VIETNAM NATIONAL UNIVERSITY HO CHI MINH CITY
UNIVERSITY OF SCIENCE
FACULTY OF INFORMATION TECHNOLOGY

APPLIED MATHEMATICS AND STATISTIC

PROJECT 3
LINEAR REGRESSION

Student's name: Nguyễn Đinh Quang Khánh
Student ID: 20127530
Class: 20CLC11
Instructor: Phan Thị Phương Uyên
Nguyễn Văn Quang Huy

Ho Chi Minh City, July/2022

Contents

1 Project ideas and algorithms

1.1 Project ideas

- In statistics, **linear regression** is a linear approach for modelling the relationship between a scalar response and one or more explanatory variables (also known as dependent and independent variables). [1]
- **Linear regression** attempts to model the relationship between two variables by fitting a linear equation to observed data. One variable is considered to be an explanatory variable, and the other is considered to be a dependent variable. [2]
- In this project, I will use ordinary least squares (OLS) method for estimating the unknown parameters in a linear regression model which is a type of linear least squares. [3]
- Because there are many requirements with different ways to process so I will go deeply for the idea in each sections.

1.2 Algorithm step by step introduction

- With OLS method we need to find a root of this equation: $Ax \approx b$
- Consider a matrix A has size $m \times n$ $(m > n)$ and column vector b has size m. We have the Euclidean norm of the remainder square of $Ax \approx b$:

$$r = ||Ax - b||^2 \quad (1)$$

- And to solve the best root of this equation, we will minimize the formula (1) above then we will get the best root of the equation as below:

$$x = (A^T A)^{-1} A^T b$$

Note: $(A^T A)^{-1} A^T$ or $A^\dagger$ is Moore–Penrose inverse of A. [4]

2 Modules

- During this project, I will use 3 moduless:
  - **numpy** module using for calculation of matrices and vectors.
  - **pandas** module using for reading file csv and convert the dataset into dataframe or series.
  - **matplotlib** module using for visualize the relationship between each features and Life Expectancy.

```
# Đọc dữ liệu bằng pandas
train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')

# Lấy các đặc trưng X và giá trị mục tiêu y cho các tập huấn luyện (train) và kiểm tra (test)
X_train = train.iloc[:, :-1]     # Dataframe (chứa 10 đặc trưng huấn luyện)
y_train = train.iloc[:, -1]      # Series    (chứa 1 giá trị mục tiêu kiểm tra)

X_test = test.iloc[:, :-1]       # Dataframe (chứa 10 đặc trưng kiểm tra)
y_test = test.iloc[:, -1]        # Series    (chứa 1 giá trị mục tiêu kiểm tra)

# Sinh viên có thể sử dụng các khác nếu cần

# Convert dataframe or series to numpy array
X_train_np = X_train.to_numpy()
y_train_np = y_train.to_numpy()

```

```
# Convert dataframe or series to numpy array
X_test_np = X_test.to_numpy()
y_test_np = y_test.to_numpy()

# Get the header names of 11 attributes
headerName = list(train.columns)
```

- In shell *data read*, I will read the *training set* and *test set* by using method *read_csv()* with the file name parameter passed in provided in *pandas module*. Then I will use *iloc* to select 10 *features* for training in *train dataframe* by select all rows and pick the first 10 *features* which is $iloc[:, :-1]$.
- Then, we will get the target columns from the last columns of *train dataframe* which is $iloc[:, -1]$. So, we have already separated the *training set* into 2 parts X_train and y_train.
- Similarly, We will apply the same ways above to seperate the *test set* into 2 parts X_test and y_test.
- For easier in the way we calculate with vectors and matrices, I will convert them to *numpy array* by using *to_numpy()* provided in *pandas module*.
- In line 23, I will get the header names of all 11 attributes which we might need later.

3 Functions

- In this project, I will create below functions to help me fit the data by using least squares technique. These functions I references from [5].

```
class OLSLinearRegression:
    def fit(self, X, y):
        X_pinv = np.linalg.inv(X.T @ X) @ X.T    # np.linalg.pinv(X)
        self.w = X_pinv @ y
        return self

    def get_params(self):
        return self.w

    def predict(self, X):
        return np.sum(self.w.ravel() * X, axis=1)

def rmse(y, y_hat):
    return np.sqrt(np.mean((y.ravel() - y_hat.ravel())**2))

def plot_data(x, y):
    plt.scatter(x, y, color='b', marker='o')

    plt.xlabel('x')
    plt.ylabel('y')

    plt.grid()
    plt.show()
```

- In the code above, I will create class *OLSLinearRegression* with 3 methods.
- In fit method, I will apply the root of the equation $Ax \approx b$ that is $x = (A^T A)^{-1} A^T b$.
  - In here, I will use different alias for input, output and parameter in linear regression. [5]
    - $A \rightarrow X$
    - $b \rightarrow y$
    - $x \rightarrow w$ (w: weight)
  - $Ax \approx b \rightarrow Xw \approx y$ or $Xw = y$ (y also called regression line)
  - In this method, I will use method of *numpy* module that is *np.linalg.inv()* to get the *inverse* form of $(X^T X)$ which can be written as $(X^T X)^{-1}$

  - Then we will multiply with X^T to get $(X^TX)^{-1}X^T$.
  - Finally, we will multiply with y and return $(X^TX)^{-1}X^Ty$ to attribute w of this class's object.

- The method $get\_params(self)$ is to return the attribute w of the $OLSLinearRegression$ object after we use method $fit()$ to fitting the data.

- The method $predict(self, X)$ is to predict the test result with the $weight$ we get after fitting data. We just simply flatten the vector w of this class by using method $ravel()$ of $numpy\ module$. Then we will get the $prediction$ set.

- The function $rmse(y,\ y\_hat)$ is used to estimate the mean of average error, and can be calculated with the below formula:

$$MSE = \frac{1}{n}\sum_{n}^{i=1}(y_i - \hat{y}_i)^2 \quad [6]$$

  With:

  - n: Number of observed samples
  - y_i: Target values of the i_th sample
  - $\hat{y}_i$: Target values of the i_th sample predicted from linear regression model.

  In this function:

  - First, I will apply the $minus\ operator$ between 2 $numpy\ arrays$.
  - Secondly, I will use $square()$ method of $numpy\ module$ to find the $square$ of every element in a given array.
  - Finally, I will use $mean()$ method of $numpy\ module$ to get the mean value of all elements in a given array.

- The function $plot\_data(x,\ y)$ is used to visualize the relationship between coordinate x and y pass in by drawing each point in Cartesian coordinate system.

  - In this function, I will use method $scatter()$ from module $matplotlib.pyplot.plot$ and pass in the x and y coordinates, set $color =$ 'b' and $marker =$ 'o' for a clearer plot.
  - To emphasize the differences between x and y, I will add grid to the plot.
  - Finally is show the plot to console by method $show()$.

4 Requirements: Construct models to predict the average life using linear regression

4.1 First requirement: Using all 10 features provided

4.1.1 Descriptions

In this requirement, we have to train the model one time only with 10 features on the whole train dataset in $train.csv$. The train dataset we have already read from the step $data\ read$ already mentioned above.

After training the model we have to fulfill the formula of linear regression (The formula to calculate y from all 10 features in x).

