Applied mathematics *University of Science*

FACULTY OF INFORMATION TECHNOLOGY

REPORT PROJECT 03
LINEAR REGRESSION

Lecturer: Nguyễn Văn Quang Huy
Phan Thị Phương Uyên
Vũ Quốc Hoàng
Lê Thanh Tùng

Student's Information:

Student's name: Nguyễn Thoại Đăng Khoa

Student's ID: 20127043

JULY 30TH, 2022
VNUHCM – UNIVERSITY OF SCIENCE

TABLE OF CONTENTS

I. INTRODUCTION

Machine Learning needs to be supervised for the computers to effectively and efficiently utilize their time and efforts. One of the top ways to do it is through linear regression.

Even the most careful managers can make mistakes in organizations. But today, we live in a world where automation is powering most industries, thereby reducing cost, increasing efficiency, and eliminate human error. This is dominated by the rising application of machine learning and artificial intelligence. So, what gives machines the ability to learn and understand large volumes of data? It is through the learning methodologies such as linear regression.

In this project, we will learn what linear regression is and its applications. 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). Linear regression has many practical uses. Most applications fall into one of the following two broad categories: If the goal is prediction, forecasting, or error reduction, linear regression can be used to fit a predictive model to an observed data set of values of the response and explanatory variables.

Reference to these links (1), (2):

- https://en.wikipedia.org/wiki/Linear_regression
- https://blog.imarticus.org/linear-regression-and-its-applications-in-machine-learning/

II. COMPLETION PROGRESS

NO.	REQUIREMENT	DETAIL	COMPLETE
1A	Use all 10 features given	- Train only once for 10 features on the entire training set ('train.csv'). - Show formula for regression model. - Report 1 result on test set ('test.csv').	100%
1B	Build a model using only 1 feature, find the model with the best results	- Test on all (10) features givens. - Requires using 5-fold Cross Validation method to find the best feature - Report 10 results for 10 models from 5-fold Cross Validation (average) - Show the formula for the best feature regression model - Report 1 result on the test set ('test.csv') for the best model found	100%
1C	Students build own models, find the model that gives the best results	- Build m different models (minimum 3), different from 1a and 1b - Requires using 5-fold Cross Validation method to find the best model - Report m results for m models from 5-fold Cross Validation (average) - Show the formula for the best regression model found by students - Report 1 result on the test set ('test.csv') for the best model found	100%
TOTAL COMPLETION			100%

III. PROCESSING DETAILS

3.1 Library and function used

```
import pandas as pd
import numpy as np
import copy
```

- In this project, I used these libraries: pandas, numpy, copy.
- I need pandas to import the dataset, which is in csv format, and use pandas dataframe to print report table.
- Lastly, copy library used for calling function deepcopy() (A deep copy of an object is **a copy whose properties do not share the same references (point to the same underlying values) as those of the source object from which the copy was made**)
- In this project, I have used these functions: read_csv(), to_numpy(), methods of OLSLinearRegression class, rmse(), numpy.random.permutation(), and some functions I built on my own will be explained in the next sections.

3.2. Handle details

3.2.1 Read data

```
# 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)

# convert to numpy array
X_train = X_train.to_numpy()
y_train = y_train.to_numpy()
X_test = X_test.to_numpy()
y_test = y_test.to_numpy()
```

IDEA

- Read data by using pandas library.
- Get the features X and the target value y for the training and test sets.
- Convert to numpy array for manipulation.

IMPLEMENT

- **Input:** File train, test (in csv format)
- Read csv files into a DataFrame by calling function read_csv() and pass the name of file (‘train.csv’ and ‘test.csv’) and assign to train variable and test variable.
- Get 10 features for training by using pandas.Dataframe.iloc[:,:-1] and assign to X_train variable (type: Dataframe).
- Get target value y in train dataset by using pandas.Dataframe.iloc[:,-1] for testing and assign to y_train variable (type: Series).
- Get 10 features for testing by using pandas.Dataframe.iloc[:,:-1] and assign to X_test variable (type: Dataframe).
- Get target value y in test dataset by using pandas.Dataframe.iloc[:,-1] for testing and assign to y_test variable (type: Series).
- **Output:** X_train, y_train, X_test, y_test (which is convert to numpy array).

