Midterm-Rom-P

October 8, 2020

1 CP - Midterm - 2020

1.1 Instruction

- Modify this file to be Midterm-, e.g., Midterm-Chaklam-S.ipynb
- This exam accounts for 25% of the overall course assessment.
- This exam is open-booked; open-internet.
- You ARE NOT allowed to use sklearn or any libraries, unless stated.
- The completed exams shall be submitted at the Google Classroom
- All code should be **complemented with comments**, unless it's really obvious. **I and Joe reserve the privilege to give you zero for any part of the question where the benefit of doubt is not justified**

1.2 Examination Rules:

- For **offline** students, you may leave the room temporarily with the approval and supervision of the proctors. No extra time will be added to the exam in such cases.
- For **online** students, you are required to turn on your webcam during the entire period of the exam time
- Students will be allowed to leave at the **earliest 45 minutes** after the exam has started
- **All work should belong to you**. A student should NOT engage in the following activities which proctors reserve the right to interpret any of such act as academic dishonesty without questioning:
  - Chatting with any human beings physically or via online methods
  - Plagiarism of any sort, i.e., copying from internet sources or friends. **Both copee and copier shall be given a minimum penalty of zero mark for that particular question or the whole exam.**
- No make-up exams are allowed. Special considerations may be given upon a valid reason on unpredictable events such as accidents or serious sickness.

1.3 Question 1 (21 pts)

1). **The rabbit**: (5pts)

Once upon a time, there is a father rabbit lives in a far away jungle. Everyday, the father rabbit has to go out and find some carrots for his family. In his family there are mother rabbit, grampa rabbit, sister rabbit, and his son. In total there are 5 rabbits to feed. In one day, the adult rabbits (himself, mother rabbit and sister rabbit) will eat 3 carrots while the elderly eat 2 carrots and baby rabbit eat 1 carrot.

Unfortunately, the carrots are not easy to find. The father rabbit has to travel into the scary jungle and find some carrot then bring them back to the family before the sunset at 6PM.

- Every 1 km, the rabbit will find 3 carrots.
- The rabbit will use 1 hour to travel 1 km.

In summary, in order to find the least number of carrot for each day, the rabbit will have to use (3 + 3 + 3 + 2 + 1)/3 = 4 hours. This mean that he has to leave the house at the latest 10AM (4 hours for go out and another 4 for comming back).

This daily work has to be done exactly on time, leaving to late will cause whether his life or his family life. Would you like to help the rabbit?

[1]:
```
# print 'yes' to help the rabbit or 'no' to refuse the challenge. (if yes -> 1␣
↪pt)

#Father Rabbit 3 carrots
#Mother Rabbit 3 carrots
#Grandpa Rabbit 2 carrots
#Sister Rabbit 3 carrots
#Son Rabbit 1 carrots

#Leaves house at the latest 10AM
#yes
```

Good to hear that young programmer!!

What I have in mind is to build a clock that when the rabbit puts the number of (adult, elderly, young) rabbit, it will calculate how many hours is required for a travel. Of cause we have to make it as a function because the number of each rabbit type will change over the time.

- Write a function carrot that takes three integers as an input in follow this format (adult, elderly, young) (1pt)
- The function will calculate number of hour required for travelling a day. (1pt)
- The function will also calcualte the time to leave. Think of it as an alarm clock for leaving the house (1pt)
- The function will return a tuple (#hours, #time) (1pt)

[2]:
```
# Your code here

def carrot(adult, elderly, young):
    hours = (adult*3+elderly*2+young)/3
    leave = 18-2*hours
#     if(hours < 12):
#         print(leave)
#         hour_str = str(int(leave))+":"+str(int(60*(leave%1)))+"AM"
#     else:
#         hour_str = str(int(leave-12))+":"+str(int(60*(leave%1)))+"PM"
```

```python
    return (hours,leave)

carrot(3,1,1)
```

[2]: (4.0, 10.0)

2). **Print the shape**: (16pts)

- Write a function square that takes integer as an input. (1pt)
- The function will return a string of * in the shape of a square with both width and height equal to the input interger. (2pts)

[3]:
```python
'''
Level: 3
***
***
***

Level: 5
*****
*****
*****
*****
*****
'''
```

[3]: '\nLevel: 3\n***\n***\n***\n\nLevel: 5\n*****\n*****\n*****\n*****\n*****\n'

[4]:
```python
# Your code here
def square(n):
    #Create two loops to print "*"
    for j in range(n):
        for i in range(n):
            print("*",end="")
        print()

square(5)
```

```
*****
*****
*****
*****
*****
```

- Write a function triangle that takes integer as an input. (1pt)
- The function will return a string of * in the shape of a triangle with level equal to the input integer. (2pts)

[5]:
```
'''
Level: 3
  *
 **
***

Level: 5
    *
   **
  ***
 ****
*****
'''
```

[5]: '\nLevel: 3\n *\n **\n***\n\nLevel: 5\n   *\n   **\n ***\n ****\n*****\n'

[6]:
```
# Your code here

def triangle(n):
    #Create a loop for each row, each row will have its corresponding number of␣
↪spaces and number of "*"
    for i in range(n):
        spaces = " "*(n-i-1)
        ast = "*"*(i+1)
        print(spaces,end="")
        print(ast)


triangle(3)
```

```
  *
 **
***
```

- Write a function pyramid that takes integer as an input. (1pt)
- The function will return the string of * in the shape of a pyramid with level equal to the input interger. (2pts)

[7]:
```
'''
Level: 3
  *
 ***
*****

Level: 5
    *
   ***
  *****
```

```
 *******
*********
'''
```

[7]: ' \nLevel: 3 \n * \n ***\n*****\n\nLevel: 5\n   *\n   ***\n *****\n
*******\n*********\n'

[8]:
```python
# Your code here

def pyramid(n):
    #similar to the previous question, each row will have their respective␣
 ↪amount of spaces and asterisks
    for i in range(n):
        spaces = " "*(n-i-1)
        ast = "*"*(i*2+1)
        print(spaces,end="")
        print(ast,end="")
        print(spaces)

pyramid(5)
```

```
    *
   ***
  *****
 *******
*********
```

Now, let's combine the three algorithms into one single class. - Create a class named MyShape that can do the followings - Take two arguments during the class construction. The first one is an integer and the second one is a string. The names are level and shape (1pt) - Check the input arguments whether the interger is in the range of [1,10] and string is in the set of {'squ','tri','pyr'}. Raise a ValueError. (2pts) - Both attributes should be able to change via a set method **only**. set[attrName] (1pt) - Of cause, the set method should check the out of range too. (1pt) - To check the current setting, write a get method. get[attrName] (1pt) - Print the shape with method show. It should return the string of the current shape with the correct level (1pt)

[9]:
```python
'''
Example 1

>>> ms = MyShape(2,'tri')
>>> ms.show()
 *
**
>>> ms.setLevel(3)
>>> ms.setShape('squ')
>>> ms.show()
***
***
```

```
***
>>> ms.setShape('a')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: ..........
>>>
'''
```

[9]: '\nExample 1\n\n>>> ms = MyShape(2,\'tri\')\n>>> ms.show()\n *\n**\n>>> ms.setLevel(3)\n>>> ms.setShape(\'squ\')\n>>> ms.show()\n***\n***\n***\n>>> ms.setShape(\'a\')\nTraceback (most recent call last):\n File "<stdin>", line 1, in <module>\nValueError: …\n>>>\n'