Finally, we have to test the result one time only on the test dataset in $test.csv$.

4.1.2 Explains

```
# Phần code cho yêu cầu 1a
# Train model with all features
lr = OLSLinearRegression().fit(X_train_np, y_train_np)
w = lr.get_params().ravel()
y_test_pred = lr.predict(X_test_np)
print(w)
```

- In the above code, firstly, I will create an object of class $OLSLinearRegression$ and use it to trigger the method $fit()$ to fitting the data and return the attribute w to this object.
- I will pass in the X_train_np and y_train_np to the method $fit()$. X_train_np is the $numpy\ array$ with size $(1085,\ 10)$ with 10 features as 10 coulmns and each columns has 1085 different instances.
- After trigger the method $fit()$, this lr object will have an attribute w inside. So we will get it by using method $get\_params()$ implemented in this class.
- Then, we will use the method $predict()$ to predict the target values with the $weight$ we have fitted before. To trigger this method we will pass in the 10 features of test set in to predict the expected outcomes.
- Finally, we will print the result w to the console.
- In the next step, we have to compare our predicted outcomes with the true target values in the test set to know how good this model is.

```
# Gọi hàm RMSE (tự cài đặt hoặc từ thư viện) trên tập kiểm tra
print(rmse(y_test_np, y_test_pred))
```

- To estimate how good our model is, we will use $RMSE$ technique. The $RMSE$ function estimates the deviation of the actual $y - values$ from the regression line. Another way to say this is that it estimates the standard deviation of the $y - values$ in a thin vertical rectangle. [6]
- In summary, The lower the $RMSE\ value$ is the better of the model be.

4.1.3 Results and comments

After execute the code above, we will get the result w:

```
 [ 1.51013627e-02  9.02199807e-02  4.29218175e-02  1.39289117e-01
-5.67332827e-01 -1.00765115e-04  7.40713438e-01  1.90935798e-01
 2.45059736e+01  2.39351661e+00]
```

Therefore, our regression formula is:

$$\begin{aligned}\text{Life expectancy} = \ & 0.0151 \times \text{Adult Mortality} \ + \\ & 0.0902 \times \text{BMI} \ + \\ & 0.0429 \times \text{Polio} \ + \\ & 0.1393 \times \text{Diphtheria} \ + \\ & (\text{-}0.5673) \times \text{HIV/AIDS} \ + \\ & (\text{-}0.0001) \times \text{GDP} \ + \\ & 0.7407 \times \text{Thinness age 10-19} \ + \\ & 0.1909 \times \text{Thinness age 5-9} \ + \\ & 24.506 \times \text{Income composition of resources} \ + \\ & 2.3935 \times \text{Schooling}\end{aligned}$$

Then we will print out the *RMSE value*:

```
7.064046430584337
```

- From the *weight* above, we can construct our regression formula to calculate the *life expectancy* from 10 features with the relative *weight* for each features.
- We can see that the *RMSE value* ≈ 7.06. This value means how accuracy of the model to predict the data. And because value 7 is quite big so the accuracy of the predicted data is not so good. [7]

4.2 Second requirement: Using one by one feature provided

4.2.1 Descriptions

In this requirement, firstly, we have to construct models for only one feature and do the experiments on all 10 features.

Secondly, we have to find which model is the best by using 5-fold cross validation technique.

Thirdly, we have to report and print out 10 results for 10 relative models from 5-fold cross validation (By taking average of 5 validation times).

After that, we have to train the best feature model again on the whole train data.

Then, we have to fulfill the formula to calculate y from the best feature model.

Finally, we have to test the result one time only on the test dataset in *test.csv*.

4.2.2 Explains

```
# Get copies of X_train and y_train
X_train_clone = X_train_np.copy()
y_train_clone = y_train_np.copy()

# Shuffle data for 1 time only before apply K-fold cross validation
# Generate a random order list from 0->1084 and assign X_train_clone and y_train_clone
# to the relative position of this random list
randomize = np.arange(len(X_train_clone))
np.random.shuffle(randomize)
X_train_clone = X_train_clone[randomize]
y_train_clone = y_train_clone[randomize]

# Create RMSE list for 10 features
RMSE = np.zeros(10)
for i in range(10):
    # Get X_train_feature from X_train_clone set of feature i_th
    X_train_feature = X_train_clone[:, i]
    # Get y_train_feature from y_train_clone
    y_train_feature = y_train_clone

    # Get number of rows in the dataset
    numOfRows = X_train_feature.shape[0]
    # Length of validation set when using KFold cross validation technique
    length_per_set = int(numOfRows/5)

    # Apply 5-fold cross validation by iterate 5 times and split train set and validation
     set
    for j in range(5):
        # Get the i_th set after split the X_train_feature into 5 equal parts
        X_val = X_train_feature[(j*length_per_set):((j+1)*length_per_set)].reshape(-1,1)
        # Get all values before X_val set after split the X_train_feature into 5 equal
    parts
        X_first_part_train_kfold = X_train_feature[:(j*length_per_set)].reshape(-1,1)
        # Get all values after X_val set after split the X_train_feature into 5 equal
    parts
        X_second_part_train_kfold = X_train_feature[((j+1)*length_per_set):].reshape
    (-1,1)
```

```
34          # Concatenate 2 train set into 1 train set to train the model and test on
    validation set
35          X_train_kfold = np.concatenate((X_first_part_train_kfold,
    X_second_part_train_kfold), axis=0).reshape(-1,1)
36
37          # Get the i_th set after split the y_train_feature into 5 equal parts
38          y_val = y_train_feature[(j*length_per_set):((j+1)*length_per_set)].reshape(-1,1)
39          # Get all values before y_val set after split the y_train_feature into 5 equal
    parts
40          y_first_part_train_kfold = y_train_feature[:(j*length_per_set)].reshape(-1,1)
41          # Get all values after y_val set after split the y_train_feature into 5 equal
    parts
42          y_second_part_train_kfold = y_train_feature[((j+1)*length_per_set):].reshape
    (-1,1)
43          # Concatenate 2 train set into 1 train set to train the model and test on
    validation set
44          y_train_kfold = np.concatenate((y_first_part_train_kfold,
    y_second_part_train_kfold), axis=0).reshape(-1,1)
45
46          # Train model with the i_th feature
47          lr = OLSLinearRegression().fit(X_train_kfold, y_train_kfold)
48          y_val_pred = lr.predict(X_val)
49          RMSE[i] += rmse(y_val, y_val_pred)
50      # Get average of RMSE for 5 validation times of i_th feature
51      RMSE[i] /= 5
52
53  # Get index of feature with minimum RMSE
54  best_feature_index = np.argmin(RMSE, axis=0)
55
56  # Create dictionay from the table of the requirement
57  dict_1b = {'Mô hình với 1 đặc trưng': headerName[:-1], 'RMSE':RMSE}
58  # creating a dataframe from dictionary
59  df_1b = pd.DataFrame(dict_1b)
60  # Align center for all columns of dataframe
61  center_aligned_df = df_1b.style.set_properties(**{'text-align': 'center'})
62  # Display dataframe
63  display(center_aligned_df)
```