3.2.2 Class OLSLinearRegression

```
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)
```

IDEA

- Build Linear regression class using least squares method.
- Find the solution of the equation Ax ≈ b, using this formula:

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

- The output and parameters in the linear regression are as follows:
  - A ➔ X
  - b ➔ y
  - x ➔ w (w: weight)
  - Ax ≈ b ➔ Xw ≈ y or Xw = y (y is called the regression line.)

IMPLEMENT

1) fit(self, X, y):

- **Input:** X_train, y_train (type: ndarray)
- **This class using least squares method (**used to determine the line of best fit for a set of data**)**
- **Feature vector W** will be calculated by this formula:

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

- In this function, we can know that it's correspondent to self.w = x X_pinv = $(A^TA)^{-1}A^T$, and y = b
- **Output:** None (return self)

2) get_params(self):

- **Input:** None (self)
- This function returns the existing feature vectors after training.
- **Output:** feature vector W (return self.w)

2) predict(self, X):

- **Input:** X_test (type: ndarray)
- This function creates y_hat (written $\hat{y}$) is **the predicted value of y (the dependent variable) in a regression equation by using formula:**

```
np.sum(self.w.ravel()* X, axis=1)
```

- **Output:** return predicted values of y

Note: All these formulas and functions, I refer to lab04 given by lecturer.

3.2.3 RMSE function

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

- **Input:** y_test, y_test_predict (type: ndarray)
- Root Mean Square Error (RMSE) is a standard way to measure the mean of error of a model in predicting quantitative data.
- It is calculated by using this formula:

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}}$$

$\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_n$ are predicted values

$y_1, y_2, \ldots, y_n$ are observed values

n is the number of observations

- In this function y_hat is $\hat{y}_i$ and y is y_i

- RMSE is a standard deviation of prediction errors or residuals. It indicates how spread out the data is around the line of best fit.
- **Output:** return the error of a model.

Note: This formula I refer to lab04 given by lecturer, change from MSE to RMSE.

3.2.4 Handle for requirement 1A

In this requirement, we need to build a model using all features.

```
# Phần code cho yêu cầu 1a
# init lr
lr = OLSLinearRegression().fit(X_train, y_train)

# print feature vector
print("Feature vectors W: ", lr.get_params())

# predict
y_test_predict = lr.predict(X_test)

# print result
print("RMSE on all features: ", rmse(y_test, y_test_predict))
print('-----------------------------------------')
prediction = pd.DataFrame({'Test': y_test, 'Predicted': y_test_predict})
print(prediction)
```

IDEA

- Calling fit function in OLSLinearRegression class to train on X_train, y_train (train once for all features).
- After training, we had feature vectors W.
- The predicted value of y can be calculated based on W by calling predict function in OLSLinearRegression class.
- Calculate the root mean squared error for **evaluating the quality of predictions.**
- Show the formula for the regression model **(**calculate y according to 10 features in **X)**

IMPLEMENT

- **Input:** X_train, y_train (type: ndarray)
- **Initialize** instance lr of OLSLinearRegression class.
- Calling lr.fit() function and pass X_train, y_train into.

- Calling lr.get_params() function to print feature vectors W to represent the regression formula later.
- Calculate the predicted value of y by using lr.predict() and pass the X_test into, then assign the result to y_test_predict.
- Calculate RMSE by calling rmse() function and pass y_test and y_test_predict.
- Using pandas.DataFrame to print the result table (assign to prediction variable).
- **Output:** RMSE value, the formula of a regression model with 10 features.

3.2.5 Handle for requirement 1B

In this requirement, we need to build a model using only 1 feature, find the model that gives the best results.

IDEA

- Apply 5-fold Cross validation method on each feature and calculate the average RMSE of single feature.
- The general idea is as follows:
    1. Shuffle the dataset randomly.
    2. Split the dataset into 5 groups
    3. For each unique group:
        - Take the group as a hold out or test data set
        - Take the remaining groups as a training data set
        - Fit a model on the training set and evaluate it on the test set
        - Retain the evaluation score and discard the model
    4. Summarize the skill of the model using the sample of model evaluation scores
- After processing, we got a report table for 10 results of 10 features.
- The best result corresponds to the smallest RMSE.
- Show the formula for the best feature regression model found.

