FE-520 Assignment 2

Submission Requirement:

For all the problems in this assignment you need to design and use Python 3, output and present the results in nicely format.

Two ways to submit homework:

1. If you use Jupyter Notebook, go to File - Download as - click HTML(.html) and Python(.py), you need to submit both HTML file and Python to Canvas.

2. If you use Other IDE, screenshot your code and result into PDF, you need to submit both PDF file and Python to Canvas.

You are strongly encouraged to write comment for your code, because it is a convention to have your code documented all the time.

Do NOT copy and paste from others, all homework will be firstly checked by plagiarism detection tools.

1 Loop Practice: Gradient Decent (30 pts)

In statistics, linear regression is a linear approach to modelling the relationship between a dependent variable and one or more independent variables. Let X be the independent variable and Y be the dependent variable. We will define a linear relationship between these two variables as follows:

$$Y = wX + c$$

This is the equation for a line that you studied in high school. w is the slope of the line and c is the y intercept. Today we will use this equation to train our model with a given dataset and predict the value of Y for any given value of X.

Our challenge today is to determine the value of w and c, such that the line corresponding to those values is the best fitting line or gives the minimum error.

One way to solve this problem is to use **Gradient Decent** (The reference here contain more details of Gradient decent and sample code, try not use numpy in this question):

You don't need the knowledge of optimization to finish the question. Please just implement the following algorithm step by step.

The Algorithm of gradient decent to find w and c is :

1. Set initial variable. w=0 and c=0, Learning rate L=0.001, number of iterations.
2. Write a for loop, in this loop, go over all pair (x_i, y_i):
   (a) calculate $y_i^{pred} = x_i * w + c$
   (b) calculate $x_i(y_i^{pred} - y_i)$, and store it in list D_w
   (c) calculate $(y_i^{pred} - y_i)$, and store it in list D_c
3. calculate the average for list D_w and D_c, and assign the values to d_w and d_c respectively.
4. update m by: $w = w - L \times d_w$
5. update c by: $c = c - L \times d_c$
6. (Bonus 3 pts) What you have done above are one iteration of gradient descent, once you repeat from step 2 to 5 again and again, the w and c will be converged to the true value. Can you wrap them in big loop for 200 iteration?

you can test your result by using following dataset
input:
x = [0.18, 1.0, 0.92, 0.07, 0.85, 0.99, 0.87]
y = [109.85, 155.72, 137.66, 76.17, 139.75, 162.6, 151.77]
Output:
for one iteration
w = 0.10275336, c = 0.13336
for 200 iterations:
w = 17.72481065, c = 22.97599012903927

2 Control Flow Practice: Element-wise logical operations (30 pts)

1. (10 pts) Given two same-length lists of bool-type variables (True/False), you need to implement the element-wise `and`, `or`, and `not` operations. Element-wise means the operations are applied to each **pair** of two element, and the results form another list. In the following sample code, the first element in `result_and` is False because `x[0] and y[0]` equals `False` (True and False == False). Complete the following code to obtain the three lists.

```
x = [True, True, False, False, True, False, True]
y = [False, True, True, True, False, False, True]

<complete the code>

# return three lists: result_and, result_or, result_not_x which are
# respectively the result of  element-wise AND, element-wise OR,
# and element-wise NOT
# result_and is [False, True, False, False, False, False, True]
# result_or is [True, True, True, True, True, False, True]
# result_not_x is [False, False, True, True, False, True, False]
```

You need to use FOR loop to do this. A hint is to check `zip()` function which can zip two same-length lists to pairs. Check the link: `https://docs.python.org/3.7/library/functions.html#zip`.

2. (10 pts) Implement another operation: given a list of bool-type variables. If True takes 40% or more of the total number of element of this list, give True as the result, otherwise give False. For an example,

```
x = [True, True, False, False, True]
y = [True, False, False, False, False]
# your code gives True for x, as True takes 3/5 (60%>=40%)
# your code gives False for y, as True takes 1/5 (20%<40%)
```

3 Practice Dictionary (20 pts)

1. (2pts) Define a string of 'python is an interpreted dynamic programming language'
2. (3pts) Create a list (`list_A`) of single-character strings out of the above string in 1 (e.g., `'hello'->['h', 'e', 'l', 'l', 'o']`).
3. (8pts) Write a loop to count the number of each unique character (ignore spaces) into dictionary, where your keys are characters in the list A, and value is the count corresponding to each character. Your result should look like :
`{'h': 1, 'e': 1, 'l': 2, 'o': 1}`.
4. (7pts) Print the characters which shows up more than 1 time. (Hint: use loop and if statement).

4 Loop Practice: Sum (20 pts)

Write a loop for calculate the average of a list.

For example: if you have a list A = [1, 2, 3, 4, 5, 6], after your loop calculation, you need to get a total_num equals to 3.5.