- In line $2 \rightarrow 3$, I will create clones of X_train_np and y_train_np because we will shuffle one time before apply the 5-fold cross validation technique by using method $copy()$ from $numpy\ module$.
- In line $8 \rightarrow 11$, I will shuffle all elements in the X_train_clone and y_train_clone. The idea to shuffle the data I references from this link [8].
  - Firstly, I will create a $numpy\ array$ with evenly spaced values is 1 and the values are from $0 \rightarrow (len(X\_train\_clone) - 1)$. These values of this $numpy\ array$ stand for the indices of $rows$ in our dataset which size is 1085 so the values are from $0 \rightarrow 1084$.
  - Secondly, I will use method $np.random.shuffle()$ with an array passed in to shuffle all the contents of an array. Therefore, we will get a $numpy\ array$ which has mixed order of values in range $[0;\ 1084]$.
  - Finally, we can get our shuffle array by get all the rows which order match with the order of the values of the $randomize\ array$. Therefore, we can get the shuffle of X_train_clone and the relative y_train_clone.
- In line 14, I will create a zeros $numpy\ array$ with size is 10 which will include the average of $RMSE$ of each features at the end of 5-fold cross validation technique.
- In line $15 \rightarrow 51$, I will apply 5-fold cross validation technique to find the best feature model. The idea of 5-fold cross validation algorithm I reference from this site [9].

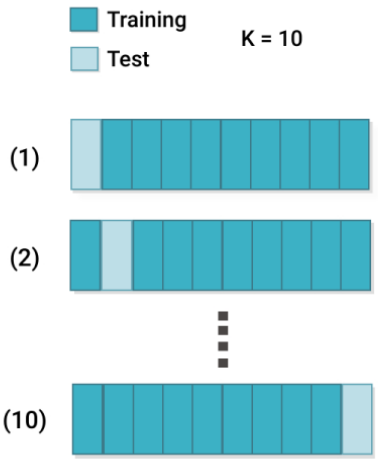


Figure 1: *Cross validation k-fold algorithm*

- The algorithm of K-fold cross validation technique:
  - Step 1: We have to select value for k. Usually, k is 5 or 10. In this project, due to the length of dataset is 1085 which is divisible by 5, I will choose value $k = 5$.
  - Step 2: We split the dataset into k parts (which are called folds).
  - Step 3: Pick one fold for the validation set, and the remaining $k - 1$ folds for the training set.
  - Step 4: Train the model on the training set. On each iteration of cross-validation, we must train a new model independently of the model trained on the previous iteration.
  - Step 5: Validate the result on the validation set.
  - Step 6: Calculate the $RMSE$ value of each iteration of cross-validation.
  - Step 7: We will iterate the steps from $3 \rightarrow 6$. At each iteration, we will add the $RMSE$ value up. Finally, we will get the mean of $RMSE$ values for 5 iteration.
- In line 15, I will iterate 10 times for 10 individual features as models.
- In line 17 and 19, I will get the X and y train set of the i_th feature.
- Then I will get the length of the data-set by using $X\_train\_feature.shape[0]$. And we can get the length for each folds by divide the length of data-set to 5.
- After that, I will implement the k-fold cross validation technique for the i_th feature without using support from the library due to the steps listed above.
  - In line 27, I will iterate 5 times because $k = 5$.
  - For each iteration, firstly, I will get the index j_th fold first and assign it to X_val. Because length of each fold is $length\_per\_set$ so we will multiply the length with the index j_th.
  - Then our mission is to get 4 remaining folds. To do this, I will get all folds before index j_th which indices from $[0;\ j\_th - 1]$. Similarly, I will get all folds with indices is after j_th which from $[j\_th + 1; 4]$. Then I will use $concatenate()$ of $numpy\ module$ to concatenate 2 parts into one and assign to X_train_kfold.
  - Similarly to X part, I will apply the same implementation to get the relative y set.
  - In summary, from line $28 \rightarrow 44$, I have already realistic the idea of the k-fold cross validation technique.
  - Then, our mission now is training model with the train set we just got. Similar to requirement 1, I will create lr object of class $OLSLinearRegression$ and trigger method $fit()$ with X_train_kfold and y_train_kfold passed in.
  - Then I use method $predict()$ to get the predicted result set. Finally, I calculate and save the $RMSE$ of this iteration.

  - I will iterate 5 times and add the $RMSE$ up. And we get the average for 5 validation times of i_th feature. And I have to apply this algorithm to all 10 individual features.
  - In the end, I will have the list $RMSE$ which conclude the average of $RMSE$ of each feature.

- Then we have to get the feature which $RMSE$ value is the lowest. In this data-set, the feature *Schooling* has the lowest $RMSE$ which is approximately 11.784.
  - I will get the minimum $RMSE$ of the list by using method $argmin()$ of *numpy module*. This function will return the index of the minimum element in the *numpy array*.
  - This function I reference from this site. [10]

- For clearer, I will visualize the $RMSE$ of each models to the table by using *pandas* module. The result of this will be listed later.

- So we have finished the implementation to find the best models among 10 features.

Now our task is to train the best feature model on the whole training set in file *train.csv*. The implementation to train model is quite similar to requirement 1 and listed below.

```
# Get Schooling training set and testing set
X_Schooling_train, y_Schooling_train = X_train_np[:, best_feature_index].reshape(-1,1),
    y_train_np.reshape(-1,1)
X_Schooling_test, y_Schooling_test = X_test_np[:, best_feature_index].reshape(-1,1),
    y_test_np.reshape(-1,1)
# Train feature Schooling in all training dataset
lr = OLSLinearRegression().fit(X_Schooling_train, y_Schooling_train)
w = lr.get_params().ravel()
y_Schooling_test_pred = lr.predict(X_Schooling_test)
print(w)
```

- In the code above, firstly, I will get X and y of feature *Schooling* which is the best feature model in training set.
- Secondly, I create object lr of class $OLSLinearRegression$ and trigger the method $fit()$ of this class to train the model.
- Then we can get the result w by using method $get\_params()$ and print the result to the console.
- Then we will predict the model with the above result.

```
print(rmse(y_Schooling_test, y_Schooling_test_pred))
```

- Finally I calculate the $RMSE$ of the best feature model and print it to the console.

4.2.3 Results and comments

This is the table of 10 $RMSE$ results of 10 relative models from 5-fold cross validation.

STT	Mô hình với 1 đặc trưng	RMSE
0	Adult Mortality	46.228491
1	BMI	27.924596
2	Polio	18.009645
3	Diphtheria	15.997620
4	HIV/AIDS	67.070168
5	GDP	60.251200
6	Thinness age 10-19	51.762326
7	Thinness age 5-9	51.674216
8	Income composition of resources	13.301988
9	Schooling	11.784275

- As we can see the results in the table, the lowest feature model $RMSE$ is a *Schooling* feature model. Or we can say that the feature *Schooling* is the best feature model to predict the $life\ expectancy$.

- According to this site [11]. Adults with higher educational attainment have better health and lifespans compared to their less-educated peers. Therefore, it is not a surprise that *Schooling* has a big impact on the *life expectancy*.