IMPLEMENT

- **Input:** X_train, y_train (type: ndarray)
- **Output:** RMSE value of single feature model, the formula of a regression model with the best features.
- **First step:** Shuffle the dataset once

```
# make deep copy
X_train_clone = copy.deepcopy(X_train)
y_train_clone = copy.deepcopy(y_train)

# shuffle dataset
shuffler = np.random.permutation(len(X_train_clone))
X_train_clone = X_train_clone[shuffler]
y_train_clone  = y_train_clone[shuffler]
```

- Firstly, I make deep copy from X_train, y_train to X_train_clone, y_train_clone (so that the original data is not changed when shuffle).
- Then, shuffle dataset by using numpy.random.permutation()
- This shuffle method, I refer to this link (3): https://www.adamsmith.haus/python/answers/how-to-shuffle-two-numpy-arrays-in-unision-in-python
- If we split the data then the resulting sets won't represent the true distribution of the dataset. Therefore, we have to shuffle the original dataset **in order to minimise variance and ensure that the model will generalise well to new, unseen data points.**
- Then, shuffle dataset on X_train_clone, and y_train_clone.

- **Second step:** Apply 5-fold cross validation method

```
RMSE = np.zeros(10) #initialize RMSE array

for i in range(0, 10):
    X_train_feature = X_train_clone[:,i]
    y_train_feature = y_train_clone.copy()

    for k in range(0, 5):
        lr = OLSLinearRegression()

        # create test dataset
        X_k_val = X_train_feature[217*k:217*k + 217]
        y_k_val = y_train_feature[217*k:217*k + 217]

        # create training dataset
        if(k == 0):
            X_k_train = X_train_feature[217:] # find x_k_train
            y_k_train = y_train_feature[217:] # find y_k_train

        else:
            # find x_k_train
            X_k_train_1 = X_train_feature[0:217*k]
            X_k_train_2 = X_train_feature[217*k + 217:]
            X_k_train = np.concatenate((X_k_train_1,X_k_train_2), axis = 0)

            # find y_k_train
            y_k_train_1 = y_train_feature[0:217*k]
            y_k_train_2 = y_train_feature[217*k + 217:]
            y_k_train = np.concatenate((y_k_train_1,y_k_train_2), axis = 0)

        # convert to 2d array
        X_k_train = X_k_train.reshape(-1,1)
        y_k_train = y_k_train.reshape(-1,1)
        X_k_val = X_k_val.reshape(-1,1)
        y_k_val = y_k_val.reshape(-1,1)

        # training
        lr.fit(X_k_train, y_k_train)
        # predict
        y_k_val_pred = lr.predict(X_k_val)
        RMSE[i] += rmse(y_k_val, y_k_val_pred)

    # calculate mean rmse i_th
    RMSE[i] /= 5
```

- Since we need to test on all 10 features, i will run from 0 to 10 (for i in range (0 – 10))
- For each i, which will correspond to each column i-th in the train dataset, we extract from X_train_clone, y_train_clone assign to X_train_feature, and y_train_feature.
- On this i-th column (i-th feature), we will proceed to apply the 5-fold-cross validation method.
- It can be done as follows:
    - ✓ Divide 1085 data lines by 5, to make 5 parts with 217 data lines each.
    - ✓ There are 5 iterations: for each i-th iteration, a portion of 217 lines will be tested, and the rest will be trained. (for k in range (0 –5))
    - ✓ A portion of 217 lines i-th will be correspondent to 217 lines i-th in X_train_feature, y_train_feature, and assign it to X_k_val, and y_k_val (this is test part or validate part)
    - ✓ The rest in X_train_feature, y_train_feature we assign to X_k_train, y_k_train (this is training part)
    - ✓ After had X_k_train, y_k_train, then call lr.fit(X_k_train, y_k_train)
    - ✓ Calculate predicted value by calling lr.fit(X_k_val) and then assign the result to y_k_val_pred
    - ✓ Calling rmse(y_k_val, y_k_val_pred), and assign the result to RMSE array which RMSE[i] is correspondent to feature i-th.
    - ✓ After going through 5 iterations, we will average the RMSE by dividing by 5 to get the average RMSE for the i-th feature.
    - ✓ Repeat these steps, until the last feature, we will have 10 RMSE average results for 10 features (in RMSE array which RMSE[i] is the RMSE for the i-th feature), choose the smallest RMSE we will know the best feature.

