Using random forest regression to determine what factors have influence on life expectancy

Team member: Jiahua Zhang , Bin Xu, Di Kang, Yiwei Zhang

Abstract

By using the Life Expectancy dataset provided by Kaggle, we wanted to see what the world needed to improve on the Life Expectancy. We used a random forest regressor to determine the effects of scaling certain features and their correlation to Life Expectancy.

1 Introduction

The dataset contains different countries with different life expectancy. Some variables have a very weak relationship with life expectancy or even cannot affect life expectancy. The life expectancy Report is a survey of the state of global life expectancy, reporting these 18 factors: Country, Year, Status, Life expectancy, Adult Mortality, infant deaths, Alcohol, percentage expenditure, Hepatitis B, Measles, BMI, under-five deaths, Polio, Total expenditure, Diphtheria, HIV/AIDS, GDP, Population, thinness 1-19 years, thinness 5-9 years, Income composition of resources, Schooling. In the project, we need to discover which factor has more influence on life expectancy.

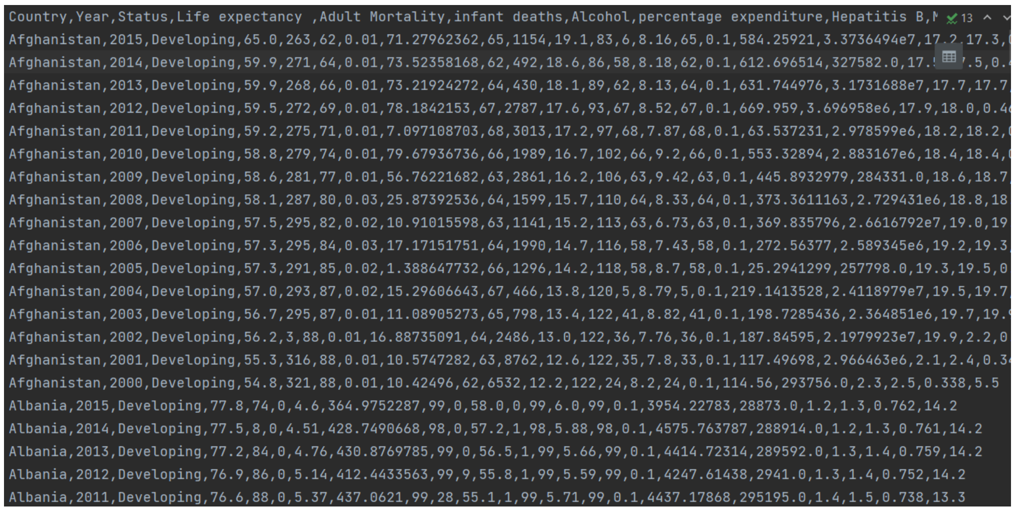
2 Background

Life expectancy is a worldwide topic. It is meaningful to discover which factor in real life will impact our life expectancy. Data we get is from people in over 150 countries. Features in the dataset will be filtered because we want to find the most important feature works on the life expectancy.

3 Dataset preparation

3.1 Data overview

There are currently 150 countries and regions included within. The first element records the name of the country, the second element indicates the year of the data. In our project, we will drop the status column from the data. Our Y value is "Life Expectancy".