Then we will print out the *RMSE* of the best feature model:

```
[5.5573994]
```

Finally, we will calculate and print out the *RMSE* value of the best feature model test on the whole test set in file *test.csv*.

```
10.260950391655376
```

Therefore, our regression formula is:

$$\text{Life expectancy} = 5.5573994 \times \text{Schooling}$$

- As we can see the differences in *RMSE* between the first compared with this requirement, with the first requirement, the *RMSE* is approximately to 7.06 while the *RMSE* of this requirement is approximately to 10.26.
- To know why is this, I have referenced this site [12]. According to this site, the number of features we add in it is depends on what we want our model to archive. If the feature is not relevant to the target value and we add it to our model so it will allow models to mistake noise for signal.

4.3 Third requirement: Using my own models

4.3.1 Descriptions

In this requirement, firstly, we have to construct our own *m* models. And I constructed my own 3 models.

Secondly, I will apply the same technique in the second requirement that is 5-fold cross validation to find the best models among 3 models I have created.

Finally, I have to train the model with the best models above on the whole train set in file *train.csv*.

4.3.2 Explains

4.3.2.a My first model

My first model:

$$\text{Life expectancy} = w_1 \times \text{Polio} + w_2 \times \text{Diphtheria} + w_3 \times \text{Income composition of resources} + W_4 \times \text{Schooling}$$

- To understand why I created this model. First, let's see the correlation of the first 10 features to our target *life expectancy*.

```
for i in range(10):
    X, y = X_train_np[:, i].reshape(-1,1), y_train_np.reshape(-1,1)
    print(f'\n{headerName[i]}:\n')
    plot_data(X, y)
```

- In the implementation above, I will plot the data which x-axis is the *i_th* feature and y-axis is our target value *life expectancy*.
- The result plots will be listed down here.