- Then, retrain the best feature model with the best feature on the entire training set. After we get the index of the column corresponding to the best feature, we extract it from X_train with this index and assign it to X_train_best_feature. Then call fit(X_train_best_feature, y_train). Next, we extract from X_test with this index and assign it to X_test_best_feature, we can calculate the predicted value by calling lr.predict(X_test_best_feature) and assign the result to y_test_predict_best_feature.

- Finally, we call rmse(y_test, y_test_predict_best_feature) to calculate the root mean squared error.

Note: I implemented all of these based on the idea suggested by Ms. Uyen

3.2.6 Handle for requirement 1C

- In this requirement, students build their own models, find the model that gives the best results.

IDEA

- My idea based on **Feature Importance** method. The coefficients of linear models are commonly interpreted as the **Feature Importance** of related variables. In general, feature importance refers to how useful a feature is at predicting a target variable. Assume that we feed the dataset to a linear model, then train the model, and make a prediction, at this point, we have the values of coefficients, and then these coefficients can be used as feature importance (ONLY IF the dataset was standardized before training). Based on this, we can drop or ignore some unimportant features to speed model training up.

- After normalizing the data, I will take out the top 4 most important features to model, from which to draw conclusions.

- I built models as follows:
  - Model 1: 2 most important features
  - Model 2: 3 most important features
  - Model 3: 4 most important features
  - Model 4: 3 most important features + 1 new feature

IMPLEMENT

- **First step**: I normalize the dataset to find feature importance (based on coefficient and standard deviation):

```
# Find 3 features importance
stdev = []
x = train[["Adult Mortality", "BMI", "Polio", "Diphtheria", "HIV/AIDS", "GDP", "Thinness age 10-19", "Thinness age 5-9",
       "Income composition of resources", "Schooling"]]
y = train["Life expectancy"]

lr = OLSLinearRegression().fit(X_train, y_train)

lr_coef = lr.get_params() # get feature vector

tables = pd.DataFrame(lr_coef, x.columns, columns=['coefficient'])
tables.coefficient = tables.coefficient.abs()

# Transforming the units of coefficients

# calcualte standard deviation for each feature
stdeviations = []
for i in x.columns:
    stdev = train[i].std()
    stdeviations.append(stdev)

tables["standard_deviation"] = np.array(stdeviations).reshape(-1,1)
tables["importance"] = tables["coefficient"] * tables["standard_deviation"]

# normalize the importance column in range of [0, 100].
tables['importance_normalized (0 - 100)'] = 100*tables['importance'] / tables['importance'].max()

# display tables
tables.head(10)
```

- **Input:** train variable (type: DataFrame).
- Get all features assigned to x (type: DataFrame), the target value assigned to y (type: Series) via train variable.
- Init instance lr of OLSLinearRegression class, then call fit(X_train, y_train) to calculate coefficients, and call lr.get_params() to get feature vectors assigning to lr_coef variable and insert to table as column "coefficient".
- Here, we can ignore the sign values because negative sign states an inversely proportional correlation (use abs() on the entire feature vector array).
- Then, I use the Pandas std() method which stands for *Standard deviation* could help us to convert the units of coefficients to the same **because we can't compare magnitudes that are different units.**
- Calculate standard deviation for each feature and assign to stdeviations array and insert to table as column "standard_deviation"
- The **importance value** we want can be calculated by this formula:

coefficient * standard deviation

- Then normalize the importance column in range of [0, 100] by this formula:

(importance * 100) / **max**(importance)

- Then, insert importance value to table as column "importance", and importance value after normalizing to table as column "importance_normalized (0-100)"
- Display result table by using DataFrame.head() method.
- **Output:** The result table with corresponding feature importance value for each feature

	coefficient	standard_deviation	importance	importance_normalized (0 - 100)
Adult Mortality	0.015101	123.993845	1.872476	26.017530
BMI	0.090220	19.587936	1.767223	24.555071
Polio	0.042922	23.775524	1.020489	14.179405
Diphtheria	0.139289	21.706896	3.023534	42.011163
HIV/AIDS	0.567333	4.869510	2.762633	38.386007
GDP	0.000101	10318.169038	1.039711	14.446500
Thinness age 10-19	0.740713	4.659256	3.451174	47.953092
Thinness age 5-9	0.190936	4.694674	0.896381	12.454967
Income composition of resources	24.505974	0.178419	4.372321	60.752170
Schooling	2.393517	3.006864	7.196978	100.000000

- After that, we can get **4 most important** features: **Schooling, Income composition of resources, Thinness age 10-19, and Diphtheria** (100 > 60 > 47 > 42)

- Then, I built 4 models based on these 4 features to draw conclusions.