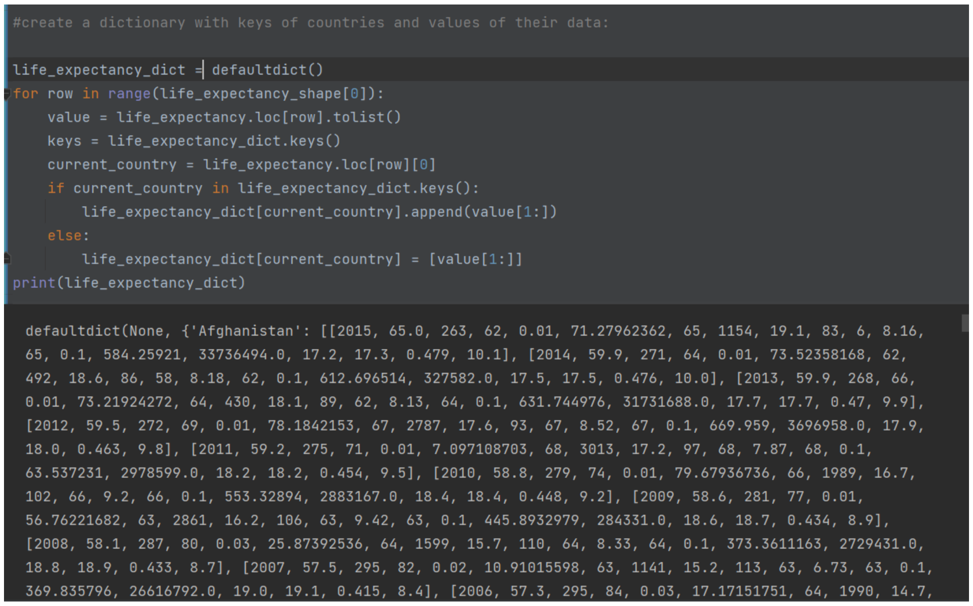


```
Country,Year,Status,Life expectancy ,Adult Mortality,infant deaths,Alcohol,percentage expenditure,Hepatitis B,M
Afghanistan,2015,Developing,65.0,263,62,0.01,71.27962362,65,1154,19.1,83,6,8.16,65,0.1,584.25921,3.3736494e7,17.2,17.3,
Afghanistan,2014,Developing,59.9,271,64,0.01,73.52358168,62,492,18.6,86,58,8.18,62,0.1,612.696514,327582.0,17.5[illegible].5,0.
Afghanistan,2013,Developing,59.9,268,66,0.01,73.21924272,64,430,18.1,89,62,8.13,64,0.1,631.744976,3.1731688e7,17.7,17.7
Afghanistan,2012,Developing,59.5,272,69,0.01,78.1842153,67,2787,17.6,93,67,8.52,67,0.1,669.959,3.696958e6,17.9,18.0,0.4
Afghanistan,2011,Developing,59.2,275,71,0.01,7.097108703,68,3013,17.2,97,68,7.87,68,0.1,63.537231,2.978599e6,18.2,18.2,
Afghanistan,2010,Developing,58.8,279,74,0.01,79.67936736,66,1989,16.7,102,66,9.2,66,0.1,553.32894,2.883167e6,18.4,18.4,
Afghanistan,2009,Developing,58.6,281,77,0.01,56.76221682,63,2861,16.2,106,63,9.42,63,0.1,445.8932979,284331.0,18.6,18.7
Afghanistan,2008,Developing,58.1,287,80,0.03,25.87392536,64,1599,15.7,110,64,8.33,64,0.1,373.3611163,2.729431e6,18.8,18
Afghanistan,2007,Developing,57.5,295,82,0.02,10.91015598,63,1141,15.2,113,63,6.73,63,0.1,369.835796,2.6616792e7,19.0,19
Afghanistan,2006,Developing,57.3,295,84,0.03,17.17151751,64,1990,14.7,116,58,7.43,58,0.1,272.56377,2.589345e6,19.2,19.3
Afghanistan,2005,Developing,57.3,291,85,0.02,1.388647732,66,1296,14.2,118,58,8.7,58,0.1,25.2941299,257798.0,19.3,19.5,0
Afghanistan,2004,Developing,57.0,293,87,0.02,15.29606643,67,466,13.8,120,5,8.79,5,0.1,219.1413528,2.4118979e7,19.5,19.7
Afghanistan,2003,Developing,56.7,295,87,0.01,11.08905273,65,798,13.4,122,41,8.82,41,0.1,198.7285436,2.364851e6,19.7,19.
Afghanistan,2002,Developing,56.2,3,88,0.01,16.88735091,64,2486,13.0,122,36,7.76,36,0.1,187.84595,2.1979923e7,19.9,2.2,0
Afghanistan,2001,Developing,55.3,316,88,0.01,10.5747282,63,8762,12.6,122,35,7.8,33,0.1,117.49698,2.966463e6,2.1,2.4,0.3
Afghanistan,2000,Developing,54.8,321,88,0.01,10.42496,62,6532,12.2,122,24,8.2,24,0.1,114.56,293756.0,2.3,2.5,0.338,5.5
Albania,2015,Developing,77.8,74,0,4.6,364.9752287,99,0,58.0,0,99,6.0,99,0.1,3954.22783,28873.0,1.2,1.3,0.762,14.2
Albania,2014,Developing,77.5,8,0,4.51,428.7490668,98,0,57.2,1,98,5.88,98,0.1,4575.763787,288914.0,1.2,1.3,0.761,14.2
Albania,2013,Developing,77.2,84,0,4.76,430.8769785,99,0,56.5,1,99,5.66,99,0.1,4414.72314,289592.0,1.3,1.4,0.759,14.2
Albania,2012,Developing,76.9,86,0,5.14,412.4433563,99,9,55.8,1,99,5.59,99,0.1,4247.61438,2941.0,1.3,1.4,0.752,14.2
Albania,2011,Developing,76.6,88,0,5.37,437.0621,99,28,55.1,1,99,5.71,99,0.1,4437.17868,295195.0,1.4,1.5,0.738,13.3
```

figure 1

3.2 Data processing

We use dict in the python to store the data from csv, which is shown at figure 2. Later we will find the exception through dict.



