Main.py

```
import numpy as np
from sklearn.metrics import mean_squared_error
import warnings
warnings.filterwarnings("ignore")
from sklearn.metrics import r2_score
np.set_printoptions(threshold=np.inf)
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
import pandas as pd
import seaborn as sns
color = sns.color_palette()
sns.set_style('darkgrid')
from sklearn.ensemble import RandomForestRegressor

def main():
    pd.set_option('display.max_columns', None)
    from sklearn.neighbors import KNeighborsClassifier
    data = pd.read_csv('Life Expectancy Data.csv')
    # print(data.columns)
    data.rename(columns={'Life expectancy ': "Life_expectancy"}, inplace=True)
    data.rename(columns={'Adult Mortality': "Adult_Mortality"}, inplace=True)
    data.rename(columns={'infant deaths': "infant_deaths"}, inplace=True)
    data.rename(columns={'percentage expenditure': 'percentage_expenditure'},
inplace=True)
    data.rename(columns={'Hepatitis B': "Hepatitis_B"}, inplace=True)
    data.rename(columns={' BMI ': "BMI"}, inplace=True)
    data.rename(columns={'under-five deaths ': "under-five_deaths"}, inplace=True)
    data.rename(columns={'Total expenditure': "Total_expenditure"}, inplace=True)
    data.rename(columns={' HIV/AIDS': "HIV/AIDS"}, inplace=True)
    data.rename(columns={' thinness  1-19 years': "thinness_1-19_years"},
inplace=True)
    data.rename(columns={' thinness 5-9 years': "thinness_5-9_years"},
inplace=True)
    data.rename(columns={'Income composition of resources':
"Income_composition_of_resources"}, inplace=True)
    data.rename(columns={'HIV/AIDS': "HIV_AIDS"}, inplace=True)
    data.rename(columns={'Measles ': "Measles"}, inplace=True)
```

```
    data.rename(columns={'Diphtheria ': "Diphtheria"}, inplace=True)

    # delet the data with null life expectancy value
    data['Life_expectancy'] = data['Life_expectancy'].fillna(999)
    drop_index = data[(data.Life_expectancy == 999)].index.tolist()
    data = data.drop(drop_index)

    # make life_expectancy as our output
    labels = data.loc[:, ['Life_expectancy']]
    # deal with categorical data "status" since it contains numerical quality
    data['Status'] = data['Status'].str.replace('Developed', '2', case=False)
    data['Status'] = data['Status'].str.replace('Developing', '1', case=False)



    # Separate the data to train, val and test data
    x, x_test, y, y_test = train_test_split(data, labels, test_size=0.2, train_size=0.8,
random_state=1)
    x_train, x_val, y_train, y_val = train_test_split(x, y, test_size=0.2, train_size=0.8,
random_state=50)

    ####fill the missing data with mean value
    x = x.fillna(x.mean())
    x_test = x_test.fillna(x.mean())

    x_num = x.loc[:, ['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']]
    x_test_num = x_test.loc[:, ['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']]
    # Since the highest correlation between country and life expectancy is 0.17, we
decide not to use the feature "country"

    # standardize the data
    standar_all = preprocessing.StandardScaler().fit(x_num)
    x = standar_all.transform(x_num)
    x_test = standar_all.transform(x_test_num)
    x1 = pd.DataFrame(data=x, columns=x_num.columns)
    x_test1 = pd.DataFrame(data=x_test, columns=x_num.columns)
    corrmat = x1.corr()

    cols = abs(corrmat).nlargest(22, 'Life_expectancy')['Life_expectancy'].index
    related_col =
cols.drop(['Life_expectancy']).drop(['Status']).drop(['Hepatitis_B']).drop(['infant_deat
hs']).drop(

['GDP']).drop(['Measles']).drop(['Population']).drop(['percentage_expenditure']).drop
(['Diphtheria'])
