HO CHI MINH CITY NATIONAL UNIVERSITY
UNIVERSITY OF NATURE SCIENCE
FACULTY OF INFORMARTION TECHNOLOGY
----------o0o----------

PROJECT 03: DATA FITTING

Applied mathematics and statistics for information technology

Lecturers:

Ph.D Vũ Quốc Hoàng

Phan Thị Phương Uyên

Nguyễn Văn Quang Huy

Trần Thị Thảo Nhi

Student:

Đặng Ngọc Tiến

Ho Chi Minh, August 2022

Table of Contents

I. Student information and complete progress:

a. Student information:

- Name: Đặng Ngọc Tiến
- ID student: 20127641
- Class: 20CLC11

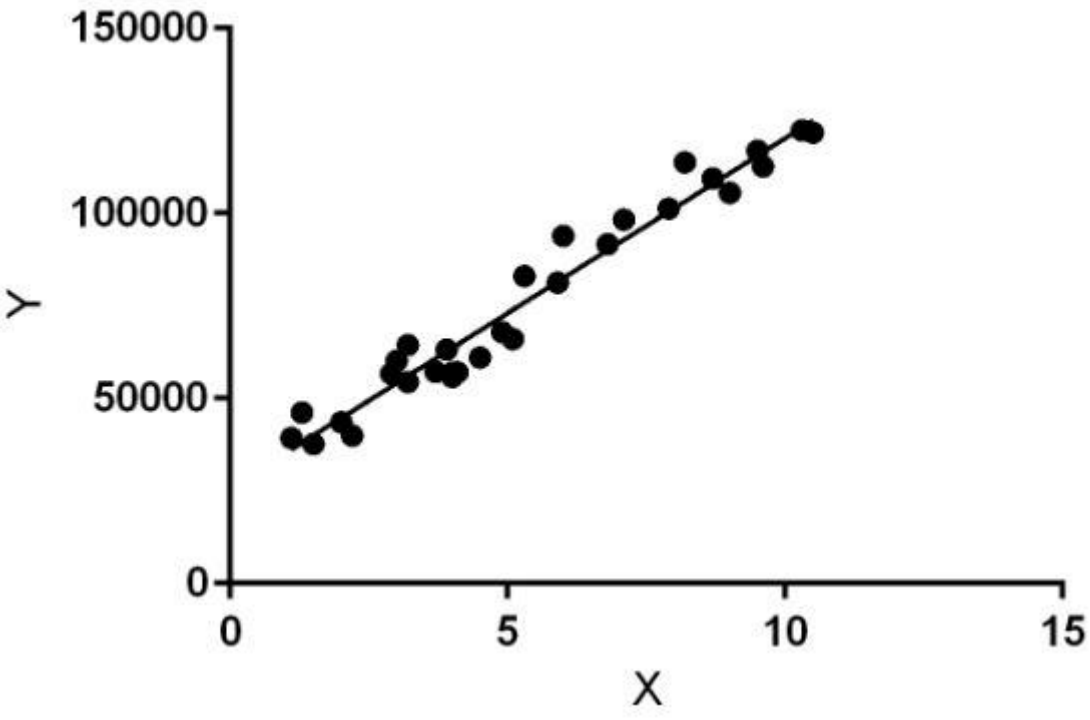
b. Complete progress:

Task	Complete
Task a	100%
Task b	100%
Task c	100%

II. Introduction to the project

a. Linear Regression[1]:

Linear Regression is a machine learning algorithm based on supervised learning. It performs a regression task. Regression models a target prediction value based on independent variables. It is mostly used for finding out the relationship between variables and forecasting. Different regression models differ based on the kind of relationship between dependent and independent variables they are considering, and the number of independent variables getting used.



Linear regression performs the task to predict a dependent variable value (y) based on a given independent variable (x). So, this regression technique finds out a linear relationship between x (input) and y(output). Hence, the name is Linear Regression.

In the figure above, X (input) is the work experience and Y (output) is the salary of a person. The regression line is the best fit line for our model.

$$y = \theta_1 + \theta_2.x$$

While training the model we are given :

x: input training data (univariate – one input variable(parameter))

y: labels to data (supervised learning)

When training the model it fits the best line to predict the value of y for a given value of x. The model gets the best regression fit line by finding the best θ1 and θ2 values.