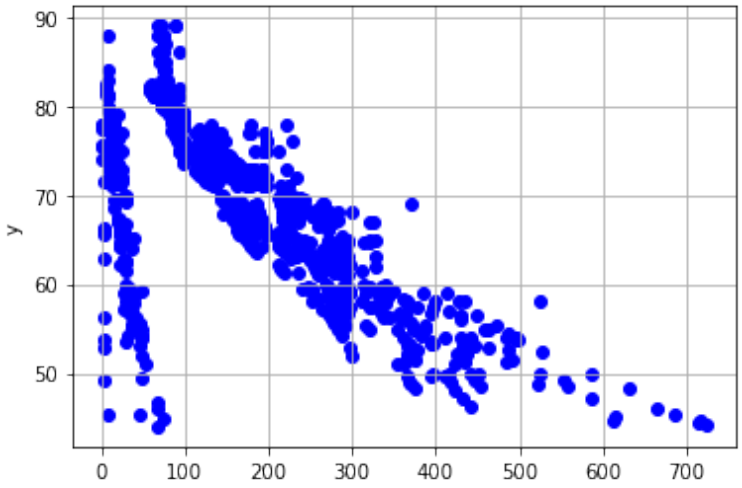


Figure 2: *Adult Mortality*

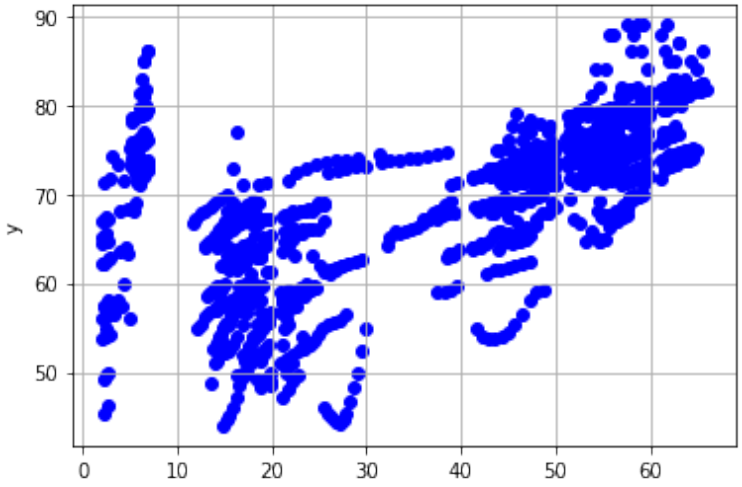


Figure 3: *BMI*

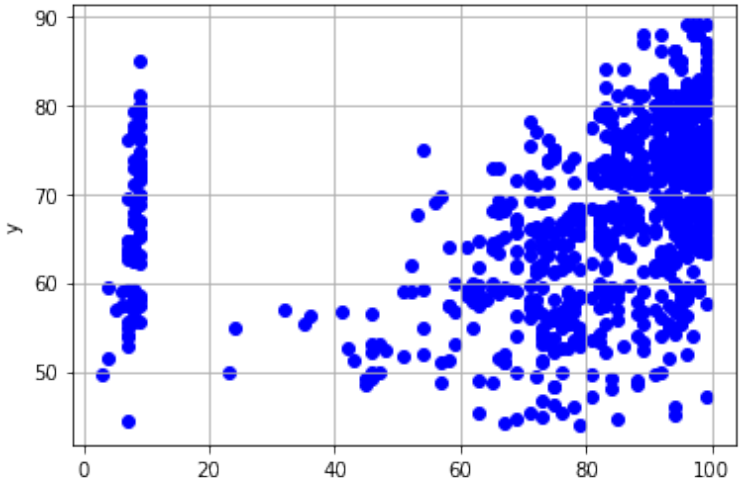


Figure 4: *Polio*

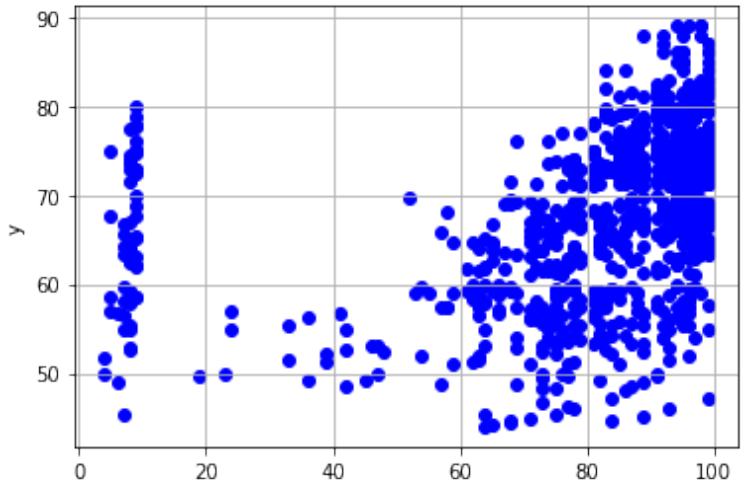


Figure 5: *Diphtheria*

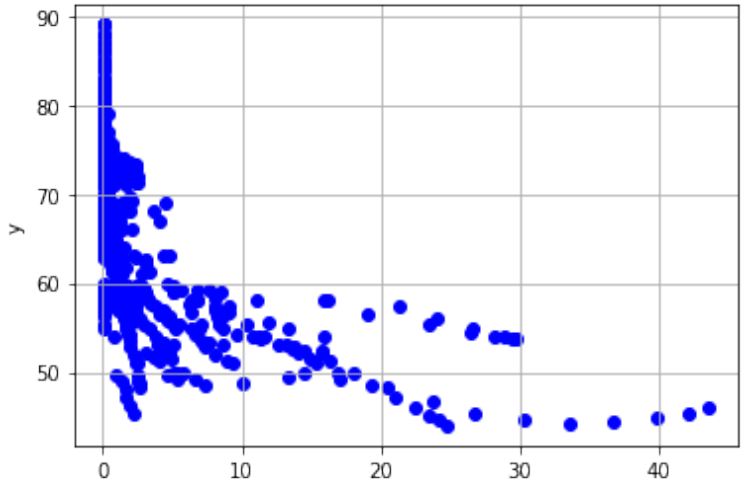


Figure 6: *HIV/AIDS*

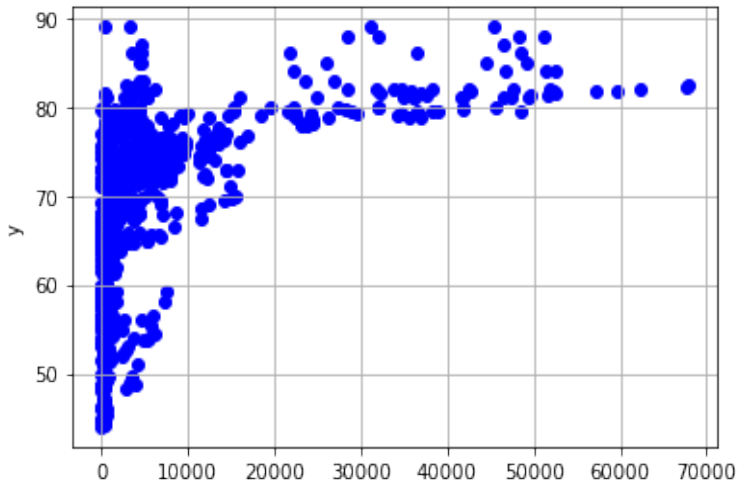


Figure 7: *GDP*

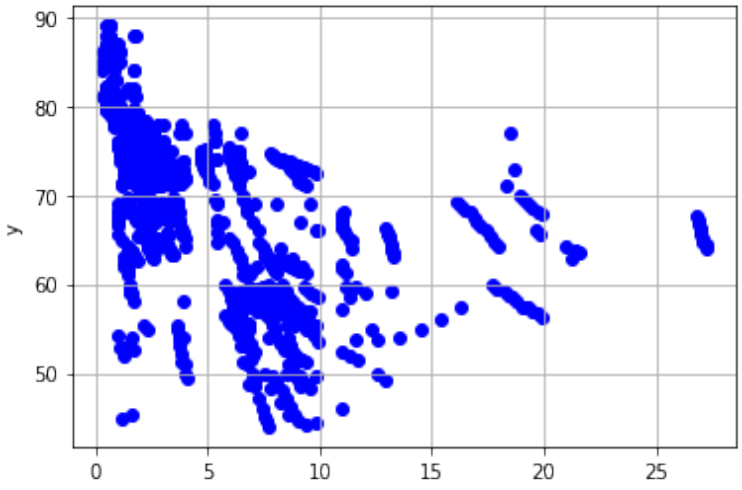


Figure 8: *Thinness age 10-19*

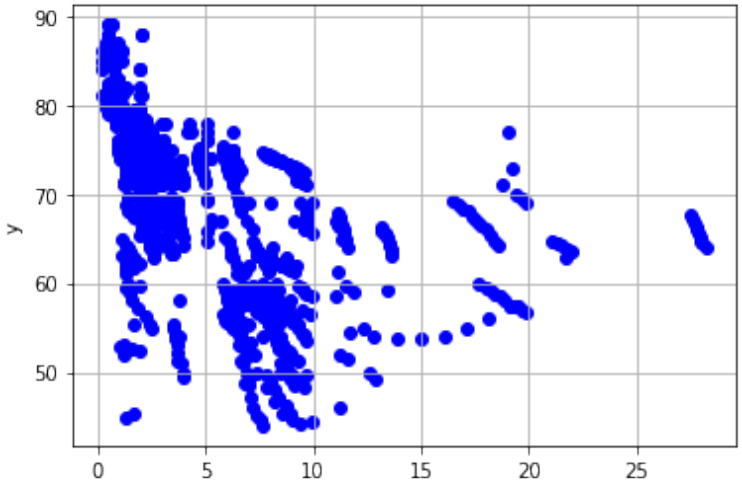


Figure 9: *Thinness age 5-9*

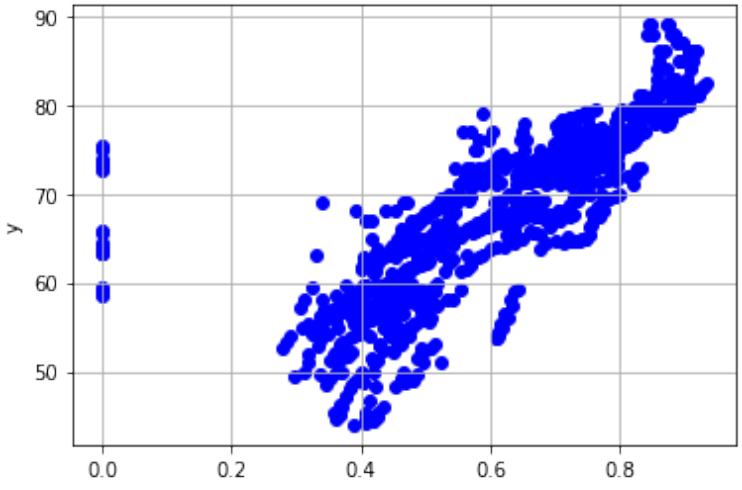


Figure 10: *Income composition of resources*

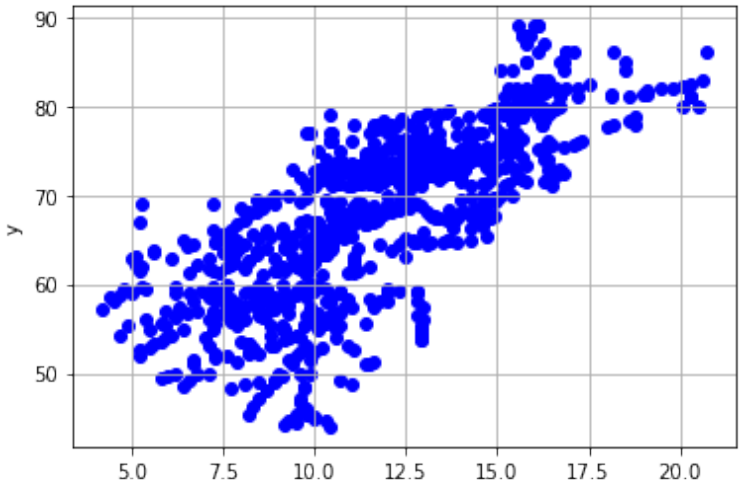


Figure 11: *Schooling*

- As we can see multiple plots above, let's concentrate on *Schooling*, *Income composition of resources*, *Diphtheria*, *Polio* figures. We can easily recognized that the shape of the graph is a linear line with positive coefficient. This mean the higher the x is the higher the y will be. And this is the reason that I choose the composite models conclude these 4 features together.
- Now I will explain deeply on the idea of my implementation listed below.

```
# Get copies of X_train and y_train
X_train_clone = train.iloc[:, :-1].copy() # Dataframe
y_train_clone = train.iloc[:, -1].copy() # Dataframe

# Shuffle data for 1 time only before apply K-fold cross validation
# Generate a random order list from 0->1084 and assign X_train_clone and y_train_clone
# to the relative position of this random list
randomize = np.arange(len(X_train_clone))
np.random.shuffle(randomize)
X_train_clone = X_train_clone.iloc[randomize]
y_train_clone = y_train_clone.iloc[randomize]

# Create RMSE list for 3 models
RMSE = np.zeros(3)
```