    x = x1[related_col]
    x_test = x_test1[related_col]

    avg_ytest = np.mean(y_test)
    one_array = np.ones([len(y_test), 1])
    mean_arr = avg_ytest['Life_expectancy'] * one_array
    baseline_mse = mean_squared_error(y_test, mean_arr)
    baseline_err = 1 - r2_score(y_test, mean_arr)
    print('#####################Baseline#######################')
    print('The baseline for the test data:')
    print('MSE = ', baseline_mse)
    print('Error Rate=', baseline_err)
    print('###################Final model: Random Forest
Regression########################')
    error_x = []
    error_test = []
```

```
    mse_x = []
    mse_test = []
    oob_error = []
    for j in range(10):
        pre_train, pre_X_train_pick, pre_y_train, pre_y_train_pick =
train_test_split(x, y, test_size=1 / 3)
        RF = RandomForestRegressor(n_estimators=38, bootstrap=True,
random_state=0, oob_score=True)
        RF.fit(pre_X_train_pick, pre_y_train_pick.values.ravel())
        predict_x = RF.predict(x)
        predict_test = RF.predict(x_test)
        acc_x = RF.score(x, y)
        acc_test = RF.score(x_test, y_test)
        error_x = np.append(error_x, 1 - acc_x)
        error_test = np.append(error_test, 1 - acc_test)
        oob_error = np.append(oob_error, 1 - RF.oob_score_)
        mse_x = np.append(mse_x, mean_squared_error(predict_x, y))
        mse_test = np.append(mse_test, mean_squared_error(predict_test,
y_test))
    mean_oob_err = np.mean(oob_error)
    meanerror_x = np.mean(error_x)
    meanerror_test = np.mean(error_test)
    mean_mse_x = np.mean(mse_x)
    mean_mse_test = np.mean(mse_test)
    var_mse_test = np.var(mse_test)
    var_err_test = np.var(error_test)
    print('In our final model (38 trees):')
    print('In the whole training data')
    print('Mean MSE =', mean_mse_x)
    print('Mean Error Rate =', meanerror_x)
    print('In the test data')
    print('Mean MSE = %.3f with variance = %.3f ' % (mean_mse_test,
var_mse_test))
    print('Mean Error Rate = %.5f with variance = %.5f ' % (meanerror_test,
var_err_test))
    print('Out Of Sample Error = ', mean_oob_err)
if __name__ == "__main__":
    main()
```

Preprocessing_and_training.py

```
from sklearn.svm import SVC
import numpy as np
from sklearn.metrics import r2_score
from sklearn import linear_model
from sklearn.metrics import mean_squared_error
from yellowbrick.regressor import AlphaSelection
import warnings

warnings.filterwarnings("ignore")
from sklearn.metrics import r2_score

np.set_printoptions(threshold=np.inf)
from sklearn.model_selection import train_test_split

from sklearn.model_selection import cross_val_score, StratifiedKFold, KFold
import matplotlib.pyplot as plt

from sklearn import preprocessing
import pandas as pd
import seaborn as sns

color = sns.color_palette()
sns.set_style('darkgrid')
from sklearn.ensemble import RandomForestRegressor



def outlier_remove_traindata(x_train, Q1, Q3, IQR):
    # try to remove the outliers
    # print('len of x train=',len(x_train))
    ColumnName = x_train.columns
    outlier_detect_column = ColumnName[3:len(ColumnName) - 1]
    x_train_out = x_train[~(
                    (x_train[outlier_detect_column] < (Q1[outlier_detect_column] -
1.5 * IQR[outlier_detect_column])) | (
                        x_train[outlier_detect_column] > (
                            Q3[outlier_detect_column] + 1.5 *
```

```
IQR[outlier_detect_column]))).any(axis=1)]

    # ----------deal with the outlier in each features----------
    x_train.loc[x_train.Adult_Mortality > (Q3.Adult_Mortality + 1.5 *
IQR.Adult_Mortality), 'Adult_Mortality'] = max(