[10]:
```
# Your code here
class MyShape():
    def __init__(self,level,shape):
        self.l = level
        self.s = shape
        self.checkLevel(self.l)
        self.checkShape(self.s)

    def checkLevel(self,level):
        if(level > 10 or level < 1):
            raise ValueError("Invalid Level")

    def checkShape(self,shape):
        if(shape not in ["squ","tri","pyr"]):
            raise ValueError("Invalid Shape")

    def setLevel(self,level):
        self.checkLevel(level)
        self.l = level

    def setShape(self,shape):
        self.checkShape(shape)
        self.s = shape

    def getLevel(self):
        return self.l

    def getShape(self):
        return self.s

    def pyramid(self,n):
        for i in range(n):
            spaces = " "*(n-i-1)
            ast = "*"*(i*2+1)
```

```python
            print(spaces,end="")
            print(ast,end="")
            print(spaces)

    def triangle(self,n):
        for i in range(n):
            spaces = " "*(n-i-1)
            ast = "*"*(i+1)
            print(spaces,end="")
            print(ast)

    def square(self,n):
        for j in range(n):
            for i in range(n):
                print("*",end="")
            print()

    def show(self):
        if(self.s == 'squ'):
            self.square(self.l)
        elif(self.s == 'tri'):
            self.triangle(self.l)
        elif(self.s == 'pyr'):
            self.pyramid(self.l)
        else:
            print("HI")


ms = MyShape(3,'pyr')
print(ms.getShape())
print(ms.getLevel())
ms.show()
ms.setShape('squ')
ms.setLevel(10)
print(ms.getShape())
print(ms.getLevel())
ms.show()
```

```
pyr
3
  *
 ***
*****
squ
10
**********
**********
```

****
****
****
****
****
****
****
****

1.4 Question 2 (10 pts)

2). **ML Skill**

$$y = ax + b$$

The above equation is your favorite linear equation where a,b are the constant value indicate the slope and the offset of the line in the graph.

We all know given and two points.

$$(x_1, y_1)(x_2, y_2)$$

you can find a,b very easy using Geometry

$$a = \frac{y_2 - y_1}{x_2 - x_1}$$
$$b = y_i - ax_i$$

Since we have learnt that using LinearRegression can find the value of the a,b too.

Now, do the followings.

- Write a function drawLine that takes two tuples as inputs.
- Calculate a,b using Geometry.
- Draw the first graph with scatter on the given two points and a line.
- Calculate a,b using LinearRegression with Batch Gradient Descent.
  - Generate 1000 sample data along the line.
  - Regress on the data using LinearRegression-BatchGradientDescent
- Draw the second graph with scatter on the given two points and a line.
- Does both method yeild the same outcome? Which method runs faster? (use timeit)
- What will happen if the data is normalize first? (draw another graph and timeit)
- What will happen if the data is standardize first? (draw another graph and timeit)

[11]:
```
# Your code here
import matplotlib.pyplot as plt
import numpy as np

class LinearRegressionModel:
#1. hypothesis function
    def h(self, X, theta):
        hypothesis = X@theta
        return hypothesis
```