Note: In this implementation above, the way to create report table, and find feature importance I learned from these links (4), (5), (6):

- https://www.youtube.com/watch?v=Y9nN_6sbXkM&t=1211s
- https://sefiks.com/2019/12/20/a-gentle-introduction-to-feature-importance-in-machine-learning/
- https://towardsdatascience.com/feature-importance-in-linear-models-four-often-neglected-but-crucial-pitfalls-e5c513e45b18

- **Build model 1**
  - **Input:** X_train_clone, y_train_clone (type: ndarray)
  - About the processing part is quite similar to part 1b, so I won't explain it again. The only difference is that we only need to train 2 features, it means X_train_feature will now only include the two most important features: Schooling (9), Income composition of resources (8).
  - **Output:** RMSE value

- **Build model 2**:
  - **Input:** X_train_clone, y_train_clone (type: ndarray)
  - About the processing part is quite similar to part 1b, so I won't explain it again. The only difference is that we only need to train 3 features, it means X_train_feature will now only include the three most important features: Schooling (9), Income composition of resources (8), Thinness age 10-19 (6).
  - **Output:** RMSE value

- **Build model 3**:
  - **Input:** X_train_clone, y_train_clone (type: ndarray)
  - About the processing part is quite similar to part 1b, so I won't explain it again. The only difference is that we only need to train 4 features, it means X_train_feature will now only include the four most important features: Schooling (9), Income composition of resources (8), Thinness age 10-19 (6), and Diphtheria (3).
  - **Output:** RMSE value

- **Build model 4**:
  - **Input:** X_train_clone, y_train_clone (type: ndarray)
  - About the processing part is quite similar to part 1b, so I won't explain it again. The only difference is that we only need to train 4 features, means X_train_feature will now only include the four most important features: Schooling (9), Income composition of resources (8), Thinness age 10-19 (6), and **Vaccination rate among 1-year-olds** (10)
  - **Output:** RMSE value

  ***Note*: Vaccination rate among 1-year-olds** (10) is new feature that I created.

- The process of creating this **Vaccination rate among 1-year-olds** feature will be presented as follows:

  - ❖ In the dataset in https://www.kaggle.com/datasets/kumarajarshi/life-expectancy-who, I see 2 features related to vaccination rate information. These are: **Polio** (Pol3) immunization coverage among 1-year-olds (%), and **Diphtheria** tetanus toxoid and pertussis (DTP3) immunization coverage among 1-year-olds (%) (same unit)

  - ❖ My idea is to combine these 2 features, calculate the average of the 2 features to get the value for the new feature. This feature I named **Vaccination rate among 1-year-olds** (index column is 10)

  - ❖ Then, I will replace the **Diphtheria** feature with the feature I just created (**Vaccination rate among 1-year-olds)**. I build a model based on this new feature that had just been replaced, to see if the results are more optimal than the old model.

  - ❖ Based on the results, I will write comments and evaluations for this model.

- In terms of implementation, we can do the following:

```
new_feature = (X_train_clone[:,2] + X_train_clone[:,3]) / 2

X_train_clone = np.append(X_train_clone, new_feature.reshape(-1,1), axis = 1)
```

  - ❖ Calculate the mean of Polio (2), and Diphtheria (3) extracted from column 2nd, and column 3rd of X_train_clone and assign to new_feature

  - ❖ Then, add to X_train_clone a new value column that is the value of new_feature or can be understood as the value of **Vaccination rate among 1-year-olds** feature (index column of the new feature is 10).

  - ❖ Reconstruct the model of 4 features but replaced the feature **Diphtheria** with the feature **Vaccination rate among 1-year-olds.**

- **After having 4 model results,** I find the best model based on RMSE, shows the formula for that best regression model, and report 1 result on the test set (test.csv) for the best model found.