        x_train_out.Adult_Mortality)
    x_train.loc[x_train.Adult_Mortality < (Q1.Adult_Mortality - 1.5 *
IQR.Adult_Mortality), 'Adult_Mortality'] = min(
        x_train_out.Adult_Mortality)

    x_train.loc[x_train.infant_deaths > (Q3.infant_deaths + 1.5 *
IQR.infant_deaths), 'infant_deaths'] = max(
        x_train_out.infant_deaths)
    x_train.loc[x_train.infant_deaths < (Q1.infant_deaths - 1.5 * IQR.infant_deaths),
'infant_deaths'] = min(
        x_train_out.infant_deaths)

    x_train.loc[x_train.Alcohol > (Q3.Alcohol + 1.5 * IQR.Alcohol), 'Alcohol'] =
max(x_train_out.Alcohol)
    x_train.loc[x_train.Alcohol < (Q1.Alcohol - 1.5 * IQR.Alcohol), 'Alcohol'] =
min(x_train_out.Alcohol)

    x_train.loc[x_train.percentage_expenditure > (
            Q3.percentage_expenditure + 1.5 *
IQR.percentage_expenditure), 'percentage_expenditure '] = max(
        x_train_out.percentage_expenditure)
    x_train.loc[x_train.percentage_expenditure < (
            Q1.percentage_expenditure - 1.5 *
IQR.percentage_expenditure), 'percentage_expenditure '] = min(
        x_train_out.percentage_expenditure)

    x_train.loc[x_train.Hepatitis_B > (Q3.Hepatitis_B + 1.5 * IQR.Hepatitis_B),
'Hepatitis_B'] = max(
        x_train_out.Hepatitis_B)
    x_train.loc[x_train.Hepatitis_B < (Q1.Hepatitis_B - 1.5 * IQR.Hepatitis_B),
'Hepatitis_B'] = min(
        x_train_out.Hepatitis_B)
```

```
    x_train.loc[x_train.Measles > (Q3.Measles + 1.5 * IQR.Measles), 'Measles'] =
max(x_train_out.Measles)
    x_train.loc[x_train.Measles < (Q1.Measles - 1.5 * IQR.Measles), 'Measles'] =
min(x_train_out.Measles)

    x_train.loc[x_train.BMI > (Q3.BMI + 1.5 * IQR.BMI), 'BMI'] =
max(x_train_out.BMI)
    x_train.loc[x_train.BMI < (Q1.BMI - 1.5 * IQR.BMI), 'BMI'] =
min(x_train_out.BMI)

    x_train.loc[x_train['under-five_deaths'] > (
                    Q3['under-five_deaths'] + 1.5 * IQR['under-five_deaths']), 'under-
five_deaths'] = max(
        x_train_out['under-five_deaths'])
    x_train.loc[x_train['under-five_deaths'] < (
                    Q1['under-five_deaths'] - 1.5 * IQR['under-five_deaths']), 'under-
five_deaths'] = min(
        x_train_out['under-five_deaths'])

    x_train.loc[x_train.Polio > (Q3.Polio + 1.5 * IQR.Polio), 'Polio'] =
max(x_train_out.Polio)
    x_train.loc[x_train.Polio < (Q1.Polio - 1.5 * IQR.Polio), 'Polio'] =
min(x_train_out.Polio)

    x_train.loc[
        x_train.Total_expenditure > (Q3.Total_expenditure + 1.5 *
IQR.Total_expenditure), 'Total_expenditure'] = max(
        x_train_out.Total_expenditure)
    x_train.loc[
        x_train.Total_expenditure < (Q1.Total_expenditure - 1.5 *
IQR.Total_expenditure), 'Total_expenditure'] = min(
        x_train_out.Total_expenditure)

    x_train.loc[x_train.Diphtheria > (Q3.Diphtheria + 1.5 * IQR.Diphtheria),
'Diphtheria'] = max(x_train_out.Diphtheria)
    x_train.loc[x_train.Diphtheria < (Q1.Diphtheria - 1.5 * IQR.Diphtheria),
'Diphtheria'] = min(x_train_out.Diphtheria)
```

```
    x_train.loc[x_train.HIV_AIDS > (Q3.HIV_AIDS + 1.5 * IQR.HIV_AIDS), 'HIV_AIDS']
= max(x_train_out.HIV_AIDS)
    x_train.loc[x_train.HIV_AIDS < (Q1.HIV_AIDS - 1.5 * IQR.HIV_AIDS), 'HIV_AIDS']
= min(x_train_out.HIV_AIDS)

    x_train.loc[x_train.GDP > (Q3.GDP + 1.5 * IQR.GDP), 'GDP'] =
max(x_train_out.GDP)
    x_train.loc[x_train.GDP < (Q1.GDP - 1.5 * IQR.GDP), 'GDP'] =
min(x_train_out.GDP)

    x_train.loc[x_train.Population > (Q3.Population + 1.5 * IQR.Population),
'Population'] = max(x_train_out.Population)
    x_train.loc[x_train.Population < (Q1.Population - 1.5 * IQR.Population),