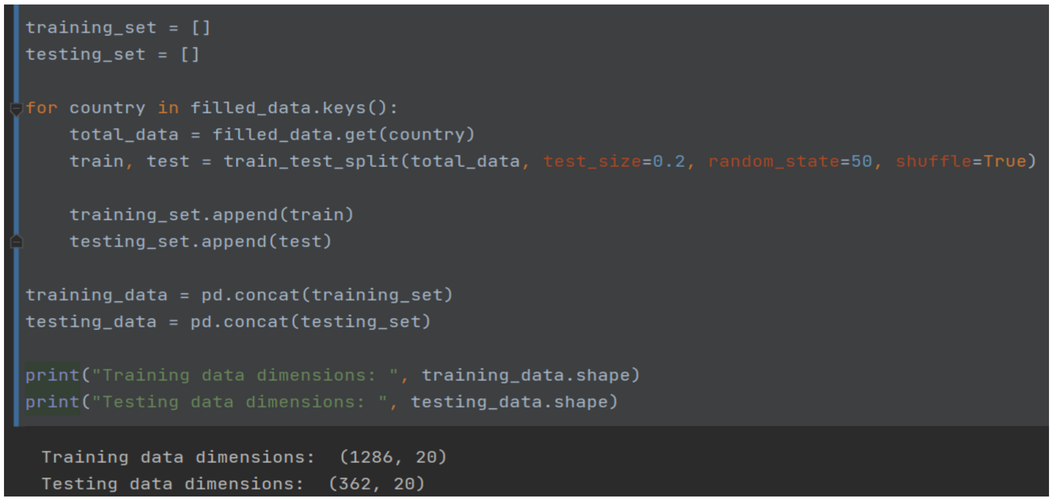
```
#create a dictionary with keys of countries and values of their data:

life_expectancy_dict = defaultdict()
for row in range(life_expectancy_shape[0]):
    value = life_expectancy.loc[row].tolist()
    keys = life_expectancy_dict.keys()
    current_country = life_expectancy.loc[row][0]
    if current_country in life_expectancy_dict.keys():
        life_expectancy_dict[current_country].append(value[1:])
    else:
        life_expectancy_dict[current_country] = [value[1:]]
print(life_expectancy_dict)
```

```
defaultdict(None, {'Afghanistan': [[2015, 65.0, 263, 62, 0.01, 71.27962362, 65, 1154, 19.1, 83, 6, 8.16,
65, 0.1, 584.25921, 33736494.0, 17.2, 17.3, 0.479, 10.1], [2014, 59.9, 271, 64, 0.01, 73.52358168, 62,
492, 18.6, 86, 58, 8.18, 62, 0.1, 612.696514, 327582.0, 17.5, 17.5, 0.476, 10.0], [2013, 59.9, 268, 66,
0.01, 73.21924272, 64, 430, 18.1, 89, 62, 8.13, 64, 0.1, 631.744976, 31731688.0, 17.7, 17.7, 0.47, 9.9],
[2012, 59.5, 272, 69, 0.01, 78.1842153, 67, 2787, 17.6, 93, 67, 8.52, 67, 0.1, 669.959, 3696958.0, 17.9,
18.0, 0.463, 9.8], [2011, 59.2, 275, 71, 0.01, 7.097108703, 68, 3013, 17.2, 97, 68, 7.87, 68, 0.1,
63.537231, 2978599.0, 18.2, 18.2, 0.454, 9.5], [2010, 58.8, 279, 74, 0.01, 79.67936736, 66, 1989, 16.7,
102, 66, 9.2, 66, 0.1, 553.32894, 2883167.0, 18.4, 18.4, 0.448, 9.2], [2009, 58.6, 281, 77, 0.01,
56.76221682, 63, 2861, 16.2, 106, 63, 9.42, 63, 0.1, 445.8932979, 284331.0, 18.6, 18.7, 0.434, 8.9],
[2008, 58.1, 287, 80, 0.03, 25.87392536, 64, 1599, 15.7, 110, 64, 8.33, 64, 0.1, 373.3611163, 2729431.0,
18.8, 18.9, 0.433, 8.7], [2007, 57.5, 295, 82, 0.02, 10.91015598, 63, 1141, 15.2, 113, 63, 6.73, 63, 0.1,
369.835796, 26616792.0, 19.0, 19.1, 0.415, 8.4], [2006, 57.3, 295, 84, 0.03, 17.17151751, 64, 1990, 14.7,
```

figure 2

3.3 Generate training and testing data and delete the exception from the dataset

We extract 80% of data as our training data and the remaining 20% as our testing data.