- Before I apply 5-fold cross validation for 3 models to find the best models among 3, I will shuffle the data 1 time only.
- The algorithm for shuffle data is the same in the second requirement. The idea of this algorithm I references from this site. [8]
- And because I created 3 models in total so I create a zero *numpy array* with the size is 3 to record all the *RMSE* when applying 5-fold cross validation technique.

```
# My first model:
# LE = w1*Polio + w2*Diphtheria + w3*Income composition of resources + w4*Schooling
X_train_Polio_Diph_Income_Schooling = X_train_clone[['Polio','Diphtheria','Income
    composition of resources','Schooling']].to_numpy()
y_train_Polio_Diph_Income_Schooling = y_train_clone.to_numpy()

# Apply 5-fold cross validation by iterate 5 times and split train set and validation set
for j in range(5):
    # Get the i_th set after split the X_train_feature into 5 equal parts
    X_val = X_train_Polio_Diph_Income_Schooling[(j*length_per_set):((j+1)*length_per_set)
    ]
    # Get all values before X_val set after split the X_train_feature into 5 equal parts
    X_first_part_train_kfold = X_train_Polio_Diph_Income_Schooling[:(j*length_per_set)]
```

```
    # Get all values after X_val set after split the X_train_feature into 5 equal parts
    X_second_part_train_kfold = X_train_Polio_Diph_Income_Schooling[((j+1)*length_per_set
    ):]
    # Concatenate 2 train set into 1 train set to train the model and test on validation
    set
    X_train_kfold = np.concatenate((X_first_part_train_kfold, X_second_part_train_kfold),
     axis=0)

    # Get the i_th set after split the y_train_feature into 5 equal parts
    y_val = y_train_Polio_Diph_Income_Schooling[(j*length_per_set):((j+1)*length_per_set)
    ].reshape(-1,1)
    # Get all values before y_val set after split the y_train_feature into 5 equal parts
    y_first_part_train_kfold = y_train_Polio_Diph_Income_Schooling[:(j*length_per_set)].
    reshape(-1,1)
    # Get all values after y_val set after split the y_train_feature into 5 equal parts
    y_second_part_train_kfold = y_train_Polio_Diph_Income_Schooling[((j+1)*length_per_set
    ):].reshape(-1,1)
    # Concatenate 2 train set into 1 train set to train the model and test on validation
    set
    y_train_kfold = np.concatenate((y_first_part_train_kfold, y_second_part_train_kfold),
     axis=0).reshape(-1,1)

    # Train model with the first model
    lr = OLSLinearRegression().fit(X_train_kfold, y_train_kfold)
    y_val_pred = lr.predict(X_val)
    RMSE[0] += rmse(y_val, y_val_pred)
RMSE[0] /= 5
```

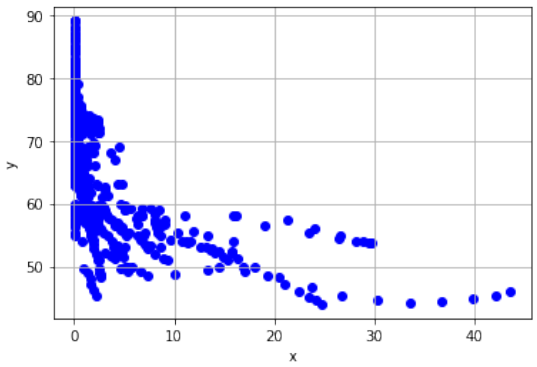
- In the implementation above, firstly, we have to preprocessing to get our first model.
  - This time, I not use the data in type *numpy array* but I will take full advantage of the properties of Dataframe provided. We just need to list all the header names of the columns we want to get, the Dataframe will get these columns for you.
  - For later easier calculation I will convert them to *numpy array*.
- Now we got the data of our model, next step we will use 5-fold cross validation technique to find the average of $RMSE$ for 5 validation times.
- And the way I implement the algorithm 5-fold cross validation here is totally the same in the second requirement.
- And in each iteration, we will train the model with $k-1$ folds and validate it on remaining 1 fold.
- After 5 iteration we will get mean of the $RMSE$. Therefore we get the average of $RMSE$ of this model.

4.3.2.b My second model

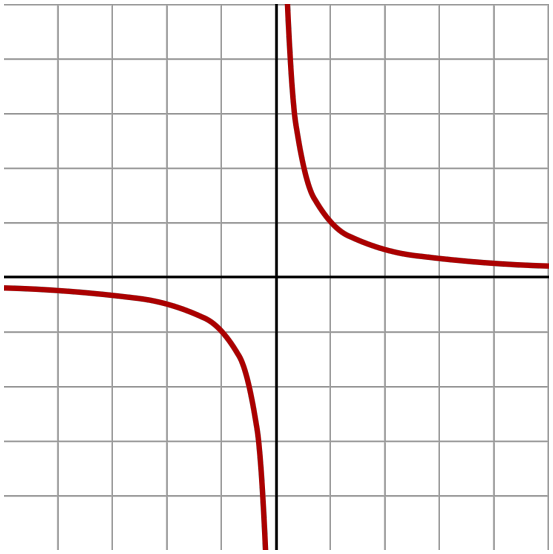
My second model is:

$$\text{Life expectancy} = w_1 \times \text{Schooling} + w_2 \times \frac{1}{HIV/AIDS}$$

- The idea of this model come from the shape of graph $HIV/AIDS$. We can recognize that the shape of this graph is similar to the shape of rectangular hyperbola shown below. The rectangular hyperbola I references from this site [13]. And because *Schooling* is the best feature model proved in the second requirement so I also add it to our model. Therefore, we will get the above model.