θ1: intercept

θ2: coefficient of x

Once we find the best θ1 and θ2 values, we get the best fit line. So when we are finally using our model for prediction, it will predict the value of y for the input value of x.

Cost Function (J):

By achieving the best-fit regression line, the model aims to predict y value such that the error difference between predicted value and true value is minimum. So, it is very important to update the θ1 and θ2 values, to reach the best value that minimize the error between predicted y value (pred) and true y value (y).

$$J = \frac{1}{n}\sum_{i=1}^{n}(pred_i - y_i)^2$$

Cost function(J) of Linear Regression is the Root Mean quared Error (RMSE) between predicted y value (pred) and true y value (y).

b. K-fold Cross validation[2]:

- **Overview**

  In this tutorial, we'll talk about two cross-validation techniques in machine learning: the k-fold and leave-one-out methods. To do so, we'll start with the train-test splits and explain why we need cross-validation in the first place. Then, we'll describe the two cross-validation techniques and compare them to illustrate their pros and cons.

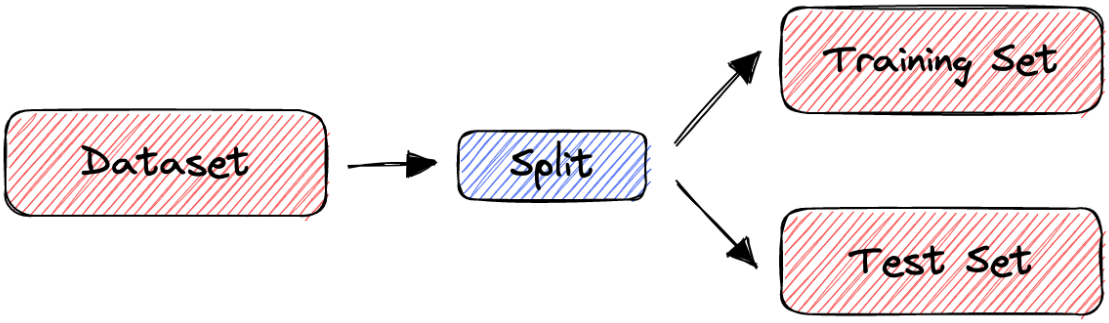
- **Train-Test Split Method**

  An important decision when developing any machine learning model is how to evaluate its final performance. To get an unbiased estimate of the model's performance, we need to evaluate it on the data we didn't use for training.

  The simplest way to split the data is to use the train-test split method. It randomly partitions the dataset into two subsets (called training and test sets) so that the predefined percentage of the entire dataset is in the training set.

Then, we train our machine learning model on the training set and evaluate its performance on the test set. In this way, we are always sure that the samples used for training are not used for evaluation and vice versa.

Visually, this is how the train-test split method works:



- **Introduction to Cross-Validation**

  However, the train-split method has certain limitations. When the dataset is small, the method is prone to high variance. Due to the random partition, the results can be entirely different for different test sets. Why? Because in some partitions, samples that are easy to classify get into the test set, while in others, the test set receives the 'difficult' ones.

  To deal with this issue, we use cross-validation to evaluate the performance of a machine learning model. In cross-validation, we don't divide the dataset into training and test sets only once. Instead, we repeatedly partition the dataset into smaller groups and then average the performance in each group. That way, we reduce the impact of partition randomness on the results.

  Many cross-validation techniques define different ways to divide the dataset at hand. We'll focus on the two most frequently used: the k-fold and the leave-one-out methods.

- **K-Fold Cross-Validation**

  In k-fold cross-validation, we first divide our dataset into k equally sized subsets. Then, we repeat the train-test method k times such that each time one of the k subsets is used as a test set and the rest k-1 subsets are used together as a training set. Finally, we compute the estimate of the model's performance estimate by averaging the scores over the k trials.

  For example, let's suppose that we have a dataset S $= \{x_1, x_2, x_3, x_4, x_5, x_{6}\}$ containing 6 samples and that we want to perform a 3-fold cross-validation.