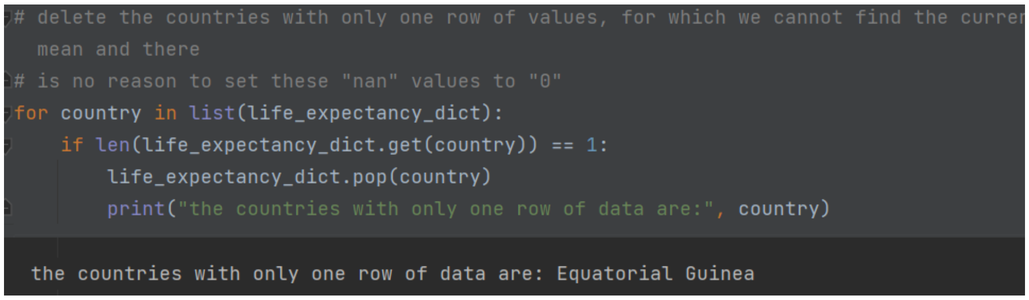


```
training_set = []
testing_set = []

for country in filled_data.keys():
    total_data = filled_data.get(country)
    train, test = train_test_split(total_data, test_size=0.2, random_state=50, shuffle=True)

    training_set.append(train)
    testing_set.append(test)

training_data = pd.concat(training_set)
testing_data = pd.concat(testing_set)

print("Training data dimensions: ", training_data.shape)
print("Testing data dimensions: ", testing_data.shape)
```

```
Training data dimensions:  (1286, 20)
Testing data dimensions:  (362, 20)
```

figure 3

We delete the exception from the data set. After we go through the dataset, we find there are two countries with 1 row. We drop it.

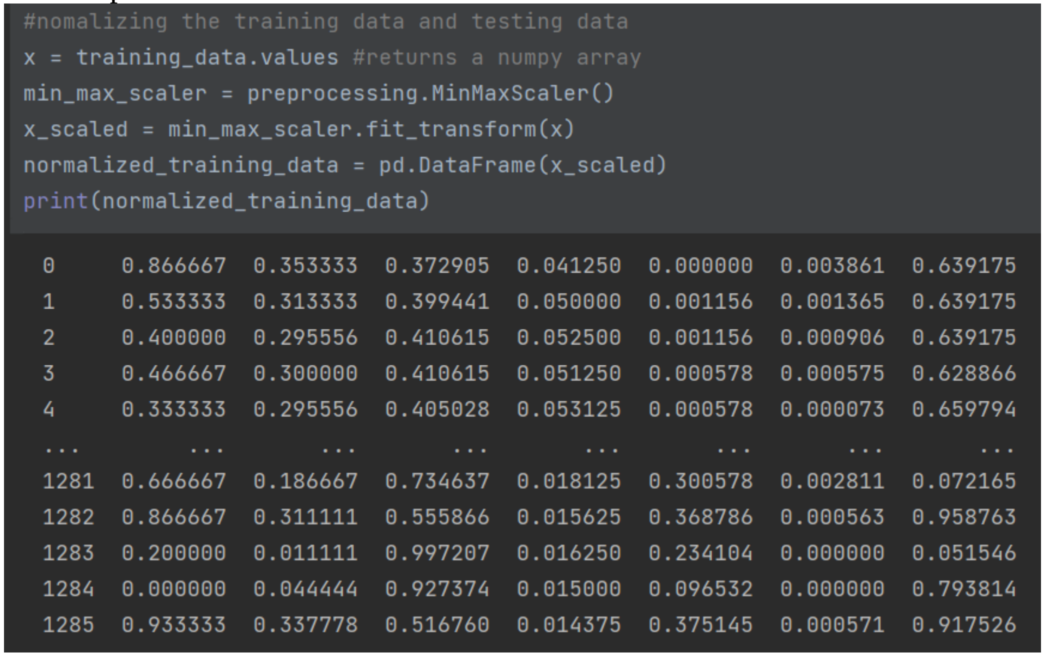


```
# delete the countries with only one row of values, for which we cannot find the curren
  mean and there
# is no reason to set these "nan" values to "0"
for country in list(life_expectancy_dict):
    if len(life_expectancy_dict.get(country)) == 1:
        life_expectancy_dict.pop(country)
        print("the countries with only one row of data are:", country)

 the countries with only one row of data are: Equatorial Guinea
```