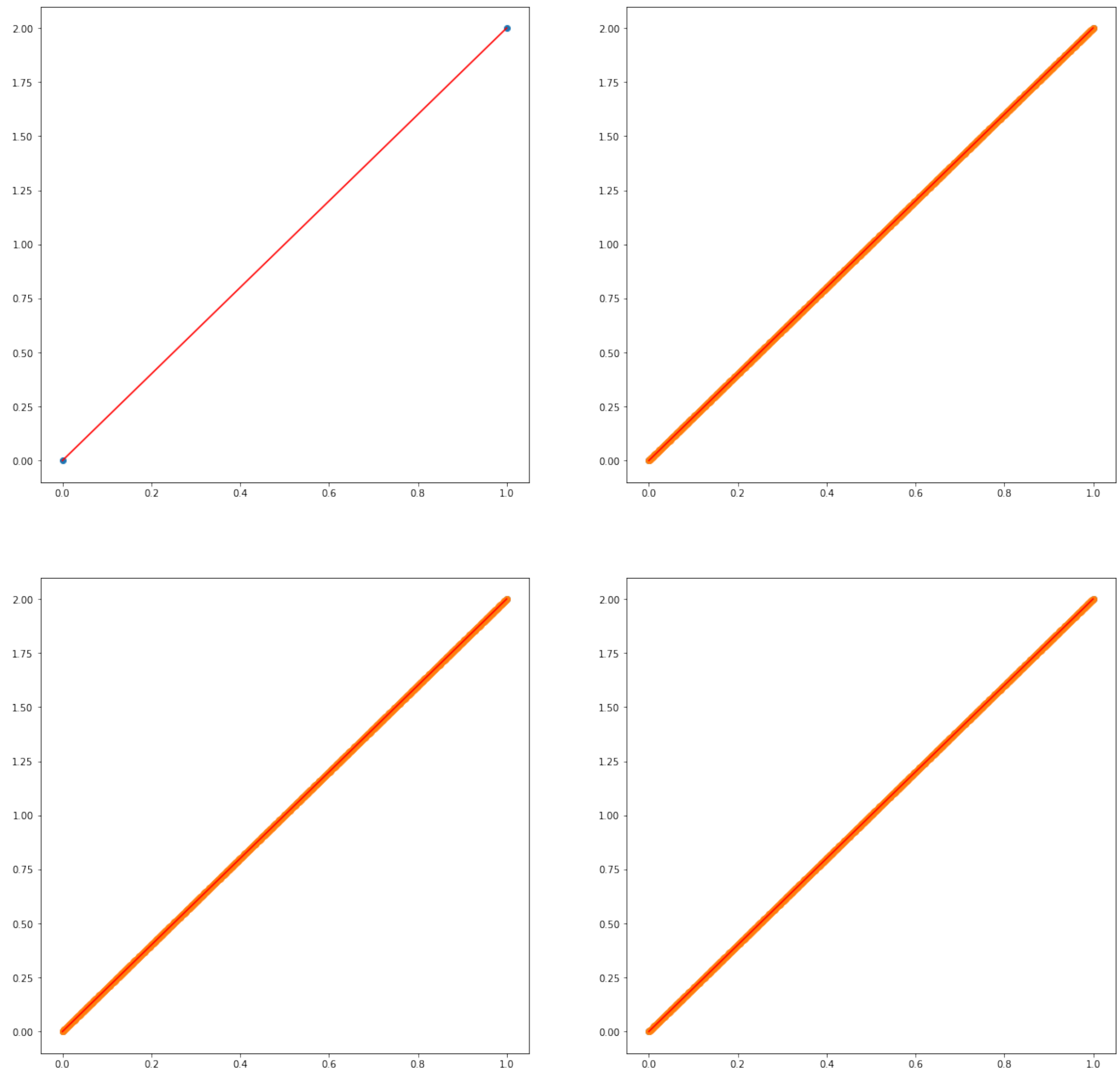
```python
#2. cost function
    def cost(self, X, y, theta, average = False):
        #expects X to be a design matrix, y to be a column vector and theta to be a column vector
        if(average == False):
            J = 1/2*(self.h(X,theta)-y).T@(self.h(X,theta)-y)
        else:
            J = 1/(2*X.shape[0])*(self.h(X,theta)-y).T@(self.h(X,theta)-y)
        return J

#3. gradient function
    def gradient(self, X, y, theta, average = False):
        if(average == False):
            dJ = X.T@(self.h(X,theta)-y)
        else:
            dJ = X.T@(self.h(X,theta)-y)/(X.shape[0])
        return dJ

#4. batch gradient descent
    def batch_gd(self, X, y, initial_theta, max_iteration, alpha, tolerance = 0,average = False):
        cost = []
        theta = initial_theta
        iteration = 0
        cost.append(self.cost(X,y,theta,average))
        for n in range(max_iteration):
            gradient = self.gradient(X,y,theta,average)
            theta = theta - alpha*gradient
            cost.append(self.cost(X,y,theta,average))
            iteration += 1
            if(self.mean_squared_error(X,y,theta) < tolerance):
                return theta,cost,iteration
        cost = np.array(cost)
        return theta,cost,iteration

#5. normal equation
    def normal_equation(self, X, y):
        theta = np.linalg.inv(X.T@X)@X.T@y
        return theta

#5. predict
    def predict(self,X,theta):
        prediction = self.h(X,theta)
        return prediction

#6. score/error calculation
```

```
    def mean_squared_error(self,X,y,theta):
        mse = self.cost(X,y,theta,average = True)*2
        return mse

#7. plotting cost
    def plot_cost(self,cost, iteration_no):
        iteration_series = np.arange(0,iteration_no+1)
        ax = plt.axes()
        ax.plot(iteration_series, cost)

def drawLine(one,two):
    # drawLine function as stated in the problem
    x,y = zip(one,two)
    print("timeit for using the equations")
    #using timeit, we can see how long it takes to perform theese operations
    a = (y[1]-y[0])/(x[1]-x[0])
    b = y[0]-a*x[0]
    %timeit (y[1]-y[0])/(x[1]-x[0]), y[0]-a*x[0]

    #Plot for equation
    fig,ax = plt.subplots(2,2,figsize = (20,20))
    ax[0,0].scatter(x,y)
    ax[0,0].plot(x,y,'r')

    x_sample = np.linspace(np.min(x),np.max(x),1000)
    y_sample = x_sample*a+b

    LR = LinearRegressionModel()
    iterations = 1000
    alpha = 0.001
    initial_theta = np.zeros(2)
    x_sample_inserted = np.insert(x_sample[:,np.newaxis],0,1,axis=1)

    #using batch gradient descent on data which hasn't been standardized or␣
 ↪normalized
    print("timeit for batch gradient descent")
    %timeit LR.
 ↪batch_gd(x_sample_inserted,y_sample,initial_theta,iterations,alpha)
    theta,cost,iteration = LR.
 ↪batch_gd(x_sample_inserted,y_sample,initial_theta,iterations,alpha,tolerance=1e-7)
    y_pred = LR.predict(x_sample_inserted,theta)
    #LR.plot_cost(cost,iteration)
    print("iteration:",iteration)

    MSE = LR.mean_squared_error(x_sample_inserted,y_sample,theta)
    print("MSE =",MSE)
```

```
    ax[0,1].scatter(x,y)
    ax[0,1].plot(x,y,'r')
    ax[0,1].scatter(x_sample,y_pred)

    mini = np.min(x_sample)
    maxi = np.max(x_sample)
    x_norm = (x_sample-mini)/(maxi-mini)

    x_norm_inserted = np.insert(x_norm[:,np.newaxis],0,1,axis=1)

    iterations2 = 1000
    alpha2 = 0.001
    initial_theta2 = np.zeros(2)

    print("timeit for normalized data")
    %timeit LR.
↪batch_gd(x_norm_inserted,y_sample,initial_theta2,iterations2,alpha2)
    theta2,cost2,iteration2 = LR.
↪batch_gd(x_norm_inserted,y_sample,initial_theta2,iterations2,alpha2,tolerance=1e-7)
    y_pred2 = LR.predict(x_norm_inserted,theta2)
    print("iteration2",iteration2)
    MSE2 = LR.mean_squared_error(x_norm_inserted,y_sample,theta2)
    print("MSE =",MSE2)

    ax[1,0].scatter(x,y)
    ax[1,0].plot(x,y,'r')
    ax[1,0].scatter(x_norm*(maxi-mini)+mini,y_pred2)

    mean = np.mean(x_sample)
    std = np.std(x_sample)
    x_stan = (x_sample-mean)/std

    x_stan_inserted = np.insert(x_stan[:,np.newaxis],0,1,axis=1)

    iterations3= 1000
    alpha3 = 0.001
    initial_theta3 = np.zeros(2)

    print("timeit for standardized data")
    %timeit LR.
↪batch_gd(x_stan_inserted,y_sample,initial_theta3,iterations3,alpha3,average=True)
    theta3,cost3,iteration3 = LR.
↪batch_gd(x_stan_inserted,y_sample,initial_theta3,iterations3,alpha3,tolerance=1e-7)
    y_pred3 = LR.predict(x_stan_inserted,theta3)
    print("iteration3",iteration3)
    MSE3 = LR.mean_squared_error(x_stan_inserted,y_sample,theta3)
    print("MSE =",MSE3)
```

```
    ax[1,1].scatter(x,y)
    ax[1,1].plot(x,y,'r')
    ax[1,1].scatter(x_stan*std+mean,y_pred3)



    print("slope =",a,"intercept =",b)

drawLine((1,2),(-0,0))
```

```
timeit for using the equations
212 ns ± 13.3 ns per loop (mean ± std. dev. of 7 runs, 1000000 loops each)
timeit for batch gradient descent
28.2 ms ± 1.26 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
iteration: 107
MSE = 9.514767563793411e-08
timeit for normalized data
26 ms ± 260 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)
iteration2 107
MSE = 9.514767563793411e-08
timeit for standardized data
29 ms ± 1.38 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
iteration3 1
MSE = 6.848366140997965e-32
slope = 2.0 intercept = 0.0
```



1.5 Question 3 (69 pts)

1). **Exploratory Data Analysis**: - Load the data "howlongwelive.csv" to pandas and print the first 5 and last 5 rows of data (1 or 0pt)

- Print the shape, feature names, and summary (describe) of the data (1 or 0pt)
- Check whether there is missing data. (1 or 0pt)
- Fix all missing data using means or mode (1 or 0pt)
- Since Hepatatis B has a lot of nans, and highly correlate with Diptheria, simply drop column Hepatatis. Also drop column Population since there are way too many nans (1 or 0pt)
- If there are any features which are string and you want to use them as features, we need to convert them to int or float. For now, convert Status to 0 or 1 (1 or 0pt)

- Rename column thinness_1-19_years to thinness_10-19_years (1 or 0pt)
- Perform a groupby country and plot their life expectancy. Which country has the lowest/highest life expectancy? (1 or 0pt)
- Plot average life expectancy of developed country vs. developing country. (1 or 0pt)
- Perform a t-test of life expectancy between developed and developing countries. Is the result significant? (1 or 0pt)
- Perform a pairplot to see which features are likely to have strong predictive power for life expectancy. Identify the most 3 important features. (1 or 0pt)
- Perform a histogram of life expectancy. Is it normal? (1 or 0pt)

2). **Regression** - Prepare your X and y into Numpy array (you have to map from Pandas to numpy). For X, prepare two versions of them. For first X_selected, you have to choose the most 3 important features from above, and for second X_all, simply use all features (you may want to omit Country since they are categorical). Set y to life expectancy. (1 or 0pt)

- Perform standardization using Numpy way (NOT sklearn way). (1 or 0pt)
- Perform train-test split by using Numpy way (NOT sklearn way). Use test size of 0.3. (1 or 0pt)
- Perform assertion whether your splitting is correct accordingly (1 or 0pt)
- Write a class Regression(X, y, grad_method, max_iter, alpha, tol, decay, decay_iter, decay_rate, stop_delay_counter, verbose, lam, poly, poly_deg) that can perform the followings:
  - Mini-batch, Stochastic, and Batch Gradient Descent (each 2pts)
  - Polynomial of degree k (2 or 0pt)
  - Decay learning rate (1 or 0pt)
    * Decay learning rate is a learning rate that becomes smaller after certian iteration. For example, after 5 iterations, the learning rate will reduce to 95% of the current learning rate.
    * To implement it, simply multiply current learning rate with some constant decay_rate. For now, set it to 0.9
  - Regularization with ridge (2 or 0pt)
  - Must have at least four methods for fit() (i.e., for finding weights) predict() (i.e., for predicting X_test data), score() (i.e., for returning r^2 score), and mse() (return mse) (each 1pt)
  - Accepts X, y, grad_method (default set to "batch"), alpha (learning rate), max_iter, tol, decay (whether to use decay learning rate; default set to False), decay_iter (after how many iterations will the decay apply), stop_delay_counter (this is the maximum number of times that decay the learning rate), verbose (default is set to False, whether model will display the Cost for each iteration), lam (this is the ridge regularization parameter), poly (default is set to False), and poly_deg (default is set to 2) (each 1/13pt)
- Create the following 3 models **from your class** (For any unspecified parameters, feel free to use any :D)