'Population'] = min(x_train_out.Population)

    x_train.loc[x_train['thinness_1-19_years'] > (
            Q3['thinness_1-19_years'] + 1.5 * IQR['thinness_1-19_years']),
'thinness_1-19_years'] = max(
        x_train_out['thinness_1-19_years'])
    x_train.loc[x_train['thinness_1-19_years'] < (
            Q1['thinness_1-19_years'] - 1.5 * IQR['thinness_1-19_years']),
'thinness_1-19_years'] = min(
        x_train_out['thinness_1-19_years'])

    x_train.loc[x_train['thinness_5-9_years'] > (
            Q3['thinness_5-9_years'] + 1.5 * IQR['thinness_5-9_years']),
'thinness_5-9_years'] = max(
        x_train_out['thinness_5-9_years'])
    x_train.loc[x_train['thinness_5-9_years'] < (
            Q1['thinness_5-9_years'] - 1.5 * IQR['thinness_5-9_years']),
'thinness_5-9_years'] = min(
        x_train_out['thinness_5-9_years'])

    x_train.loc[x_train.Income_composition_of_resources > (
            Q3.Income_composition_of_resources + 1.5 *
IQR.Income_composition_of_resources), 'Income_composition_of_resources'] =
max(
        x_train_out.Income_composition_of_resources)
```

```
    x_train.loc[x_train.Income_composition_of_resources < (
                        Q1.Income_composition_of_resources - 1.5 *
IQR.Income_composition_of_resources), 'Income_composition_of_resources'] =
min(
            x_train_out.Income_composition_of_resources)

    x_train.loc[x_train.Schooling > (Q3.Schooling + 1.5 * IQR.Schooling), 'Schooling']
= max(x_train_out.Schooling)
    x_train.loc[x_train.Schooling < (Q1.Schooling - 1.5 * IQR.Schooling), 'Schooling']
= min(x_train_out.Schooling)

    return x_train


def showplot(x_train, time):
    if time == 'before':
        print("Before dealing with the outliers")
    if time == 'after':
        print("After dealing with the outliers")
    # feature "Adult_Mortality"
    sns.boxplot(data=x_train['Adult_Mortality'])
    plt.xlabel('Adult_Mortality')
    plt.show()

    # feature "infant_deaths"
    sns.boxplot(data=x_train['infant_deaths'])
    plt.xlabel('infant_deaths')
    plt.show()

    # feature "Alcohol"
    sns.boxplot(data=x_train['Alcohol'])
    plt.xlabel('Alcohol')
    plt.show()
    # feature "percentage_expenditure "
    sns.boxplot(data=x_train['percentage_expenditure'])
    plt.xlabel('percentage_expenditure')
    plt.show()
    # feature "Hepatitis_B"
```

```
sns.boxplot(data=x_train['Hepatitis_B'])
plt.xlabel('Hepatitis_B')
plt.show()
# feature "Measles"
sns.boxplot(data=x_train['Measles'])
plt.xlabel('Measles')
plt.show()
# feature "BMI"
# sns.boxplot(data=x_train['BMI'])
plt.xlabel('BMI')
plt.show()
# feature "under-five_deaths"
sns.boxplot(data=x_train['under-five_deaths'])
plt.xlabel('under-five_deaths')
plt.show()
# feature "Polio"
sns.boxplot(data=x_train['Polio'])
plt.xlabel('Polio')
plt.show()
# feature "Total_expenditure"
sns.boxplot(data=x_train['Total_expenditure'])
plt.xlabel('Total_expenditure')
plt.show()
# feature "Diphtheria "
sns.boxplot(data=x_train['Diphtheria'])
plt.xlabel('Diphtheria')
plt.show()
# feature "HIV_AIDS"
sns.boxplot(data=x_train['HIV_AIDS'])
plt.xlabel('HIV_AIDS')
plt.show()
# feature "GDP"
sns.boxplot(data=x_train['GDP'])
plt.xlabel('GDP')
plt.show()
# feature "Population"
sns.boxplot(data=x_train['Population'])
plt.xlabel('Population')
```

```
    plt.show()
    # feature "thinness_1-19_years"
    sns.boxplot(data=x_train['thinness_1-19_years'])
    plt.xlabel('thinness_1-19_years')
    plt.show()
    # feature "thinness_5-9_years"
    sns.boxplot(data=x_train['thinness_5-9_years'])
    plt.xlabel('thinness_5-9_years')
    plt.show()    # feature "Income_composition_of_resources"