  First, we divide S into 3 subsets randomly. For instance:

$$S_1 = \{x_1, x_2\}$$
$$S_2 = \{x_3, x_4\}$$
$$S_3 = \{x_5, x_6\}$$

Then, we train and evaluate our machine-learning model 3 times. Each time, two subsets form the training set, while the remaining one acts as the test set. In our example:

Finally, the overall performance is the average of the model's performance scores on those three test sets:

$$\text{overall score} = \frac{score_1 + score_2 + score_3}{3}$$

- **Leave-One-Out Cross-Validation**
  In the leave-one-out (LOO) cross-validation, we train our machine-learning model n times where n is to our dataset's size. Each time, only one sample is used as a test set while the rest are used to train our model.

  We'll show that LOO is an extreme case of k-fold where k = n. If we apply LOO to the previous example, we'll have 6 test subsets:

$$S_1 = \{x_1\}$$
$$S_2 = \{x_2\}$$
$$S_3 = \{x_3\}$$
$$S_4 = \{x_4\}$$
$$S_5 = \{x_5\}$$
$$S_6 = \{x_6\}$$

  Iterating over them, we use S \ S_i as the training data in iteration $i = 1,2, \dots, 6$, and evaluate the model on S_i:

The final performance estimate is the average of the six individual scores:

$$\text{overall score} = \frac{score_1 + score_2 + score_3 + score_4 + score_5 + score_6}{6}$$

c. Life Expectancy:

Life expectancy data were collected from WHO and the United Nations website from 2000 to 2015 across all countries.

The dataset has 2938 rows and 22 columns. The meanings and data types of each column are shown in the following table:

STT	Tên cột	Ý nghĩa	Kiểu dữ liệu
1	Country	Tên quốc gia	String
2	Year	Năm	Integer
3	Status	Đất nước phát triển hay đang phát triển	String
4	Life expectancy	Tuổi thọ trung bình	Decimal
5	Adult mortality	Tỉ lệ tử vong ở người trưởng thành trên 1000 dân số (độ tuổi từ 15 đến 60)	Decimal
6	Infant deaths	Số trẻ sơ sinh tử vong trên 1000 dân số	Decimal
7	Alcohol	Mức tiêu thụ rượu bình quân đầu người (từ 15 tuổi) - tính trên lít rượu nguyên chất	Decimal
8	percentage expenditure	Chi tiêu cho y tế tính theo phần trăm GDP (%)	Decimal
9	Hepatitis B	Tỉ lệ tiêm chủng viêm gan B ở trẻ 1 tuổi (%)	Decimal
10	Measles	Số ca mắc bệnh sởi được báo cáo tính trên 1000 dân số	Integer
11	BMI	Chỉ số khối lượng cơ thể (BMI) trung bình của toàn bộ dân số	Decimal
12	under-five deaths	Số lượng người tử vong dưới 5 tuổi trên 1000 dân số	Integer
13	Polio	Tỉ lệ tiêm chủng bại liệt (Pol3) ở trẻ 1 tuổi (%)	Decimal
14	Total expenditure	Chi tiêu chung cho y tế tính tính theo phần trăm tổng chi tiêu của chính phủ	Decimal
15	Diphtheria	Tỉ lệ tiêm chủng giải độc uốn ván và ho gà (DTP3) ở trẻ 1 tuổi (%)	Decimal
16	HIV/AIDS	Tỉ lệ tử vong trên 1000 trẻ nhiễm HIV/AIDS (từ 0 đến 4 tuổi)	Decimal
17	GPD	Tổng sản phẩm quốc nội bình quân đầu người (đơn vị USD)	Decimal
18	Population	Dân số quốc gia	Decimal
19	thinness 1-19 years	Tỉ lệ gầy ốm ở trẻ em và thanh thiếu niên từ 10-19 tuổi (%)	Decimal
20	thinness 5-9 years	Tỉ lệ gầy ốm ở trẻ em từ 5-9 tuổi (%)	Decimal
21	Income composition of resources	Chỉ số phát triển con người (HDI) tính theo thành phần thu nhập tài nguyên (thuộc [0, 1])	Decimal
22	Schooling	Số năm đi học	Decimal

See more[3].