1. For the first model, transform your feature using polynomial degree 3, then perform linear regression with batch gradient descent with early stopping of tol 1e-3 (1 or 0pt)
2. For the second model, perform linear regression with mini-batch gradient desent with early stopping of tol 1e-3 (1 or 0pt)
3. For the third model, perform ridge regression with stochastic gradient desent with early stopping of tol 1e-3 and decay set to True and lam to 1e-4 (1 or 0pt)

- Create Lasso model from Sklearn with default parameters (1 or 0pt)
- For these four models, using two different versions of X, perform a cross validation of 10 folds, comparing the four models * two versions of X. Here you should implement cross validation. Report which one is the best candidate model (3pts for implement from scratch or 1pt for using sklearn)
  - Recall that in a 10 folds cross validation, you split your data into 10 even pieces. Then you run 10 iterations, where in each iteration, you pick 1 of this piece as the validation set, and the rest as training set. Once you reach the 10th iteration, you would have already exhaust all the 10 pieces as validation set.
- Using the best model, fit again with the training data. Plot the weights using bar charts along the feature names. Before you actually plot the weights, we need to multiply these weights by their feature standard deviation, so to reduce these weights to same unit of measure. Interpret these weights and what they imply. (For those who are curious why we need to multiply with std, you may read this > https://scikit-learn.org/stable/auto_examples/inspection/plot_linear_model_coefficient_interpretation.html#interpreti coefficients-scale-matters (2 or 0pt)
- Perform predictions on testing data. Print adjusted r^2 and mse. (1 or 0pt)
- Plot the predicted values against actual values (1 or 0pt)

3). **Classification**

- Change your y to discrete value. Here split y into three class, {0, 1, 2}, where 0 belongs to low life expectancy group, and 2 for the high life expectancy group. (1 or 0pt)
- Write a class for multinomial logistic regression with stochastic gradient descent. Must have at least six methods for fit() (i.e., for finding weights) predict() (i.e., for predicting X_test data), accuracy() (i.e., for returning accuracy score), recall(), precision(), and f1() (each 1pt)
- Using the best X_train of the two suggested by the cross validation step, fit the data with your class. (1 or 0pt)
- Perform predictions on testing data. Print accuracy, recall, precision, and f1_score from your class. (1 or 0pt)
- Plot the decision boundary with the X_test data. To plot this, you may want to choose only 2 features. (1 or 0pt)

4). **Final verdict**

- Attempt to do whatever ways - including sklearn or scratch - or change your features, or do feature enginnering such that your mse is lowest possible. (0 to 5pts - following class normal distributions)

1). **Exploratory Data Analysis**: - Load the data ”howlongwelive.csv” to pandas and print the first 5 and last 5 rows of data (1 or 0pt)

- Print the shape, feature names, and summary (describe) of the data (1 or 0pt)
- Check whether there is missing data. (1 or 0pt)
- Fix all missing data using means or mode (1 or 0pt)
- Since Hepatatis B has a lot of nans, and highly correlate with Diptheria, simply drop column Hepatatis. Also drop column Population since there are way too many nans (1 or 0pt)
- If there are any features which are string and you want to use them as features, we need to convert them to int or float. For now, convert Status to 0 or 1 (1 or 0pt)
- Rename column thinness_1-19_years to thinness_10-19_years (1 or 0pt)
- Perform a groupby country and plot their life expectancy. Which country has the lowest/highest life expectancy? (1 or 0pt)
- Plot average life expectancy of developed country vs. developing country. (1 or 0pt)
- Perform a t-test of life expectancy between developed and developing countries. Is the result significant? (1 or 0pt)
- Perform a pairplot to see which features are likely to have strong predictive power for life expectancy. Identify the most 3 important features. (1 or 0pt)
- Perform a histogram of life expectancy. Is it normal? (1 or 0pt)

[12]:
```python
# Your code here
import pandas as pd

data = pd.read_csv("howlongwelive.csv")
```

[13]:
```python
data.head()
```

[13]:
```
       Country  Year      Status  Life expectancy   Adult Mortality  \
0  Afghanistan  2015  Developing             65.0             263.0
1  Afghanistan  2014  Developing             59.9             271.0
2  Afghanistan  2013  Developing             59.9             268.0
3  Afghanistan  2012  Developing             59.5             272.0
4  Afghanistan  2011  Developing             59.2             275.0

   infant deaths  Alcohol  percentage expenditure  Hepatitis B  Measles   …  \
0             62     0.01               71.279624         65.0     1154   …
1             64     0.01               73.523582         62.0      492   …
2             66     0.01               73.219243         64.0      430   …
3             69     0.01               78.184215         67.0     2787   …
4             71     0.01                7.097109         68.0     3013   …

   Polio  Total expenditure  Diphtheria    HIV/AIDS         GDP  Population  \
0    6.0               8.16        65.0         0.1  584.259210  33736494.0
```

```
1   58.0                8.18          62.0          0.1  612.696514    327582.0
2   62.0                8.13          64.0          0.1  631.744976  31731688.0
3   67.0                8.52          67.0          0.1  669.959000   3696958.0
4   68.0                7.87          68.0          0.1   63.537231   2978599.0

    thinness  1-19 years   thinness 5-9 years  \
0                   17.2                 17.3
1                   17.5                 17.5
2                   17.7                 17.7
3                   17.9                 18.0
4                   18.2                 18.2

   Income composition of resources  Schooling
0                            0.479       10.1
1                            0.476       10.0
2                            0.470        9.9
3                            0.463        9.8
4                            0.454        9.5

[5 rows x 22 columns]
```

[14]:
```
data.tail()
```

[14]:
```
        Country  Year      Status  Life expectancy   Adult Mortality  \
2933  Zimbabwe  2004  Developing              44.3             723.0
2934  Zimbabwe  2003  Developing              44.5             715.0
2935  Zimbabwe  2002  Developing              44.8              73.0
2936  Zimbabwe  2001  Developing              45.3             686.0
2937  Zimbabwe  2000  Developing              46.0             665.0

      infant deaths  Alcohol  percentage expenditure  Hepatitis B  Measles   \
2933             27     4.36                     0.0         68.0        31
2934             26     4.06                     0.0          7.0       998
2935             25     4.43                     0.0         73.0       304
2936             25     1.72                     0.0         76.0       529
2937             24     1.68                     0.0         79.0      1483

      …  Polio  Total expenditure  Diphtheria    HIV/AIDS         GDP  \
2933  …   67.0               7.13        65.0        33.6  454.366654
2934  …    7.0               6.52        68.0        36.7  453.351155
2935  …   73.0               6.53        71.0        39.8   57.348340
2936  …   76.0               6.16        75.0        42.1  548.587312
2937  …   78.0               7.10        78.0        43.5  547.358879

      Population   thinness  1-19 years   thinness 5-9 years  \
2933  12777511.0                    9.4                  9.4
2934  12633897.0                    9.8                  9.9
```

```
2935    125525.0                     1.2                     1.3
2936  12366165.0                     1.6                     1.7
2937  12222251.0                    11.0                    11.2

      Income composition of resources  Schooling
2933                            0.407        9.2
2934                            0.418        9.5
2935                            0.427       10.0
2936                            0.427        9.8
2937                            0.434        9.8

[5 rows x 22 columns]
```