figure 4

3.4 Normalizing the data

Using minmaxscaler to normalize the testing data and training data, then we can convert the data to a pickle file.



```
#nomalizing the training data and testing data
x = training_data.values #returns a numpy array
min_max_scaler = preprocessing.MinMaxScaler()
x_scaled = min_max_scaler.fit_transform(x)
normalized_training_data = pd.DataFrame(x_scaled)
print(normalized_training_data)

0     0.866667  0.353333  0.372905  0.041250  0.000000  0.003861  0.639175
1     0.533333  0.313333  0.399441  0.050000  0.001156  0.001365  0.639175
2     0.400000  0.295556  0.410615  0.052500  0.001156  0.000906  0.639175
3     0.466667  0.300000  0.410615  0.051250  0.000578  0.000575  0.628866
4     0.333333  0.295556  0.405028  0.053125  0.000578  0.000073  0.659794
...        ...       ...       ...       ...       ...       ...       ...
1281  0.666667  0.186667  0.734637  0.018125  0.300578  0.002811  0.072165
1282  0.866667  0.311111  0.555866  0.015625  0.368786  0.000563  0.958763
1283  0.200000  0.011111  0.997207  0.016250  0.234104  0.000000  0.051546
1284  0.000000  0.044444  0.927374  0.015000  0.096532  0.000000  0.793814
1285  0.933333  0.337778  0.516760  0.014375  0.375145  0.000571  0.917526
```

figure 5

4 Experimentation

4.1 overview

We plan to find the optimal value from the decision tree. Using a random forest regressor to decide which factor will have a bigger influence on life expectancy.

4.2 Algorithm decision

When we figure out which algorithms will be used in our project. We rule out some other different algorithms. For example, linear regression, SVM and so on. Based on our dataset, we choose to use random regression to predict the factor which has a big influence on our life expectancy. From the dataset, the highest level of life expectancy does not show that a high life expectancy is equally as high compared to countries with less. After we rule out other possible algorithms, the data leads us to the random forest.

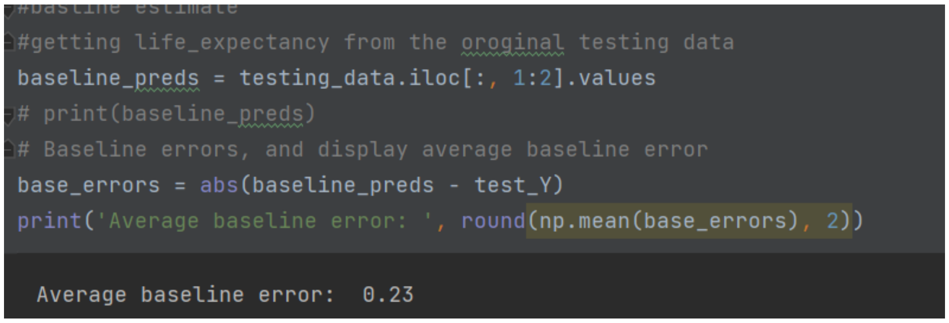
4.3 Implementation

We set the coverage for the algorithm, add the trees amount and increase the possible nodes per tree. We observe the incremental changes on the tree and improvement decreases. Through the binary tree, we need to check the nodes with different features, then we can figure out the progress of prediction for each feature which impacts the life expectancy.

5 Results

5.1 Set Baseline

Before we can make and evaluate predictions, we need to set up the baseline. In this way, we can measure if our model can beat after training.



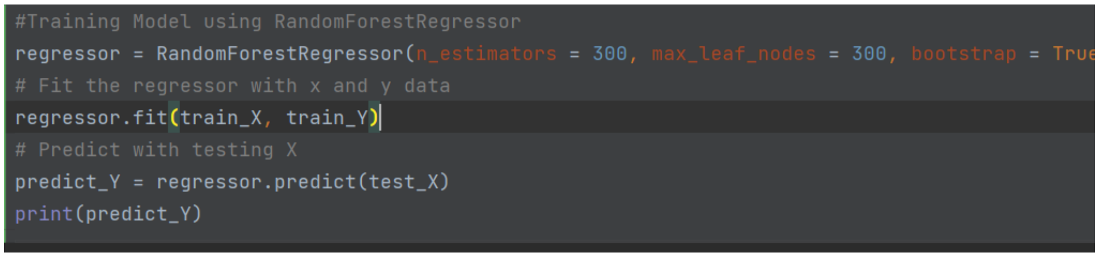
```
#basline estimate
#getting life_expectancy from the oroginal testing data
baseline_preds = testing_data.iloc[:, 1:2].values
# print(baseline_preds)
# Baseline errors, and display average baseline error
base_errors = abs(baseline_preds - test_Y)
print('Average baseline error: ', round(np.mean(base_errors), 2))
```

```
 Average baseline error:  0.23
```

figure 6

5.2 Train model

Using scikit learn, instantiate the model. We import the random first regression model, and use training data.