In this project, the above data has been performed the following preprocessing steps:

1. Remove data lines with incomplete information (with NaN value in line)
2. Select only the lines related to the top 95 countries with the largest population
3. Normalize and rename some features: thinness 1-19 years → Thinness age 10-19, thinness 5-9 years → Thinness age 5-9
4. Remove 2 columns with string values: Country, Status
5. Based on the correlation measure, remove the 9 columns that are least relevant to the goal value (Life expectancy): Population, Measles, Year, infant deaths, Total expenditure, under-five deaths, Hepatitis B, percentage expenditures, Alcohol

After preprocessing, the new data has:

- 1180 data lines
- 11 columns of data including:
  - 1 goal value y: Life expectancy
  - 10 features that explain X (features that help find target values) include: Adult Mortality, BMI, Polio, Diphtheria, HIV/AIDS, GDP, Thinness age 10-19, Thinness age 5-9, Income composition of resources, Schooling, Life expectancy

Students are provided with 2 files:

- train.csv: Contains 1085 samples used to train the model
- test.csv: Contains 95 samples used to test the model

	Adult Mortality	BMI	Polio	Diphtheria	HIV/AIDS	GDP	Thinness age 10-19	Thinness age 5-9	Income composition of resources	Schooling	Life expectancy
0	268.0	18.1	62.0	64.0	0.1	631.744976	17.7	17.7	0.470	9.9	59.9
1	272.0	17.6	67.0	67.0	0.1	669.959000	17.9	18.0	0.463	9.8	59.5
2	275.0	17.2	68.0	68.0	0.1	63.537231	18.2	18.2	0.454	9.5	59.2
3	279.0	16.7	66.0	66.0	0.1	553.328940	18.4	18.4	0.448	9.2	58.8
4	281.0	16.2	63.0	63.0	0.1	445.893298	18.6	18.7	0.434	8.9	58.6

III. Library and necessary functions:

- Import Numpy
  Create matrices and process mathematical operations

- Import Pandas
  Read file data .csv

- import matplotlib.pyplot as plt
- import seaborn as sns
  Correlation chart

- Import sklearn.model_selection from train_test_split[4]
  Split arrays or matrices into random train and test subsets.
  Quick utility that wraps input validation and next(ShuffleSplit().split(X, y)) and application to input data into a single call for splitting (and optionally subsampling) data in a oneliner.

  Parameters

  array: s*sequence of indexables with same length / shape[0]
  Allowed inputs are lists, numpy arrays, scipy-sparse matrices, or pandas dataframes.

  test_size: *float or int, default=None*
  If float, should be between 0.0 and 1.0 and represent the proportion of the dataset to include in the test split. If int, represents the absolute number of test samples. If None, the value is set to the complement of the train size. If train_size is also None, it will be set to 0.25.

  train_size: *float or int, default=None*
  If float, should be between 0.0 and 1.0 and represent the proportion of the dataset to include in the train split. If int, represents the absolute number of train samples. If None, the value is automatically set to the complement of the test size.

  random_state: *int, RandomState instance or None, default=None*
  Controls the shuffling applied to the data before applying the split. Pass an int for reproducible output across multiple function calls.

  Shuffle: *bool, default=True*
  Whether or not to shuffle the data before splitting. If shuffle=False then stratify must be None.

  Stratify: *array-like, default=None*
  If not None, data is split in a stratified fashion, using this as the class labels.

  Returns

  Splitting: list, length=2 * len(arrays)
  List containing train-test split of inputs.

- Class OLSLinearRegression[5]

- fit(self, X, y):
  Fit linear model.
  Parameters:
  X: {array-like, sparse matrix} of shape (n_samples, n_features)
  Training data.
  y: array-like of shape (n_samples,) or (n_samples, n_targets)
  Target values. Will be cast to X's dtype if necessary.
  Returns
  self: object
  Fitted Estimator.
- get_params(self):
  Get parameters for this estimator.
  Returns:
  Params: dict
  Parameter names mapped to their values.
- predict(self, X):
  Predict using the linear model.
  Parameters:
  X: array-like or sparse matrix, shape (n_samples, n_features)
  Samples.
  Returns:
  C: array, shape (n_samples,)
  Returns predicted values.

- rmse(y, y_hat):
  Mean squared error regression loss.
  Parameters:
  y: array-like of shape (n_samples,) or (n_samples, n_outputs)
  Ground truth (correct) target values.
  y_pred: array-like of shape (n_samples,) or (n_samples, n_outputs)
  Estimated target values.
  Returns:
  Loss: float or ndarray of floats
  A non-negative floating point value (the best value is 0.0), or an array of floating point values, one for each individual target.