IV. RESULT ANALYSIS

4.1. Requirement 1A

- **Feature vectors W**

```
Feature vectors 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]
```

Comment:

The linear regression equation in the project we used is:

$$f(x) = wx$$

- The job of the model is to predict a real-value *y* for an unseen value of the feature *x*. Therefore, our job is to find the optimal set of values *w* which will minimize the error between the predictions made by the model *f(x)* and the actual results *y*.
- **We can interpret this as a set of regression coefficients** indicating the direction of the relationship between a feature variable and the response variable.
    - A positive sign indicates that as the feature variable increases, the response variable also increases.
    - A negative sign indicates that as the feature variable increases, the response variable decreases.
- Based on the above results, we see that most of the results have a positive sign, meaning that when the values of these features increase, the life expectancy value also increases. Only two features: **HIV/AIDS** and **GDP** have negative signs indicating that, as the values of these two variables increase, the target values will decrease.

- **RMSE value on 10 features (on test.csv)**

```
RMSE on all features:  7.064046430584466
```

Comment:

- **RMSE** is the way to evaluate the quality of predictions.
- **RMSE** on all features is 7.06, which means the mean of error of the predicted value and the actual value is 7.06.
- We can say that the error of the prediction model for the target value of **life expectancy** is about 7 years old.
- If **RMSE** is small, this generally means our model is good at predicting our observed data, and if **RMSE** is large, this generally means our model is failing to account for important features underlying our data.
- **RMSE** on 10 features is small, and it is likely that this model with 10 features will be the best with the best RMSE value since there are many relevant features that support finding the target value (because the data in the project has been preprocessed like select rows related to the top 95 most populous countries and remove columns that are less relevant to the target value).

- **Show formula for this regression model**

$$\text{Life expectancy} = 0.015101 * AdultMortality + 0.090220 * BMI + 0.042922 * Polio + 0.139289 * Diphtheria - 0.567333 * HIV/AIDS - 0.000101 * GDP + 0.740713 * ThinnessAge(10-19) + 0.190936 * ThinessAge(5-9) + 24.505974 * IncomeCompositionOfResources + 2.393517 * Schooling$$

Comment:

- Since the requirement is to use 10 features, simple linear regression will become multiple linear regression which is a method we can use to quantify the relationship between two or more feature variables and a response variable.
- The above result is a regression formula (calculating y by 10 features in X), with the general formula being:

$$y_i = \beta_0 + \beta_1 x_{i,1} + \beta_2 x_{i,2} + \ldots + \beta_k x_{i,k} + \epsilon_i.$$

4.2. Requirement 1B

- **Cross-validation (average RMSE of 10 features)**

```
Cross-validation:  [46.33803678 27.96474329 18.01199879 16.03470662 67.08694932 60.21667488
 51.92118693 51.85398937 13.31398403 11.80270604]
```

- **Report table (from cross validation method)**

NO.	Model with 1 feature	RMSE
1	Adult Mortality	46.338037
2	BMI	27.964743
3	Polio	18.011999
4	Diphtheria	16.034707
5	HIV/AIDS	67.086949
6	GDP	60.216675
7	Thinness age 10-19	51.921187
8	Thinness age 5-9	51.853989
9	Income composition of resources	13.313984
10	Schooling	11.802706

Comment:

- Based on the RMSE measure, it can be seen that the model for the **Schooling** feature is the best due to the lowest RMSE, and the model for the **HIV/AIDS** feature is the worst due to the highest RMSE.
- Lower values of RMSE indicate a **better fit**. Inversely, if the RMSE for the test set is much higher than that of the training set, **it is likely that we've badly overfit the data (**a model that tests well in the sample, but has little predictive value when tested out of the sample**)**
- In my opinion, **Schooling** is the most important and best feature because Lifespan can be improved due to education. A Master's degree brings about a life expectancy of 14.7 years more than people who had not finished high school, and a life expectancy of 8.3 years more than people who had finished high school (Singh & Lee, 2021). The chance to get access to education can depend on their race and that opportunity may lead to a longer life for some than for other.

That information I referred to this link (7):

https://www.gvsu.edu/cms4/asset/298A9768-AC3E-2839-8911A1BAD580BF4F/group_20.pdf

- **Feature vector W (the best feature)**

```
Feature vector W:  [5.5573994]
```

Comment:

- This is the coefficient W in the linear regression model for the best feature which is **Schooling**.
- A positive sign indicates that as the feature **Schooling** increases (while keeping other features stable), the target value **Life expectancy** also increases.
- In other words, if **Schooling** increases by 1 unit, **Life expectancy** increases by ~ 5.5.