[15]:
```python
print("shape:",data.shape)
print("features:",data.columns)
data.describe()
```

```
shape: (2938, 22)
features: Index(['Country', 'Year', 'Status', 'Life expectancy ', 'Adult
Mortality',
       'infant deaths', 'Alcohol', 'percentage expenditure', 'Hepatitis B',
       'Measles ', ' BMI ', 'under-five deaths ', 'Polio', 'Total expenditure',
       'Diphtheria ', ' HIV/AIDS', 'GDP', 'Population',
       ' thinness  1-19 years', ' thinness 5-9 years',
       'Income composition of resources', 'Schooling'],
      dtype='object')
```

[15]:
```
              Year  Life expectancy   Adult Mortality  infant deaths  \
count  2938.000000        2928.000000      2928.000000    2938.000000
mean   2007.518720          69.224932       164.796448      30.303948
std       4.613841           9.523867       124.292079     117.926501
min    2000.000000          36.300000         1.000000       0.000000
25%    2004.000000          63.100000        74.000000       0.000000
50%    2008.000000          72.100000       144.000000       3.000000
75%    2012.000000          75.700000       228.000000      22.000000
max    2015.000000          89.000000       723.000000    1800.000000

           Alcohol  percentage expenditure  Hepatitis B       Measles   \
count  2744.000000             2938.000000  2385.000000    2938.000000
mean      4.602861              738.251295    80.940461    2419.592240
std       4.052413             1987.914858    25.070016   11467.272489
min       0.010000                0.000000     1.000000       0.000000
25%       0.877500                4.685343    77.000000       0.000000
50%       3.755000               64.912906    92.000000      17.000000
75%       7.702500              441.534144    97.000000     360.250000
max      17.870000            19479.911610    99.000000  212183.000000
```

```
             BMI  under-five deaths        Polio  Total expenditure  \
count  2904.000000        2938.000000  2919.000000         2712.00000
mean     38.321247          42.035739    82.550188            5.93819
std      20.044034         160.445548    23.428046            2.49832
min       1.000000           0.000000     3.000000            0.37000
25%      19.300000           0.000000    78.000000            4.26000
50%      43.500000           4.000000    93.000000            5.75500
75%      56.200000          28.000000    97.000000            7.49250
max      87.300000        2500.000000    99.000000           17.60000

        Diphtheria     HIV/AIDS            GDP    Population  \
count  2919.000000  2938.000000    2490.000000  2.286000e+03
mean     82.324084     1.742103    7483.158469  1.275338e+07
std      23.716912     5.077785   14270.169342  6.101210e+07
min       2.000000     0.100000       1.681350  3.400000e+01
25%      78.000000     0.100000     463.935626  1.957932e+05
50%      93.000000     0.100000    1766.947595  1.386542e+06
75%      97.000000     0.800000    5910.806335  7.420359e+06
max      99.000000    50.600000  119172.741800  1.293859e+09

         thinness  1-19 years   thinness 5-9 years  \
count            2904.000000          2904.000000
mean                4.839704             4.870317
std                 4.420195             4.508882
min                 0.100000             0.100000
25%                 1.600000             1.500000
50%                 3.300000             3.300000
75%                 7.200000             7.200000
max                27.700000            28.600000

        Income composition of resources    Schooling
count                       2771.000000  2775.000000
mean                           0.627551    11.992793
std                            0.210904     3.358920
min                            0.000000     0.000000
25%                            0.493000    10.100000
50%                            0.677000    12.300000
75%                            0.779000    14.300000
max                            0.948000    20.700000
```

[16]:
```python
print("amount of missing data for each column:")
print(np.sum(data.isnull()))
```

```
amount of missing data for each column:
Country                              0
Year                                 0
Status                               0
```

```
Life expectancy                     10
Adult Mortality                     10
infant deaths                        0
Alcohol                            194
percentage expenditure               0
Hepatitis B                        553
Measles                              0
 BMI                                34
under-five deaths                    0
Polio                               19
Total expenditure                  226
Diphtheria                          19
 HIV/AIDS                            0
GDP                                448
Population                         652
 thinness  1-19 years               34
 thinness 5-9 years                 34
Income composition of resources    167
Schooling                          163
dtype: int64
```

[17]:
```python
data.fillna(data.mean(),inplace=True)
print("No more missing data:")
print(np.sum(data.isnull()))
```

```
No more missing data:
Country                            0
Year                               0
Status                             0
Life expectancy                    0
Adult Mortality                    0
infant deaths                      0
Alcohol                            0
percentage expenditure             0
Hepatitis B                        0
Measles                            0
 BMI                               0
under-five deaths                  0
Polio                              0
Total expenditure                  0
Diphtheria                         0
 HIV/AIDS                          0
GDP                                0
Population                         0
 thinness  1-19 years              0
 thinness 5-9 years                0
Income composition of resources    0
Schooling                          0
```

```
dtype: int64
```

[18]:
```python
data.drop(columns = ["Hepatitis B","Population"],inplace=True)
```

[19]:
```python
data["Status"]
```

[19]:
```
0       Developing
1       Developing
2       Developing
3       Developing
4       Developing
           …
2933    Developing
2934    Developing
2935    Developing
2936    Developing
2937    Developing
Name: Status, Length: 2938, dtype: object
```

[20]:
```python
labels, levels = pd.factorize(data["Status"])
data["Status"] = labels
```

[21]:
```python
data
```

[21]:
```
          Country  Year  Status  Life expectancy   Adult Mortality  \
0     Afghanistan  2015       0              65.0             263.0
1     Afghanistan  2014       0              59.9             271.0
2     Afghanistan  2013       0              59.9             268.0
3     Afghanistan  2012       0              59.5             272.0
4     Afghanistan  2011       0              59.2             275.0
…             …   …     …               …                …
2933     Zimbabwe  2004       0              44.3             723.0
2934     Zimbabwe  2003       0              44.5             715.0
2935     Zimbabwe  2002       0              44.8              73.0
2936     Zimbabwe  2001       0              45.3             686.0
2937     Zimbabwe  2000       0              46.0             665.0

      infant deaths  Alcohol  percentage expenditure  Measles    BMI   \
0                62     0.01               71.279624     1154   19.1
1                64     0.01               73.523582      492   18.6
2                66     0.01               73.219243      430   18.1
3                69     0.01               78.184215     2787   17.6
4                71     0.01                7.097109     3013   17.2
…             …       …                      …        …     …
2933             27     4.36                0.000000       31   27.1
2934             26     4.06                0.000000      998   26.7
2935             25     4.43                0.000000      304   26.3
```

```
2936               25     1.72                 0.000000       529   25.9
2937               24     1.68                 0.000000      1483   25.5

      under-five deaths   Polio  Total expenditure  Diphtheria    HIV/AIDS  \
0                    83     6.0               8.16        65.0         0.1
1                    86    58.0               8.18        62.0         0.1
2                    89    62.0               8.13        64.0         0.1
3                    93    67.0               8.52        67.0         0.1
4                    97    68.0               7.87        68.0         0.1
…                   …    …                 …          …         …
2933                 42    67.0               7.13        65.0        33.6
2934                 41     7.0               6.52        68.0        36.7
2935                 40    73.0               6.53        71.0        39.8
2936                 39    76.0               6.16        75.0        42.1
2937                 39    78.0               7.10        78.0        43.5

             GDP   thinness  1-19 years   thinness 5-9 years  \
0     584.259210                  17.2                 17.3
1     612.696514                  17.5                 17.5
2     631.744976                  17.7                 17.7
3     669.959000                  17.9                 18.0
4      63.537231                  18.2                 18.2
…         …                    …                 …
2933  454.366654                   9.4                  9.4
2934  453.351155                   9.8                  9.9
2935   57.348340                   1.2                  1.3
2936  548.587312                   1.6                  1.7
2937  547.358879                  11.0                 11.2

      Income composition of resources  Schooling
0                               0.479       10.1
1                               0.476       10.0
2                               0.470        9.9
3                               0.463        9.8
4                               0.454        9.5
…                               …        …
2933                            0.407        9.2
2934                            0.418        9.5
2935                            0.427       10.0
2936                            0.427        9.8
2937                            0.434        9.8

[2938 rows x 20 columns]
```