(a) *HIV/AIDS*



(b) *Rectangular_hyperbola*

- We can easily see that in the first quadrant of coordinate system, the shapes of these 2 figures are quite the same. So we can create a model from these relevant shapes.
- Now I will explain deeply on the idea of my implementation listed below.

```
# My second model:
# LE = w1*(Schooling) + w2*(1/HIVS)
X_train_Schooling = X_train_clone[['Schooling']].to_numpy().reshape(-1,1)
X_train_HIVS = X_train_clone[['HIV/AIDS']].to_numpy().reshape(-1,1)
X_train_HIVS = 1/X_train_HIVS
X_train_Schooling_HIVS = np.hstack((X_train_Schooling, X_train_HIVS))

y_train_Schooling_HIVS = y_train_clone.to_numpy()

for j in range(5):
    # Get the i_th set after split the X_train_feature into 5 equal parts
    X_val = X_train_Schooling_HIVS[(j*length_per_set):((j+1)*length_per_set)]
    # Get all values before X_val set after split the X_train_feature into 5 equal parts
    X_first_part_train_kfold = X_train_Schooling_HIVS[:(j*length_per_set)]
    # Get all values after X_val set after split the X_train_feature into 5 equal parts
    X_second_part_train_kfold = X_train_Schooling_HIVS[((j+1)*length_per_set):]
    # Concatenate 2 train set into 1 train set to train the model and test on validation
    set
    X_train_kfold = np.concatenate((X_first_part_train_kfold, X_second_part_train_kfold),
     axis=0)

    # Get the i_th set after split the y_train_feature into 5 equal parts
    y_val = y_train_Schooling_HIVS[(j*length_per_set):((j+1)*length_per_set)].reshape
    (-1,1)
    # Get all values before y_val set after split the y_train_feature into 5 equal parts
    y_first_part_train_kfold = y_train_Schooling_HIVS[:(j*length_per_set)].reshape(-1,1)
    # Get all values after y_val set after split the y_train_feature into 5 equal parts
    y_second_part_train_kfold = y_train_Schooling_HIVS[((j+1)*length_per_set):].reshape
    (-1,1)
    # Concatenate 2 train set into 1 train set to train the model and test on validation
    set
    y_train_kfold = np.concatenate((y_first_part_train_kfold, y_second_part_train_kfold),
     axis=0).reshape(-1,1)

    # Train model with the first model
    lr = OLSLinearRegression().fit(X_train_kfold, y_train_kfold)
    y_val_pred = lr.predict(X_val)
    RMSE[1] += rmse(y_val, y_val_pred)
RMSE[1] /= 5
```

- Similar to the first model, firstly, the preprocessing step is to get the data of our model.

  - Firstly, I will get column *Schooling* and column *HIV/AIDS*. With the column *HIV/AIDS* I will get inverse of this column which is $\frac{1}{HIV/AIDS}$.
  - Secondly, we just need to horizontally stack them together. Therefore, we get a *numpy array* of our second model.

- Now, after we get the data of our model, we will apply 5-fold cross validation technique to find the average of *RMSE* of 5 validation times of this model. These implementation is totally the same in the first model.

- So in the end, we will get the average of *RMSE* of the second model.

4.3.2.c My third model

My third model is:

Life expectancy = $w_1 \times$ (Income composition of resources + Schooling) + $w_2 \times$ (Polio + Diphtheria)

- The idea of this model come from my first model. Due to the shapes of the relative graphs and the relative *RMSE* values of these features, I will pack 2 models into 1 group.

- Therefore we will have 2 columns as 2 composite features will construct our third model. Actually, this model I base only on the shapes of graphs and the *RMSE* values of each individual features so there are no references for this model.

- Now I will explain deeply on the idea of my implementation listed below.

```
# My third model:
# LE = w1*(Income composition of resources + Schooling) + w2*(Polio + Diphtheria)
# Get array of (Income composition of resources + Schooling)
X_train_IncomeSchooling = X_train_clone['Income composition of resources'].to_numpy().reshape(-1,1) + X_train_clone['Schooling'].to_numpy().reshape(-1,1)
# Get array of (Polio + Diphtheria)
X_train_PolioDiph = X_train_clone['Polio'].to_numpy().reshape(-1,1) + X_train_clone['Diphtheria'].to_numpy().reshape(-1,1)
# Get array of column (Income composition of resources + Schooling) and column (Polio + Diphtheria)
X_train_IncomeSchooling_PolioDiph = np.hstack((X_train_IncomeSchooling, X_train_PolioDiph))
y_train_IncomeSchooling_PolioDiph = y_train_clone.to_numpy()

for j in range(5):
    # Get the i_th set after split the X_train_feature into 5 equal parts
    X_val = X_train_IncomeSchooling_PolioDiph[(j*length_per_set):((j+1)*length_per_set)]
    # Get all values before X_val set after split the X_train_feature into 5 equal parts
    X_first_part_train_kfold = X_train_IncomeSchooling_PolioDiph[:(j*length_per_set)]
    # Get all values after X_val set after split the X_train_feature into 5 equal parts
    X_second_part_train_kfold = X_train_IncomeSchooling_PolioDiph[((j+1)*length_per_set):]
    # Concatenate 2 train set into 1 train set to train the model and test on validation set
    X_train_kfold = np.concatenate((X_first_part_train_kfold, X_second_part_train_kfold), axis=0)

    # Get the i_th set after split the y_train_feature into 5 equal parts
    y_val = y_train_IncomeSchooling_PolioDiph[(j*length_per_set):((j+1)*length_per_set)].reshape(-1,1)
    # Get all values before y_val set after split the y_train_feature into 5 equal parts
    y_first_part_train_kfold = y_train_IncomeSchooling_PolioDiph[:(j*length_per_set)].reshape(-1,1)
    # Get all values after y_val set after split the y_train_feature into 5 equal parts
    y_second_part_train_kfold = y_train_IncomeSchooling_PolioDiph[((j+1)*length_per_set):].reshape(-1,1)
    # Concatenate 2 train set into 1 train set to train the model and test on validation set
    y_train_kfold = np.concatenate((y_first_part_train_kfold, y_second_part_train_kfold), axis=0).reshape(-1,1)

```

```
    # Train model with the first model
    lr = OLSLinearRegression().fit(X_train_kfold, y_train_kfold)
    y_val_pred = lr.predict(X_val)
    RMSE[2] += rmse(y_val, y_val_pred)
RMSE[2] /= 5
```

- Similar to the first and second models, firstly, the preprocessing step is to get the data of our model.
  - Firstly, I will get column *Incomecompositionofresources* add with the column *Schooling*. To add these 2 arrays I will convert them to *numpy arrays* and apply plus operator of *numpy*.
  - Secondly, I will get column *Polio* add with the column *Diphtheria*. To add these 2 arrays I will convert them to *numpy arrays* and apply plus operator of *numpy*.
  - Finally, I just need to horizontally stack them together. Therefore, we get a *numpy array* of our third model.
- Now, after I get the data of our model, I will apply 5-fold cross validation technique to find the average of *RMSE* of 5 validation times of this model. These implementation is totally the same in the first and second models.
- So in the end, I will get the average of *RMSE* of the third model.
- Right now, I have calculated the average of *RMSE* values of my 3 models. Next, I will find the *RMSE* with the lowest value, or find the best models among my 3 models. The code is shown below:

```
# Get index of models with minimum RMSE
my_best_model = np.argmin(RMSE, axis=0)
```