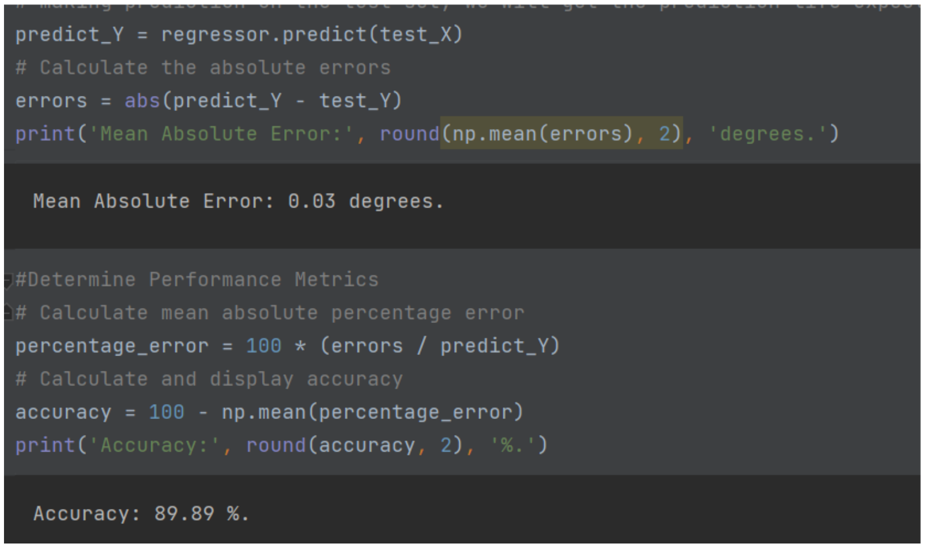


```
#Training Model using RandomForestRegressor
regressor = RandomForestRegressor(n_estimators = 300, max_leaf_nodes = 300, bootstrap = True
# Fit the regressor with x and y data
regressor.fit(train_X, train_Y)
# Predict with testing X
predict_Y = regressor.predict(test_X)
print(predict_Y)
```

figure 7

5.3 make prediction on the test set and determine the performance

We have got the relationships between the feature and target. Then we need to figure out if our result is good enough. In order to make predictions, we need to use the absolute error. We compare the predictions to the known answer. We want to know how far the prediction value is from the actual value.



```
predict_Y = regressor.predict(test_X)
# Calculate the absolute errors
errors = abs(predict_Y - test_Y)
print('Mean Absolute Error:', round(np.mean(errors), 2), 'degrees.')
```

```
Mean Absolute Error: 0.03 degrees.
```

```
#Determine Performance Metrics
# Calculate mean absolute percentage error
percentage_error = 100 * (errors / predict_Y)
# Calculate and display accuracy
accuracy = 100 - np.mean(percentage_error)
print('Accuracy:', round(accuracy, 2), '%.')
```

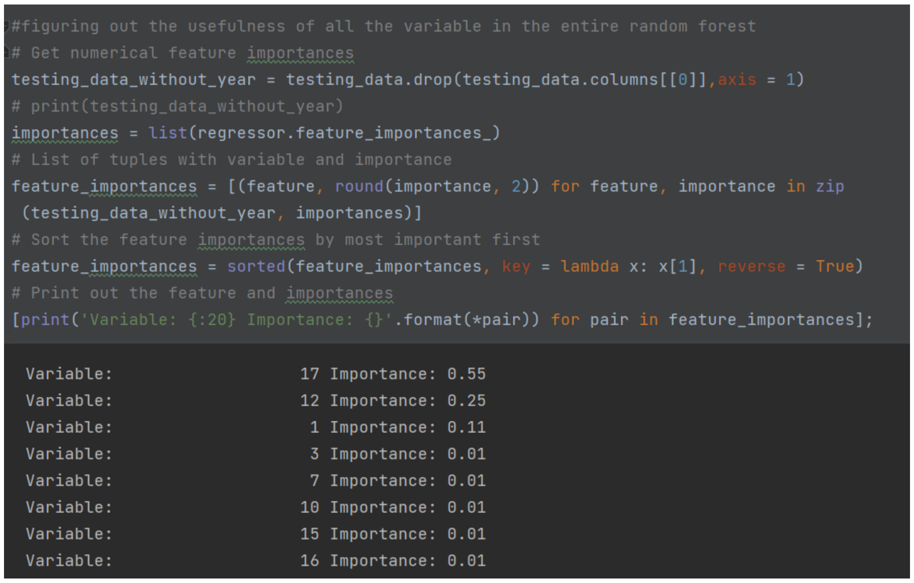
```
Accuracy: 89.89 %.
```

figure 8

From the above figure, we get the 0.03 degree, which is much better than the baseline error. Also, we get 89.89% accuracy. We will try to improve this accuracy later.