- get_features(correlation_threshold)
  get features (correlation_threshold > abs(Correlations))
  Parameters:
  correlation_threshold: float
  Returns:
  Params: dict
  Parameter names

IV. Project details:

a. Task a:

Requirement a: Use all 10 features provided by the problem (2 points)

- Train only once for 10 features on the entire training set (train.csv)
- Show formula for regression model (calculate y by 10 features in X)
- Report 1 results on the test set (test.csv) for the newly trained model

Implementation steps:

- Train only once for 10 features on the entire training set (train.csv)

Use the function fit(X, y) in the OLSLinearRegression class to train 10 sets of features.

- Show formula for regression model

Use get_params() function in OLSLinearRegression class to get coefficients.

```
model = OLSLinearRegression().fit(X_train, y_train)
model.get_params()
```

```
0     0.015101
1     0.090220
2     0.042922
3     0.139289
4    -0.567333
5    -0.000101
6     0.740713
7     0.190936
8    24.505974
9     2.393517
```

Use the predict(X_test) function in OLSLinearRegression class to predict.
Use the rmse(y_test, y_hat) function to calculate RMSE.

```
# Gọi hàm RMSE (tự cài đặt hoặc từ thư viện) trên tập kiểm tra
y_hat = model.predict(X_test)
print('RMSE:', rmse(y_test, y_hat))
```

```
RMSE: 7.064046430584031
```

Regression formula:

$$Life\ expectancy = 0.015101 * X_1 + 0.090220 * X_2 + 0.042922 * X_3 + 0.139289 * X_4 + (-0.567333) * X_5 + (-0.000101) * X_6 + 0.740713 * X_7 + 0.190936 * X_8 + 24.505974 * X_9 + 2.393517 * X_{10}$$

Conclusion:

In this part, 100% of the data used to build the model, so the model obtained is not completely guaranteed in terms of accuracy and stability. Proof is error ~ 7.064046430584031

b. Task b:

Requirement b: Build a model using only 1 feature, find the model that gives the best results (2 points)

- Test on all (10) features the topic offers
- Request use 5-fold Cross Validation method to find the best feature
- Report 10 corresponding results for 10 models from 5-fold Cross Validation (average)

Implementation steps:

- Clone X_train, y_train from data

```
X_train_clone = pd.Series.to_frame(X_train[name]).copy()
y_train_clone = y_train.copy()
```

- Use train_test_split() function to split data into 5 shuffle parts.

```
train_X, test_X, train_y, test_y = train_test_split(X_train_clone, y_train_clone, shuffle = True, test_size=0.2)
```

- Use 5-fold Cross validation method to find the best feature. (RMSE average calculation)

```
RMSE = 0
for i in range(n):
    lr = OLSLinearRegression()
    lr.fit(train_X, train_y)
    y_pred = lr.predict(test_X)
    RMSE += rmse(test_y, y_pred)
list_rmse.append([name, RMSE/n])
```

- With RMSE of Schooling being the smallest. We find that the best feature is **Schooling**

```
The best feature is: Schooling
```

	Feature	RMSE
0	Adult Mortality	46.625184
1	BMI	25.130981
2	Polio	16.510741
3	Diphtheria	17.285067
4	HIV/AIDS	66.141804
5	GDP	61.031703
6	Thinness age 10-19	50.285493
7	Thinness age 5-9	50.948514
8	Income composition of resources	13.711117
9	Schooling	11.871151

- Retrain the model '**Schooling**' with the best feature on the entire training set

```
best_feature_model = OLSLinearRegression().fit(X_train_best, y_train)
best_feature_model.get_params()
```

✓ 0.6s

```
array([5.5573994])
```

- Use the rmse(y_test, y_hat) function to calculate RMSE.

```
y_hat_best = best_feature_model.predict(X_test[min_rmse[0]].values.reshape(-1, 1))
print('RMSE:', rmse(y_test, y_hat_best))
```
✓ 0.8s

```
RMSE: 10.26095039165537
```

Show the formula for the best feature regression model:

$$Life\ expectancy = 5.5573994 * X$$

Conclusion:

After using 5-fold cross validation method we get the average RMSEs from which we conclude that **Schooling** is the best feature.