[22]:

```
#the name of the columns has to be found by using data.columns first
data.rename(columns={" thinness  1-19 years":
 ↪"thinness_10-19_years"},inplace=True)
```

[23]:
```
data
```

[23]:
```
          Country  Year  Status  Life expectancy   Adult Mortality  \
0     Afghanistan  2015       0             65.0             263.0
1     Afghanistan  2014       0             59.9             271.0
2     Afghanistan  2013       0             59.9             268.0
3     Afghanistan  2012       0             59.5             272.0
4     Afghanistan  2011       0             59.2             275.0
…             …   …     …               …                …
2933     Zimbabwe  2004       0             44.3             723.0
2934     Zimbabwe  2003       0             44.5             715.0
2935     Zimbabwe  2002       0             44.8              73.0
2936     Zimbabwe  2001       0             45.3             686.0
2937     Zimbabwe  2000       0             46.0             665.0

      infant deaths  Alcohol  percentage expenditure  Measles    BMI   \
0                62     0.01               71.279624      1154   19.1
1                64     0.01               73.523582       492   18.6
2                66     0.01               73.219243       430   18.1
3                69     0.01               78.184215      2787   17.6
4                71     0.01                7.097109      3013   17.2
…             …      …                     …       …    …
2933             27     4.36                0.000000        31   27.1
2934             26     4.06                0.000000       998   26.7
2935             25     4.43                0.000000       304   26.3
2936             25     1.72                0.000000       529   25.9
2937             24     1.68                0.000000      1483   25.5

      under-five deaths   Polio  Total expenditure  Diphtheria    HIV/AIDS  \
0                    83     6.0               8.16        65.0         0.1
1                    86    58.0               8.18        62.0         0.1
2                    89    62.0               8.13        64.0         0.1
3                    93    67.0               8.52        67.0         0.1
4                    97    68.0               7.87        68.0         0.1
…                  …    …                …          …         …
2933                 42    67.0               7.13        65.0        33.6
2934                 41     7.0               6.52        68.0        36.7
2935                 40    73.0               6.53        71.0        39.8
2936                 39    76.0               6.16        75.0        42.1
2937                 39    78.0               7.10        78.0        43.5

             GDP  thinness_10-19_years   thinness 5-9 years  \
0     584.259210                  17.2                 17.3
1     612.696514                  17.5                 17.5
2     631.744976                  17.7                 17.7
3     669.959000                  17.9                 18.0
4      63.537231                  18.2                 18.2
```

```
…          …                             …                         …
2933   454.366654                      9.4                       9.4
2934   453.351155                      9.8                       9.9
2935    57.348340                      1.2                       1.3
2936   548.587312                      1.6                       1.7
2937   547.358879                     11.0                      11.2

      Income composition of resources  Schooling
0                               0.479       10.1
1                               0.476       10.0
2                               0.470        9.9
3                               0.463        9.8
4                               0.454        9.5
…                                 …          …
2933                            0.407        9.2
2934                            0.418        9.5
2935                            0.427       10.0
2936                            0.427        9.8
2937                            0.434        9.8

[2938 rows x 20 columns]
```

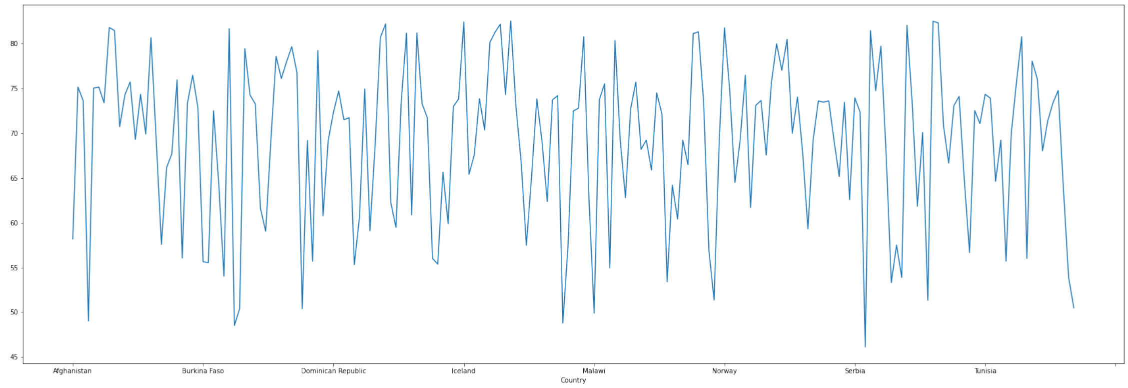
[24]:
```
data.columns
```

[24]:
```
Index(['Country', 'Year', 'Status', 'Life expectancy ', 'Adult Mortality',
       'infant deaths', 'Alcohol', 'percentage expenditure', 'Measles ',
       ' BMI ', 'under-five deaths ', 'Polio', 'Total expenditure',
       'Diphtheria ', ' HIV/AIDS', 'GDP', 'thinness_10-19_years',
       ' thinness 5-9 years', 'Income composition of resources', 'Schooling'],
      dtype='object')
```

[25]:
```
grouped = data.groupby('Country')
fig = plt.figure(figsize=(30,10))
ax = plt.axes()
grouped.mean()['Life expectancy '].plot(ax = ax)
```

```
/home/rom/Desktop/AIT/Programming/lib/python3.6/site-
packages/pandas/plotting/_matplotlib/core.py:1235: UserWarning: FixedFormatter
should only be used together with FixedLocator
  ax.set_xticklabels(xticklabels)
```

[25]:
```
<AxesSubplot:xlabel='Country'>
```

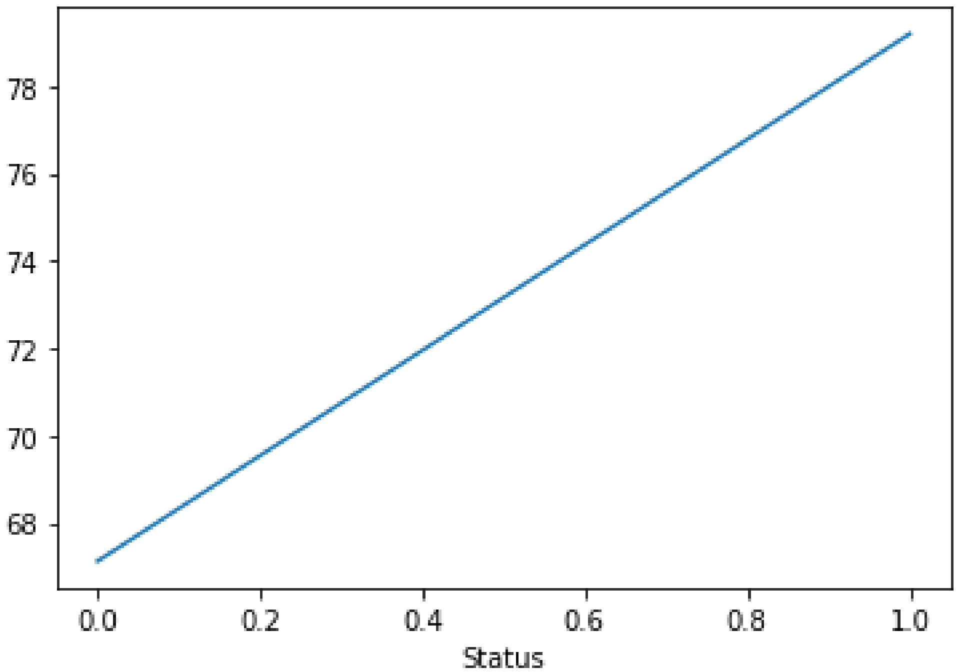


[26]:
```
#print(grouped.mean()['Life expectancy '])
series = grouped.mean()['Life expectancy ']
series.sort_values()
```