5.3 variable importance

In order to quantify the usefulness of all the variables from the testing data, we want to find the relative importances of variables. The code we use the zip, osring and argument unpacking.

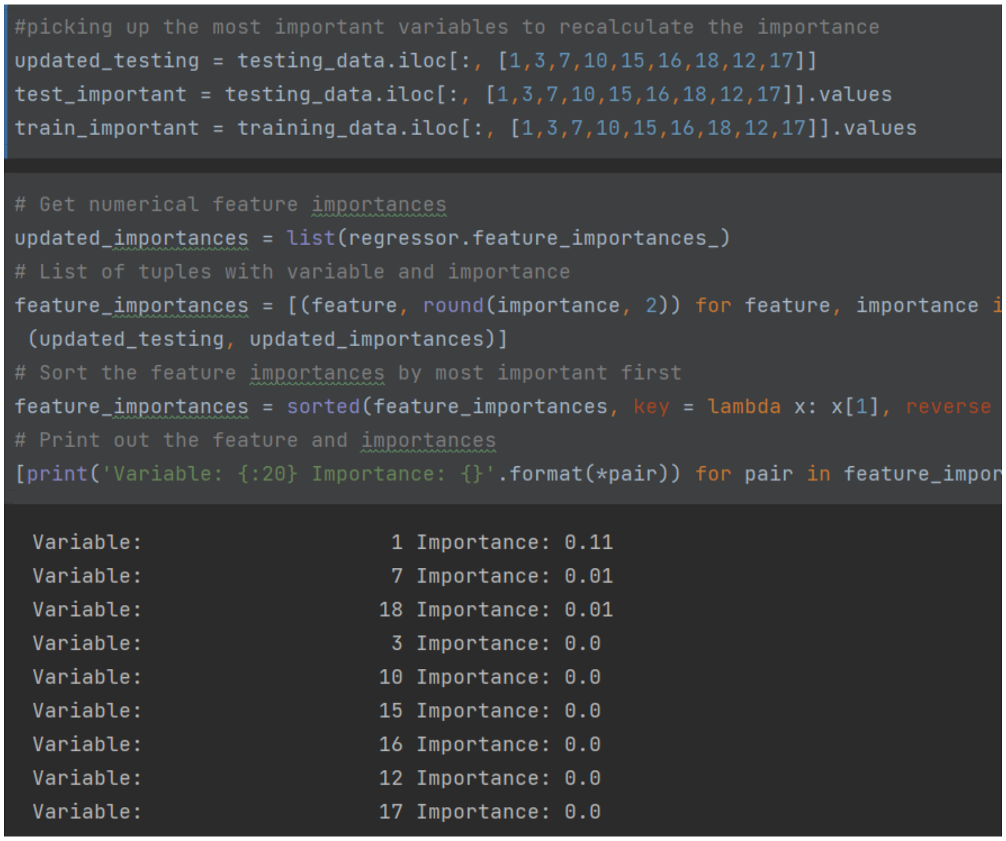


```
#figuring out the usefulness of all the variable in the entire random forest
# Get numerical feature importances
testing_data_without_year = testing_data.drop(testing_data.columns[[0]],axis = 1)
# print(testing_data_without_year)
importances = list(regressor.feature_importances_)
# List of tuples with variable and importance
feature_importances = [(feature, round(importance, 2)) for feature, importance in zip
 (testing_data_without_year, importances)]
# Sort the feature importances by most important first
feature_importances = sorted(feature_importances, key = lambda x: x[1], reverse = True)
# Print out the feature and importances
[print('Variable: {:20} Importance: {}'.format(*pair)) for pair in feature_importances];
```

```
 Variable:                   17 Importance: 0.55
 Variable:                   12 Importance: 0.25
 Variable:                    1 Importance: 0.11
 Variable:                    3 Importance: 0.01
 Variable:                    7 Importance: 0.01
 Variable:                   10 Importance: 0.01
 Variable:                   15 Importance: 0.01
 Variable:                   16 Importance: 0.01
```

figure 9

We can get the index 17, 12 , 1… are most relative variables in our testing data. Then we calculate importance twice, we pick the most important column from testing data. We calculate the importance again.



