W1: [[1.3520225]]

Which is quite close to the previous results we obtained by calculations.

Code

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split


x =
np.array([32,25,34,35,20,26,22,26,30,21,18,28,29,26,29,28,28,27,40,57,56,34,32,34,4
4,25,46,43,48,41,47,38]).reshape(-1, 1)
y =
np.array([71,48,59,64,42,54,37,56,60,38,35,47,44,58,52,62,45,44,68,94,85,62,58,59,6
8,53,81,64,77,78,81,59]).reshape(-1, 1)



SEED = 42
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size = 0.1)


reg = LinearRegression()
reg.fit(X_train, y_train)
print("W0: ", reg.intercept_)
print("W1: ", reg.coef_)
```

Using the following code to predict new values using the previous regression code we get the following:

```
W0:  [14.65950235]
W1:  [[1.33603188]]
Predicted for 10: [[28.01982114]]
Predicted for 15: [[34.69998053]]
Predicted for 20: [[41.38013993]]
Predicted for 30: [[54.74045871]]
Predicted for 14: [[33.36394866]]
```

Code

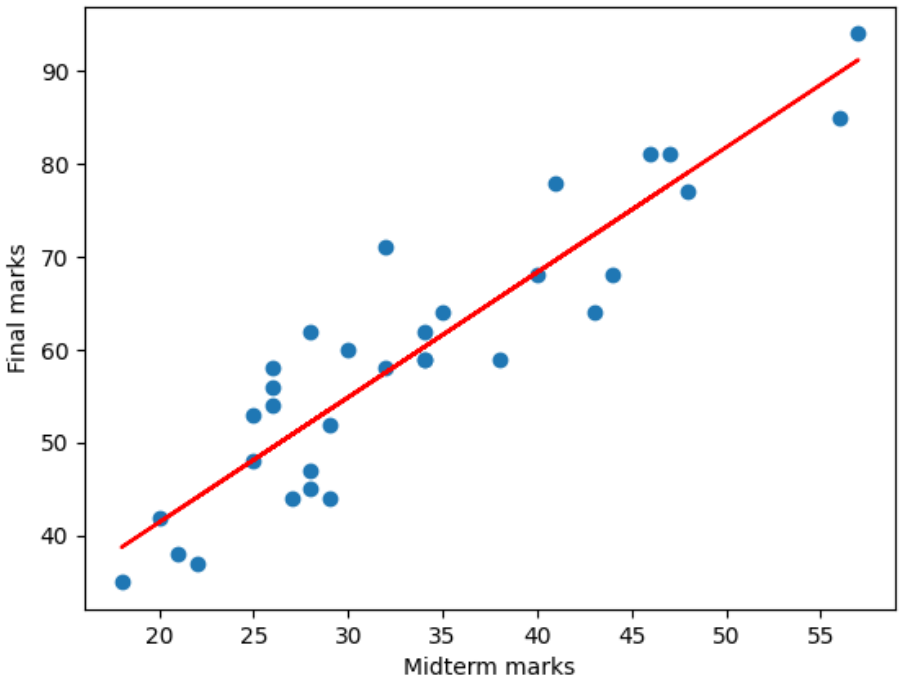
```
def predict(w1, w0, mid):
    return w0 + w1*mid
vals = [10, 15, 20, 30, 14]
for val in vals:
    print(f"Predicted for {val}:", predict(reg.coef_, reg.intercept_, val));
```

Code to plot the Linear Regression:

```
mid =
np.array([32,25,34,35,20,26,22,26,30,21,18,28,29,26,29,28,28,27,40,57,56,34,32,34,4
4,25,46,43,48,41,47,38])
final =
np.array([71,48,59,64,42,54,37,56,60,38,35,47,44,58,52,62,45,44,68,94,85,62,58,59,6
8,53,81,64,77,78,81,59])


#fit data to the model
model = LinearRegression(normalize=True)
x = np.expand_dims(mid, 1)
y = final

model.fit(x,y)

print("w1: ", model.coef_, "w0: ", model.intercept_)
```

```
plt.scatter(mid, final)
x = mid
f = model.coef_*x + model.intercept_
plt.plot(x, f, 'r')
plt.xlabel("Midterm marks")
plt.ylabel("Final marks")
plt.show()
```

For w1: 1.34423801 w0: 14.56279883497848 the Plot for Linear Regression:



The plot w0 and w1 are pretty close for what we have achieved in calculations.

Conclusion

After all we got the following three equations:

Calculated Linear Regression:

$$f(x) = 14.58 + 1.344\text{x}$$

Gradient Decent:

$$f(x) = 14.3673 + 1.3496x$$

Scikit-Learn:

$$f(x) = 14.5778 + 1.352x$$

I used the following code to get the square root difference of the predicted and the actual value to measure how good is the algorithm for the training values that we have.

```
# code to compare the three implementations of the regression
#using the least squares method
import numpy as np

X =
np.array([32,25,34,35,20,26,22,26,30,21,18,28,29,26,29,28,28,27,40,57,56,34,32,34,4
4,25,46,43,48,41,47,38])
Y =
np.array([71,48,59,64,42,54,37,56,60,38,35,47,44,58,52,62,45,44,68,94,85,62,58,59,6
8,53,81,64,77,78,81,59])

def calculated_LR(x):
    return 14.58 + 1.344*x

def gradient_decent(x):
    return 14.3673 + 1.3496*x

def scikit_learn(x):
    return 14.5778 + 1.352*x

def total_error(function, X, Y):
    s = 0
```

```
    for i in range(0, 32):
        err = abs(function(X[i]) - Y[i])
        sq_err = err**2
        s += sq_err
    return s/ 32


print("SR Manual Regression: ",total_error(calculated_LR, X, Y))
print("SR Gradient Decent: ",total_error(gradient_decent, X, Y))
print("SR Scikit Learn: ",total_error(scikit_learn, X, Y))
```

Results

```
SR Manual Regression:  33.229620000000004
SR Gradient Decent:  33.232621535
SR Scikit Learn:  33.31068824000002
```

As the values we got from results, it is clear that the first method is slightly better than the others, but the difference here can be ignored, because it is very small regarding the sum of the squared errors for tall the values.