[26]:
```
Country
Sierra Leone                46.11250
Central African Republic    48.51250
Lesotho                     48.78125
Angola                      49.01875
Malawi                      49.89375
                              …
France                      82.21875
Switzerland                 82.33125
Iceland                     82.44375
Sweden                      82.51875
Japan                       82.53750
Name: Life expectancy , Length: 193, dtype: float64
```

[27]:
```
grouped = data.groupby("Status")
grouped.mean()['Life expectancy '].plot()
print(levels)
```

```
Index(['Developing', 'Developed'], dtype='object')
```



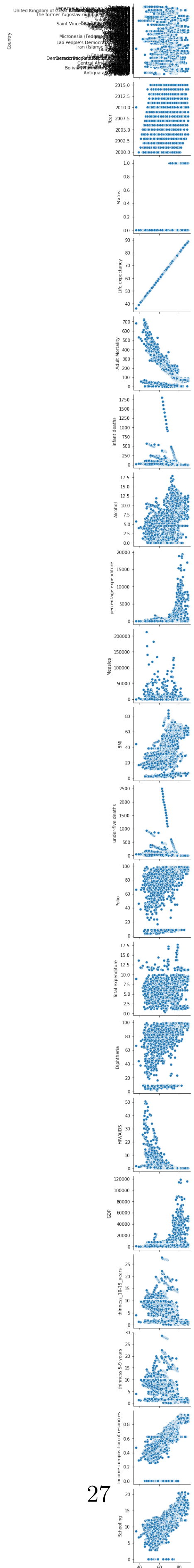
[28]:
```
from scipy import stats
data.groupby("Status").apply(lambda df: stats.ttest_ind(df['Life expectancy '],
 ↪df['Status']))
```

[28]:
```
Status
0    (367.7986278092848, 0.0)
1    (450.1250388656375, 0.0)
dtype: object
```

[29]:
```
col = list(data.columns)
```

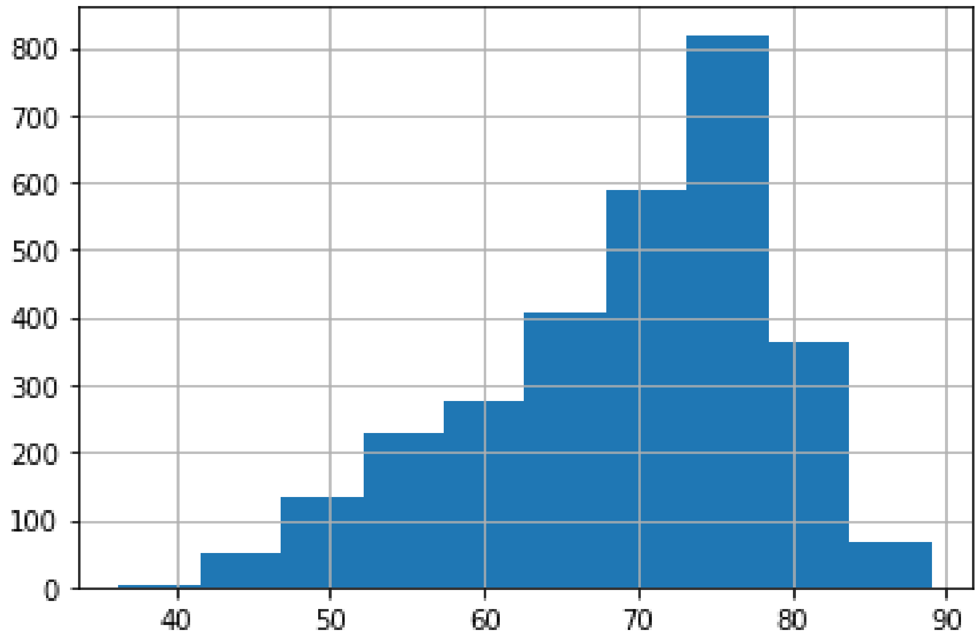
[30]:
```
import seaborn as sns

sns.pairplot(data,x_vars=["Life expectancy "],y_vars = col)
#adult mortality, schoolilng, income composition seem to be the most correlated
```

[30]:
```
<seaborn.axisgrid.PairGrid at 0x7f0f0a5cd7b8>
```


Country
Year
Status
Life expectancy
Adult Mortality
infant deaths
Alcohol
percentage expenditure
Measles
BMI
under-five deaths
Polio
Total expenditure
Diphtheria
HIV/AIDS
GDP
thinness 10-19 years
thinness 5-9 years
Income composition of resources
Schooling
Life expectancy

[31]:
```
data['Life expectancy '].hist()
#its a skewed graph so it's not normal
```

[31]: <AxesSubplot:>



[32]:
```
col
```

[32]:
```
['Country',
 'Year',
 'Status',
 'Life expectancy ',
 'Adult Mortality',
 'infant deaths',
 'Alcohol',
 'percentage expenditure',
 'Measles ',
 ' BMI ',
 'under-five deaths ',
 'Polio',
 'Total expenditure',
 'Diphtheria ',
 ' HIV/AIDS',
 'GDP',
```

```
 'thinness_10-19_years',
 ' thinness 5-9 years',
 'Income composition of resources',
 'Schooling']
```

[33]:
```
#From the question above, these three columns seem to have the best linear␣
↪correlation to life expectancy
X = data.loc[:,["Adult Mortality","Income composition of␣
↪resources","Schooling"]].values.astype(float)
X_all = data.drop(columns=['Country','Life expectancy ']).values.astype(float)
y = data['Life expectancy '].values.astype(float)

print(X.shape)
print(X_all.shape)

#Standardizing the data
mean = np.mean(X,axis=0)
std = np.std(X,axis=0)
X_norm = (X-mean)/std

mean_all = np.mean(X_all,axis=0)
std_all = np.std(X_all,axis=0)
X_all_norm = (X_all-mean_all)/std_all

#Shuffling the data
ix = np.arange(X.shape[0])
np.random.shuffle(ix)
m = X.shape[0]
percentage = 0.7
ix_train = ix[:int(m*percentage)]
ix_test = ix[int(m*percentage):]

X_norm_train = X_norm[ix_train]
X_all_norm_train = X_all_norm[ix_train]
X_norm_test = X_norm[ix_test]
X_all_norm_test = X_all_norm[ix_test]
y_train = y[ix_train]
y_test = y[ix_test]

print(X_norm_test.shape[0]/(X_norm_train.shape[0]+X_norm_test.shape[0]))

#Checking whether the shapes are in the correct form
assert X_norm_test.shape[0]/(X_norm_train.shape[0]+X_norm_test.shape[0]) < 0.31␣
↪and X_norm_test.shape[0]/(X_norm_train.shape[0]+X_norm_test.shape[0]) > 0.29
```

```
(2938, 3)
(2938, 18)
```