    sns.boxplot(data=x_train['Income_composition_of_resources'])
    plt.xlabel('Income_composition_of_resources')
    plt.show()    # feature "Schooling"
    sns.boxplot(data=x_train['Schooling'])
    plt.xlabel('Schooling')


def main():
    pd.set_option('display.max_columns', None)
    from sklearn.neighbors import KNeighborsClassifier
    data = pd.read_csv('Life Expectancy Data.csv')
    # print(data.columns)
    data.rename(columns={'Life expectancy ': "Life_expectancy"}, inplace=True)
    data.rename(columns={'Adult Mortality': "Adult_Mortality"}, inplace=True)
    data.rename(columns={'infant deaths': "infant_deaths"}, inplace=True)
    data.rename(columns={'percentage expenditure': 'percentage_expenditure'},
inplace=True)
    data.rename(columns={'Hepatitis B': "Hepatitis_B"}, inplace=True)
    data.rename(columns={' BMI ': "BMI"}, inplace=True)
    data.rename(columns={'under-five deaths ': "under-five_deaths"}, inplace=True)
    data.rename(columns={'Total expenditure': "Total_expenditure"}, inplace=True)
    data.rename(columns={' HIV/AIDS': "HIV/AIDS"}, inplace=True)
    data.rename(columns={' thinness  1-19 years': "thinness_1-19_years"},
inplace=True)
    data.rename(columns={' thinness 5-9 years': "thinness_5-9_years"},
inplace=True)
    data.rename(columns={'Income composition of resources':
"Income_composition_of_resources"}, inplace=True)
    data.rename(columns={'HIV/AIDS': "HIV_AIDS"}, inplace=True)
```

```
    data.rename(columns={'Measles ': "Measles"}, inplace=True)
    data.rename(columns={'Diphtheria ': "Diphtheria"}, inplace=True)

    # delet the data with null life expectancy value
    data['Life_expectancy'] = data['Life_expectancy'].fillna(999)
    drop_index = data[(data.Life_expectancy == 999)].index.tolist()
    data = data.drop(drop_index)
    # print(data['Life_expectancy'])

    # make life_expectancy as our output
    labels = data.loc[:, ['Life_expectancy']]

    # del data['Life_expectancy']

    # deal with categorical data "status" since it contains numerical quality
    data['Status'] = data['Status'].str.replace('Developed', '2', case=False)
    data['Status'] = data['Status'].str.replace('Developing', '1', case=False)
    # print('data=',data)
    #       data=pd.get_dummies(data, prefix=['Country'], columns=['Country'])
    #       print('size after dummy', data.shape)

    # Separate the data to train, val and test data
    x, x_test, y, y_test = train_test_split(data, labels, test_size=0.2, train_size=0.8,
random_state=1)
    x_train, x_val, y_train, y_val = train_test_split(x, y, test_size=0.2, train_size=0.8,
random_state=50)
    # print(x_train)
    # calculate the null value in each column
    NoNull_train = x_train.isnull().sum(axis=0)
    # print('--------Null data in train----------')
    # print(NoNull_train)
    # print('--------Null data in train----------')

    '''
    sns.distplot(y_train['Life_expectancy'])
    mu=y_train['Life_expectancy'].mean()
    sigma=y_train['Life_expectancy'].std()
    #Now plot the distribution
```

```
    plt.legend(['Original dist. ($\mu=$ {:.2f} and $\sigma=$ {:.2f} )'.format(mu,
sigma)],loc='best')
    plt.ylabel('Frequency')
    plt.title('Life_expectancy')

    #Get also the QQ-plot
    fig = plt.figure()
    res = stats.probplot(y_train['Life_expectancy'], plot=plt)
    plt.show()
    '''

    # make the target_train data become more closed to the normal distribution
    # ----------------------------------------------
    # We use the numpy fuction log1p which    applies log(1+x) to all elements of
the column