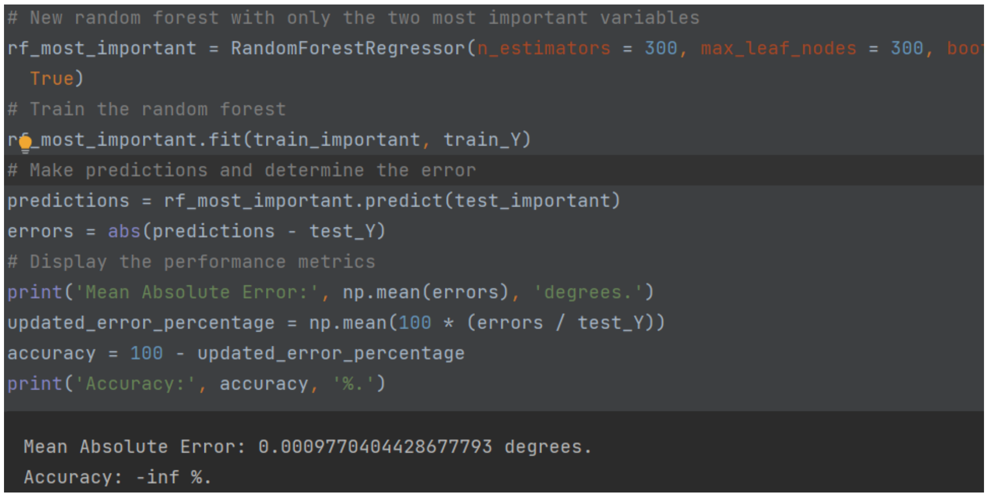
```
#picking up the most important variables to recalculate the importance
updated_testing = testing_data.iloc[:, [1,3,7,10,15,16,18,12,17]]
test_important = testing_data.iloc[:, [1,3,7,10,15,16,18,12,17]].values
train_important = training_data.iloc[:, [1,3,7,10,15,16,18,12,17]].values
```

```
# Get numerical feature importances
updated_importances = list(regressor.feature_importances_)
# List of tuples with variable and importance
feature_importances = [(feature, round(importance, 2)) for feature, importance i
 (updated_testing, updated_importances)]
# Sort the feature importances by most important first
feature_importances = sorted(feature_importances, key = lambda x: x[1], reverse
# Print out the feature and importances
[print('Variable: {:20} Importance: {}'.format(*pair)) for pair in feature_impor
```

```
 Variable:                    1 Importance: 0.11
 Variable:                    7 Importance: 0.01
 Variable:                   18 Importance: 0.01
 Variable:                    3 Importance: 0.0
 Variable:                   10 Importance: 0.0
 Variable:                   15 Importance: 0.0
 Variable:                   16 Importance: 0.0
 Variable:                   12 Importance: 0.0
 Variable:                   17 Importance: 0.0
```

figure 10

After we get the updated importance, we can figure which index will be more important. We also test the mean absolute and accuracy again. We hope after training the most relative variable, the error and accuracy can further improve.



```
# New random forest with only the two most important variables
rf_most_important = RandomForestRegressor(n_estimators = 300, max_leaf_nodes = 300, boo
  True)
# Train the random forest
rf_most_important.fit(train_important, train_Y)
# Make predictions and determine the error
predictions = rf_most_important.predict(test_important)
errors = abs(predictions - test_Y)
# Display the performance metrics
print('Mean Absolute Error:', np.mean(errors), 'degrees.')
updated_error_percentage = np.mean(100 * (errors / test_Y))
accuracy = 100 - updated_error_percentage
print('Accuracy:', accuracy, '%.')

 Mean Absolute Error: 0.0009770404428677793 degrees.
 Accuracy: -inf %.
```

figure 11

From the above data we get, the error is improved. The accuracy is infinitely closed to the 100%

6. Math Background

We need to measure the performance of the model, In order to calculate the loss, using below formula:

$$\sum_{n=1}^{n} \frac{abs(predict_n - truth_n)}{truth_n}$$

We get the predicted value from regress.predict(), then calculate the percentage to the loss. Adding up all dimension test data through the formula above will help me to get a loss percentage later. Another method we used to test for errors is to calculate the mean square error of the random forest regressor model. For calculate the MSE, we use following formula:

$$\text{NRMSE} = \frac{\sqrt{\frac{1}{N}\sum_{n=1}^{N}\left(\hat{c}_n - c_n\right)^2}}{\sigma\left(c\right)},$$

6 Limitations

For the dataset, we drop the status of the country and year column. If we want to use status and year, we may consider other methods to predict life expectancy. More comprehensive feature works for the real life situation.

7 Additional works

In future implementations of the model, we can remove the variables that have no importance. The performance will not suffer. If we use a differing model, like a support vector machine, we can use the random forest feature importance as a feature selection method.

https://github.com/xu842251462/cs542FinalProject