0.30020422055820284

[34]:
```
class Regression():
    def␣
↪__init__(self,X,y,grad_method,max_iter,alpha,tol,decay,decay_iter,decay_rate,stop_delay_cou
↪
        self.X = X
        self.y = y
        self.grad_method = grad_method
        self.max_iter = max_iter
        self.alpha = alpha
        self.tol = tol
        self.decay = decay
        self.decay_iter = decay_iter
        self.decay_rate = decay_rate
        self.stop_delay_counter = stop_delay_counter
        self.verbose = verbose
        self.lam = lam
        self.poly = poly
        self.poly_deg = poly_deg

    #1. hypothesis function
    def h(self, theta):
        hypothesis = self.X@theta
        return hypothesis

#2. cost function
    def cost(self, theta):
        J = 1/(2*X.shape[0])*(self.h(X,theta)-y).T@(self.h(X,theta)-y)
        return J

    def cost_reg(self, X, y, theta,lamb):
        J = 1/(2*X.shape[0])*(self.h(X,theta)-y).T@(self.h(X,theta)-y) +␣
↪lamb*np.sum(theta@theta)
        return J

#3. gradient function
    def gradient_reg(self, X, y, theta, average = False):
        dJ = X.T@(self.h(X,theta)-y)/(X.shape[0]) + theta*lamb
        return dJ

    def gradient(self, X, y, theta, average = False):
        dJ = X.T@(self.h(X,theta)-y)/(X.shape[0])
        return dJ

    def mini_batch():
        cost = []
```

```
        theta = initial_theta
        iteration = 0
        cost.append(self.cost(X,y,theta,average))
        ix = np.arange(y.size)
        percentage = 0.1
        ix_range = int(percentage*ix.size)
        for n in range(max_iteration):
            np.random.shuffle(ix)
            X_batch = X[:ix_range]
            y_batch = X[:ix_range]
            gradient = self.gradient(X_batch,y_batch,theta,average)
            theta = theta - alpha*gradient
            cost.append(self.cost(X_batch,y_batch,theta,average))
            iteration += 1
        cost = np.array(cost)
        return theta,cost,iteration

    def stochastic():
        cost = []
        theta = initial_theta
        iteration = 0
        cost.append(self.cost(X,y,theta,average))
        for n in range(max_iteration):
            for i in X.shape[0]:
                if(self.grad_method == 'ridge'):
                    gradient = self.gradient_reg(X[i],y,theta,average)
                else:
                    gradient = self.gradient(X[i],y,theta,average)
                theta = theta - alpha*gradient
                cost.append(self.cost(X[i],y,theta,average))
                iteration += 1
        cost = np.array(cost)
        return theta,cost,iteration

    def batch():
        cost = []
        theta = initial_theta
        iteration = 0
        cost.append(self.cost(X,y,theta,average))
        for n in range(max_iteration):
            gradient = self.gradient(X,y,theta,average)
            theta = theta - alpha*gradient
            cost.append(self.cost(X,y,theta,average))
            iteration += 1
        cost = np.array(cost)
        return theta,cost,iteration
```

```python
    def polynomial_features():
        pass

    def decay_learning_rate():
        pass

    def ridge():
        pass

    def fit():
        pass

    def predict():
        pass

    def score():
        pass

    def mse():
        pass

#would've been better if the class wasn't fixed to the form Regression(X, y,
 ↪grad_method, max_iter, alpha, tol, decay, decay_iter, decay_rate,
 ↪stop_delay_counter, verbose, lam, poly, poly_deg)
#because i have written a class but not in this way
```

2). **Regression** - Prepare your X and y into Numpy array (you have to map from Pandas to numpy). For X, prepare two versions of them. For first X_selected, you have to choose the most 3 important features from above, and for second X_all, simply use all features (you may want to omit Country since they are categorical). Set y to life expectancy. (1 or 0pt)

- Perform standardization using Numpy way (NOT sklearn way). (1 or 0pt)
- Perform train-test split by using Numpy way (NOT sklearn way). Use test size of 0.3. (1 or 0pt)
- Perform assertion whether your splitting is correct accordingly (1 or 0pt)
- Write a class Regression(X, y, grad_method, max_iter, alpha, tol, decay, decay_iter, decay_rate, stop_delay_counter, verbose, lam, poly, poly_deg) that can perform the followings:
  - Mini-batch, Stochastic, and Batch Gradient Descent (each 2pts)
  - Polynomial of degree k (2 or 0pt)
  - Decay learning rate (1 or 0pt)
    * Decay learning rate is a learning rate that becomes smaller after certian iteration. For example, after 5 iterations, the learning rate will reduce to 95% of the current learning rate.
    * To implement it, simply multiply current learning rate with some constant decay_rate. For now, set it to 0.9

  - Regularization with ridge (2 or 0pt)
  - Must have at least four methods for fit() (i.e., for finding weights) predict() (i.e., for predicting X_test data), score() (i.e., for returning r^2 score), and mse() (return mse) (each 1pt)
  - Accepts X, y, grad_method (default set to "batch"), alpha (learning rate), max_iter, tol, decay (whether to use decay learning rate; default set to False), decay_iter (after how many iterations will the decay apply), stop_delay_counter (this is the maximum number of times that decay the learning rate), verbose (default is set to False, whether model will display the Cost for each iteration), lam (this is the ridge regularization parameter), poly (default is set to False), and poly_deg (default is set to 2) (each 1/13pt)

- Create the following 3 models **from your class** (For any unspecified parameters, feel free to use any :D)

  1. For the first model, transform your feature using polynomial degree 3, then perform linear regression with batch gradient descent with early stopping of tol 1e-3 (1 or 0pt)
  2. For the second model, perform linear regression with mini-batch gradient desent with early stopping of tol 1e-3 (1 or 0pt)
  3. For the third model, perform ridge regression with stochastic gradient desent with early stopping of tol 1e-3 and decay set to True and lam to 1e-4 (1 or 0pt)

- Create Lasso model from Sklearn with default parameters (1 or 0pt)

- For these four models, using two different versions of X, perform a cross validation of 10 folds, comparing the four models * two versions of X. Here you should implement cross validation. Report which one is the best candidate model (3pts for implement from scratch or 1pt for using sklearn)

  - Recall that in a 10 folds cross validation, you split your data into 10 even pieces. Then you run 10 iterations, where in each iteration, you pick 1 of this piece as the validation set, and the rest as training set. Once you reach the 10th iteration, you would have already exhaust all the 10 pieces as validation set.

- Using the best model, fit again with the training data. Plot the weights using bar charts along the feature names. Before you actually plot the weights, we need to multiply these weights by their feature standard deviation, so to reduce these weights to same unit of measure. Interpret these weights and what they imply. (For those who are curious why we need to multiply with std, you may read this > https://scikit-learn.org/stable/auto_examples/inspection/plot_linear_model_coefficient_interpretation.html#interpreti coefficients-scale-matters (2 or 0pt)

- Perform predictions on testing data. Print adjusted r^2 and mse. (1 or 0pt)

- Plot the predicted values against actual values (1 or 0pt)

3). **Classification**

- Change your y to discrete value. Here split y into three class, {0, 1, 2}, where 0 belongs to low life expectancy group, and 2 for the high life expectancy group. (1 or 0pt)

- Write a class for multinomial logistic regression with stochastic gradient descent. Must have at least six methods for fit() (i.e., for finding weights) predict() (i.e., for predicting X_test data), accuracy() (i.e., for returning accuracy score), recall(), precision(), and f1() (each 1pt)

- Using the best X_train of the two suggested by the cross validation step, fit the data with your class. (1 or 0pt)
- Perform predictions on testing data. Print accuracy, recall, precision, and f1_score from your class. (1 or 0pt)
- Plot the decision boundary with the X_test data. To plot this, you may want to choose only 2 features. (1 or 0pt)

4). **Final verdict**

- Attempt to do whatever ways - including sklearn or scratch - or change your features, or do feature enginnering such that your mse is lowest possible. (0 to 5pts - following class normal distributions)