c. Task c:

Requirement c: Students build their own models, find the model that gives the best results (3 points)

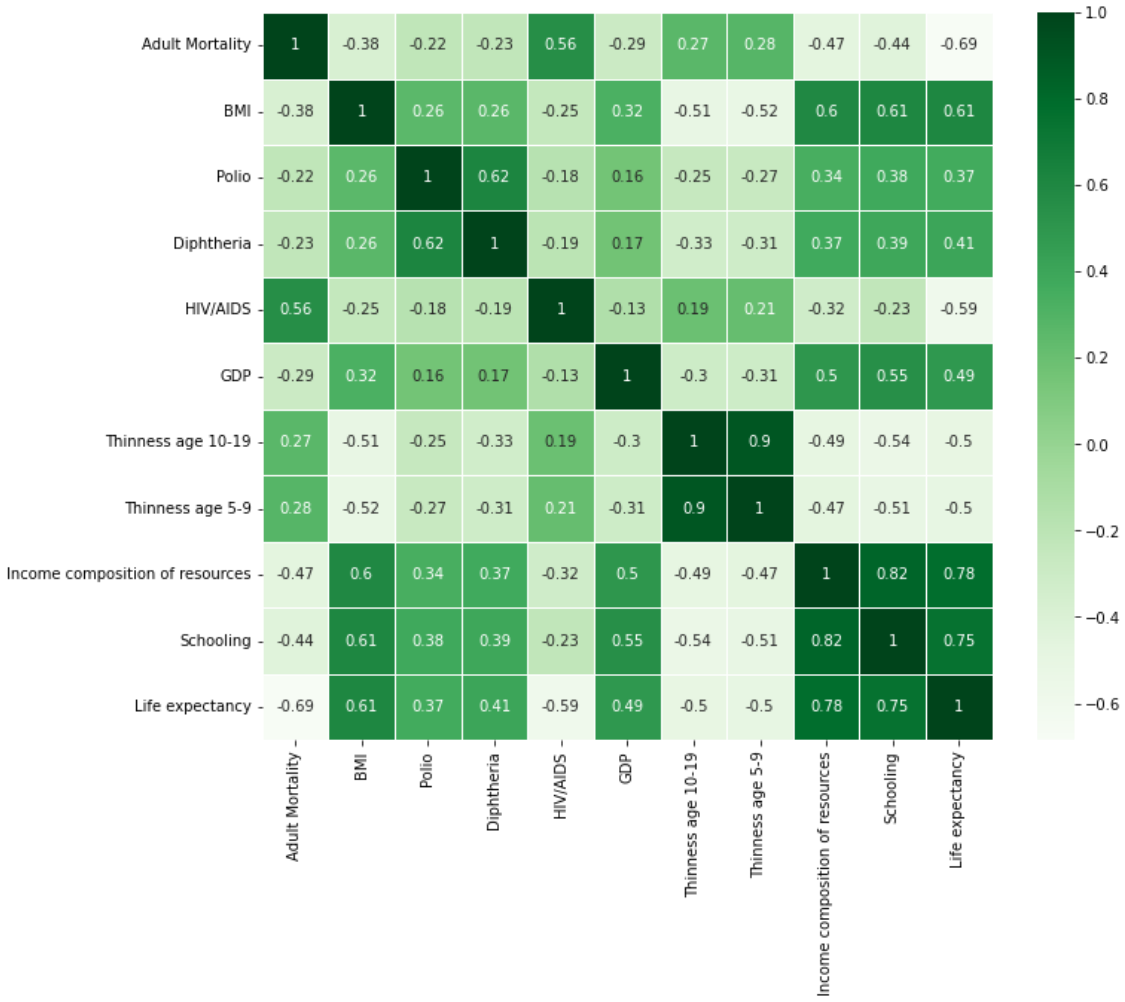
- Build m different models (minimum 3), and at the same time different models at 1a and 1b
  - The model can be a combination of 2 or more features
  - Models can use normalized or transformed features (squared, cubed...)
  - The model can use features created from 2 or more different features (add 2 features, multiply 2 features...)
  - ...
- Request use 5-fold Cross Validation method to find the best model
- Report m corresponding results for m models from 5-fold Cross Validation (average)

Implementation steps:

- Explain the reason for choosing the design for the model:
  - Use Correlations[6] method based on coefficient

Correlation is a method of analysis to investigate the relationship between two fields/characteristics on a set (eg height and weight, price of gold and price of rice,...). Based on the correlation coefficient, we can make a preliminary assessment that which feature affects our model and affects more or less. The correlation coefficient of r with r is equal to 1 and other fields are less than 1. The closer to 1, the more relevant, and the closer to -1, the more irrelevant.