- Therefore, we will get the index of our best model.
- After getting the best model, we wil train this best model on the whole train set in file *train.csv*.

```
X_train_clone = train.iloc[:, :-1].copy() # Dataframe
y_train_clone = train.iloc[:, -1].copy() # Dataframe
X_test_clone = test.iloc[:, :-1].copy() # Dataframe
y_test_clone = test.iloc[:, -1].copy() # Dataframe

X_my_best_model_train = []
y_my_best_model_train = []
X_my_best_model_test = []
y_my_best_model_test = []

if(my_best_model == 0):
    # My first model:
    # LE = w1*Polio + w2*Diphtheria + w3*Income composition of resources + w4*Schooling
    print('First model is the best model of three\'s')
    X_my_best_model_train = X_train_clone[['Polio','Diphtheria','Income composition of
    resources','Schooling']].to_numpy()
    y_my_best_model_train = y_train_clone.to_numpy()
    X_my_best_model_test = X_test_clone[['Polio','Diphtheria','Income composition of
    resources','Schooling']].to_numpy()
    y_my_best_model_test = y_test_clone.to_numpy()
elif(my_best_model == 1):
    # My second model:
    # LE = w1*(Polio)^2 + w2*(Diphtheria)^2 + w3*(Income composition of resources)^2 + w4
    *(Schooling)^2
    print('Second model is the best model of three\'s')
    X_train_Schooling = X_train_clone[['Schooling']].to_numpy().reshape(-1,1)
    X_train_HIVS = X_train_clone[['HIV/AIDS']].to_numpy().reshape(-1,1)
    X_train_HIVS = 1 / X_train_HIVS
    X_my_best_model_test = np.hstack((X_train_Schooling, X_train_HIVS))
    y_my_best_model_test = y_train_clone.to_numpy()
else:
    # My third model:
    # LE = w1*(Income composition of resources + Schooling) + w2*(Polio + Diphtheria)
```

```
    # Get array of (Income composition of resources + Schooling)
    print('Third model is the best model of three\'s')
    X_train_IncomeSchooling = X_train_clone['Income composition of resources'].to_numpy()
    .reshape(-1,1) + X_train_clone['Schooling'].to_numpy().reshape(-1,1)
    # Get array of (Polio + Diphtheria)
    X_train_PolioDiph = X_train_clone['Polio'].to_numpy().reshape(-1,1) + X_train_clone['
    Diphtheria'].to_numpy().reshape(-1,1)
    # Get array of column (Income composition of resources + Schooling) and column (Polio
     + Diphtheria)
    X_my_best_model_train = np.hstack((X_train_IncomeSchooling, X_train_PolioDiph))
    y_my_best_model_train = y_train_clone.to_numpy()

    # Get array of (Income composition of resources + Schooling)
    X_test_IncomeSchooling = X_test_clone['Income composition of resources'].to_numpy().
    reshape(-1,1) + X_test_clone['Schooling'].to_numpy().reshape(-1,1)
    # Get array of (Polio + Diphtheria)
    X_test_PolioDiph = X_test_clone['Polio'].to_numpy().reshape(-1,1) + X_test_clone['
    Diphtheria'].to_numpy().reshape(-1,1)
    # Get array of column (Income composition of resources + Schooling) and column (Polio
     + Diphtheria)
    X_my_best_model_test = np.hstack((X_test_IncomeSchooling, X_test_PolioDiph))
    y_my_best_model_test = y_test_clone.to_numpy()


# Train feature Schooling in all training dataset
lr = OLSLinearRegression().fit(X_my_best_model_train, y_my_best_model_train)
w = lr.get_params().ravel()
y_my_best_model_test_pred = lr.predict(X_my_best_model_test)
print('w:', w)
```

- Firstly, I will preprocessing for 3 situations to get the data of 3 models. If the index of my_best_model is 0 then the first model is the best model, if the index of my_best_model is 1 then the second model is the best model, else the third model is the best model.
- In the end, we will train the best model on the whole dataset and predict the model from the result and print out the w value of this best model.
- Finally, we will print out the $RMSE$ value of this predicted result. The implementation is shown below.

```
# Gọi hàm RMSE (tự cài đặt hoặc từ thư viện) trên tập kiểm tra với mô hình my_best_model
print(rmse(y_my_best_model_test, y_my_best_model_test_pred))
```

- Now, we will come to the results section and explain these results.

4.3.3 Results and comments

This is the table of 3 average values of $RMSE$ results of my 3 relative models from 5-fold cross validation technique.

STT	Mô hình	RMSE
0	$w_1 \times$ Polio $+ w_2 \times$ Diphtheria $+ w_3 \times$ Income composition of resources $+ w_4 \times$ Schooling	9.641595
1	$w_1 \times$ Schooling $+ w_2 \times \frac{1}{HIV/AIDS}$	11.805521
2	$w_1 \times$ (Income composition of resources + Schooling) $+ w_2 \times$ (Polio + Diphtheria)	9.951234

- As we can see here, the first and the third models have the average of $RMSE$ are below 10. The reason of this $RMSE$ is that the $RMSE$ of these features in second requirement are the lowest so maybe the correlation between these features and target value are quite high so they might be good to use to predict our target value.
- The second model result is more than 10, which mean that this is not good to predict our target value.

- And from the table, the lowest $RMSE$ result is the first model, because their $RMSE$ of each individual features are the lowest in 10 features in second requirement. Therefore, the first model is my best model among 3 created models.

Below is the w result after we train my best model on the whole dataset.

```
First model is the best model of three's
w: [ 0.08707345  0.1896732  27.37948694  2.27304759]
```

Therefore, our regression formula is:

$$\begin{aligned}\text{Life expectancy} = \quad & 0.08707345 \times \text{ Polio } + \\ & 0.1896732 \times \text{Diphtheria } + \\ & 27.37948694 \times \text{Income composition of resources } + \\ & 2.27304759 \times \text{Schooling}\end{aligned}$$

- We can easily see the best model is the first model which $RMSE$ value is the lowest value among 3 models.

The $RMSE$ value of my first model is:

```
8.362637897580539
```

- Compared with the $RMSE$ in the second requirement which is approximately 10.26, obviously, my best model give the better $RMSE$ value which is approximately 8.36.
- This can be explain due to the number of features of our model. Because my best model is made up from 4 features, these 4 features have the lowest $RMSE$ values among 10 features so they are quite good for using them to predict the target value. This idea of model construction I references from this site. [12]

5 References

[1] https://en.wikipedia.org/wiki/Linear_regression#Least-squares_estimation_and_related_techniques

[2] http://www.stat.yale.edu/Courses/1997-98/101/linreg.htm

[3] https://en.wikipedia.org/wiki/Ordinary_least_squares

[4] https://en.wikipedia.org/wiki/Moore%E2%80%93Penrose_inverse

[5] https://courses.ctda.hcmus.edu.vn/pluginfile.php/87134/mod_resource/content/2/lab04.ipynb

[6] https://facweb.cdm.depaul.edu/sjost/it403/documents/regression.htm#:~:text=The%20RMSE%20estimates%20the%20deviation,in%20a%20thin%20vertical%20rectangle.&text=where%20ei%20%3D%20yi,(1%20%2D%20r2).

[7] https://www.researchgate.net/post/Whats-the-acceptable-value-of-Root-Mean-Square-Error-RMSE-Sum-of-Squares-due-to-error-SSE-and-Adjusted-R-square#:~:text=Based%20on%20a%20rule%20of,more%20is%20acceptable%20as%20well.

[8] https://stackoverflow.com/questions/4601373/better-way-to-shuffle-two-numpy-arrays-in-unison#comment62932389_37710486

[9] https://neptune.ai/blog/cross-validation-in-machine-learning-how-to-do-it-right

[10] https://numpy.org/doc/stable/reference/generated/numpy.argmin.html

[11] https://archpublichealth.biomedcentral.com/articles/10.1186/s13690-020-00402-5

[12] https://stats.stackexchange.com/questions/291258/are-more-features-always-better

[13] https://vi.wikipedia.org/wiki/Hyperbol#/media/T%E1%BA%ADp_tin Rectangular_hyperbola.svg