- **RMSE value (the best feature) (on test.csv)**

```
RMSE on best feature model:  10.260950391655376
```

Comment:

- This is the RMSE value of the best feature after retraining the model with the best feature over the entire training set.
- There is a difference between this RMSE value and the RMSE value reported in the report table because the RMSE value in the table is the averaged value based on the 5-fold Cross validation method (i.e., train only on 217 x 4 samples), and this RMSE value is the retrained value over the entire training set.

- **Show formula for this best feature regression model**

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

➔ There is a formula for the best feature regression model in which the best feature is **Schooling**.

4.3. Requirement 1C

- As explained above, to build my own model, I learned how to find **Feature Importance**. Based on this method, I will extract the 4 most important features to build the model.
  - Model 1: 2 most important features (Schooling, Income composition of resources)
  - Model 2: 3 most important features (Schooling, Income composition of resources, Thinness age 10- 19)
  - Model 3: 4 most important features (Schooling, Income composition of resources, Thinness age 10- 19, Diphtheria)
  - Model 4: 3 most important features + 1 new feature (Schooling, Income composition of resources, Thinness age 10- 19, Vaccination rate among 1-years-old)
- Based on this, we can drop or ignore some unimportant features to speed model training up.

- **REPORT RESULT**

NO.	Model	RMSE
1	Using 2 features (Schooling, Income composition of resources)	11.2983
2	Using 3 features (Schooling, Income composition of resources, Thinness age 10-19)	9.4950
3	Using 4 features (Schooling, Income composition of resources, Thinness age 10-19, Diphtheria)	8.4157
4	Using 4 features with one new added (Schooling, Income composition of resources, Thinness age 10-19, Vaccination rate among 1-years-old)	8.4220

Comment:

- Based on the Feature Importance method, we can see that the better the feature selection, the smaller RMSE value.
- We can double-check with the **2 least** important features, **Thinness age 5-9**, and **Polio** and RMSE value returned ~ 17 > 11.2983 of **2 most** important features (**Schooling**, **Income composition of resources**)
- Therefore, the **more important** features are selected, the **better** our model will be (11.2983 > 9.4950 > 8.4157).
- This method becomes very useful to overcome with **overfitting** problem that I mentioned above. It helps us in determining the smallest set of features that are needed to predict the response variable with high accuracy.
- In model 4, I built a new feature **Vaccination rate among 1-years-old** (based on **Polio** and **Diphtheria**), and the RMSE results show that this model is **worse** than the model that chooses the **4 most important** features.

➔ This shows that when we create a new feature or choose a feature to build a model, this feature must be filtered scientifically, and cannot be based on any individual's opinion, otherwise **the performance** of the model will be **adversely affected**.

- In general, the best model I have built is a model using 4 features: **Schooling**, **Income composition of resources**, **Thinness age 10-19**, and **Diphtheria**.

- After **retraining** my best model on the entire training set, I got the following results:

```
Feature vectors W:  [ 2.39880247 29.55036258  0.93308232  0.19428694]
```

➔ This is the set coefficients W in my best linear regression model.

```
RMSE on my best model:  7.245848160503847
```

➔ This RMSE value of my best model is retrained over the entire training set and tested on **test.csv**. This result is close to the result of training once for 10 features in requirement 1A, showing that the feature selection method based on feature importance is quite accurate.

- **Show formula for my best regression model**

$$\text{Life expectancy} = 2.39 * Schooling + 29.55 * IncomeCompositionOfResource + 0.93 * ThinnessAge(10-19) + 0.19 * Diphtheria$$

V. REFERENCES

1. https://en.wikipedia.org/wiki/Linear_regression
2. https://blog.imarticus.org/linear-regression-and-its-applications-in-machine-learning/
3. https://www.adamsmith.haus/python/answers/how-to-shuffle-two-numpy-arrays-in-unision-in-python
4. https://www.youtube.com/watch?v=Y9nN_6sbXkM&t=1211s
5. https://sefiks.com/2019/12/20/a-gentle-introduction-to-feature-importance-in-machine-learning/
6. https://towardsdatascience.com/feature-importance-in-linear-models-four-often-neglected-but-crucial-pitfalls-e5c513e45b18
7. https://www.gvsu.edu/cms4/asset/298A9768-AC3E-2839-8911A1BAD580BF4F/group_20.pdf