- Correlation chart:



  - Select the features with correlation greater than alpha. With alpha = {0.1, 0.25, 0.5, 0.75}.
  - After testing with 5-fold cross validation method, the RMSE result is not very good. And after receiving a *"whispers of ancestors"* then I standardized the model to square root 2.
- From the alpha of the above design, we have:
  - P1 = ['Schooling', 'Income composition of resources', 'BMI', 'GDP', 'Diphtheria', 'Polio', 'HIV/AIDS', 'Adult Mortality', 'Thinness age 5-9', 'Thinness age 10-19']
  - P2 = ['Schooling', 'Income composition of resources', 'BMI', 'GDP', 'Diphtheria', 'Polio', 'Adult Mortality', 'Thinness age 5-9', 'Thinness age 10-19']
  - P3 = ['Schooling', 'Income composition of resources', 'BMI', 'GDP', 'Thinness age 5-9', 'Thinness age 10-19']
  - P4 = ['Schooling', 'Income composition of resources']
- Process the model by design and put it on the list

```
models1 = []

for i in range(4):
    X_1_train = X_train[p[i]].copy()
    X_1_test = X_test[p[i]].copy()
    models1.append(['Model ' + str(i+1), X_1_train, X_1_test])

    X_sqrt_train = np.sqrt(X_train[p[i]]).copy()
    X_sqrt_test = np.sqrt(X_test[p[i]]).copy()
    models1.append(['Sqrt Model ' + str(i+1), X_sqrt_train, X_sqrt_test])
```

- Use 5-fold Cross validation method to find the best feature. (RMSE average calculation)

	Feature	RMSE
0	Model 1	7.082796
1	Sqrt Model 1	5.354952
2	Model 2	8.678640
3	Sqrt Model 2	5.286792
4	Model 3	8.428662
5	Sqrt Model 3	5.858038
6	Model 4	10.736598
7	Sqrt Model 4	6.902832

- We find that the best feature is **Sqrt model 1** with coefficients:

```
0    14.091507
1     9.579369
2     0.552901
3     0.001705
4     0.801555
5     0.186563
6    -3.208658
7    -0.009331
8    -0.009752
9     1.818148
```

- Use the rmse(y_test, y_hat) function to calculate RMSE.

```
y_pred = lr_best_model.predict(X_test_best_model)
print('RMSE:', rmse(y_test, y_pred))
```

✓ 0.6s

```
RMSE: 4.7680380241569695
```

Show the formula for the best feature regression model:

$$Life\ expectancy = 14.091507 * \sqrt{X_1} + 9.579369 * \sqrt{X_2} + 0.552901 * \sqrt{X_3} + 0.001705 * \sqrt{X_4} + 0.801555 * \sqrt{X_5} + 0.186563 * \sqrt{X_6} + (-3.208658) * \sqrt{X_7} + (-0.009331) * \sqrt{X_8} + (-0.009752) * \sqrt{X_9} + 1.818148 * \sqrt{X_{10}}$$

Conclusion:

Compared with the initial error in question a is 7.064046430584031, then the error in the individual model obtained is 4.7680380241569695 proving that the individual model just built is closer to the data.

The reason is because the transformed data has reduced dispersion, changing the subdata model has changed the bias by subcomponents leading to less biased large model.

V. Reference:

[1] ML | Linear Regression - GeeksforGeeks

[2] Cross-Validation: K-Fold vs. Leave-One-Out | Baeldung on Computer Science

[3] Life Expectancy (WHO) | Kaggle

[4] sklearn.model_selection.train_test_split — scikit-learn 1.1.1 documentation

[5] **Lab 4 by Ms. Uyen**

[6] Correlation - Statistical Techniques, Rating Scales, Correlation Coefficients, and More - Creative Research Systems (surveysystem.com)