    '''
    #Check the new distribution
    sns.distplot(y_train['Life_expectancy'] , fit=norm);

    # Get the fitted parameters used by the function
    (mu, sigma) = norm.fit(y_train['Life_expectancy'])
    print( '\n mu = {:.2f} and sigma = {:.2f}\n'.format(mu, sigma))

    #Now plot the distribution
    plt.legend(['Normal dist. ($\mu=$ {:.2f} and $\sigma=$ {:.2f} )'.format(mu,
sigma)],loc='best')
    plt.ylabel('Frequency')
    plt.title('Life Expectancy')

    #Get also the QQ-plot
    fig = plt.figure()
    res = stats.probplot(y_train['Life_expectancy'], plot=plt)
    plt.show()
    '''
    Null_train_ratio = (x_train.isnull().sum() / len(x_train)) * 100
    Null_train_ratio = Null_train_ratio.sort_values(ascending=False)
    AllNull_train_ratio = Null_train_ratio.drop(Null_train_ratio[Null_train_ratio ==
0].index)
```

```
    missing_train_ratio = pd.DataFrame({'Missing train data ratio':
AllNull_train_ratio})
    # print(missing_train_ratio)

    f, ax = plt.subplots(figsize=(15, 12))
    plt.xticks(rotation='90')    # ratate direction of words for each feature
    sns.barplot(x=Null_train_ratio.index, y=Null_train_ratio)
    plt.xlabel('Features', fontsize=15)
    plt.ylabel('Percent of missing values', fontsize=15)
    plt.title('missing data percentage by feature', fontsize=15)
    plt.show()

    # print("--------------train data description-------------")
    # print(x_train.describe())
    # draw to recognize the outliers
    # Show the boxplot before dealing with outliers
    # feature "Adult_Mortality"
    # showplot(x_train,'before')

    # try to remove the outliers
    # print('len of x train=',len(x_train))
    Q1 = x_train.quantile(0.25)
    Q3 = x_train.quantile(0.75)
    IQR = Q3 - Q1
    x_train = outlier_remove_traindata(x_train, Q1, Q3, IQR)
    x_val = outlier_remove_traindata(x_val, Q1, Q3, IQR)
    # print('finish removing the outliers')

    # ----------finished dealing with the outlier in each features----------

    # Show the boxplot after dealing with outliers
    # feature "Adult_Mortality"
    # showplot(x_train,'after')

    ####fill the missing data with mean value
    x_train = x_train.fillna(x_train.mean())
    x_val = x_val.fillna(x_train.mean())
    x = x.fillna(x.mean())
```

```
    x_test = x_test.fillna(x.mean())
    ####fill the missing data with median
    # x_train=x_train.fillna(x_train.median())
    # x_val=x_val.fillna(x_train.median())

    ####fill the missing data with -1
    # x_train=x_train.fillna(-1)
    # x_val=x_val.fillna(-1)

    ####fill the missing data with 0
    # x_train=x_train.fillna(0)
    # x_val=x_val.fillna(0)

    NoNull_train = x_train.isnull().sum(axis=0)
    # print('--------Null data in train----------')
    # print('NoNull_train',NoNull_train)

    # separate the string and numerical data
    x_train_str = x_train.loc[:, ["Country"]]
    x_train_num = x_train.loc[:, ['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']]
    x_val_str = x_val.loc[:, ["Country"]]
    x_val_num = x_val.loc[:, ['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']]
    x_str = x.loc[:, ["Country"]]
```

```
    x_num = x.loc[:, ['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']]
    x_test_num = x_test.loc[:, ['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']]

    x_train_str = pd.get_dummies(x_train_str)
    # Try to see the correlation between country(after get dummy) and life
expectancy
    country_col = x_train_str.columns
    # print('country column =',len(country_col))
    # Try to see the correlation between country(after get dummy) and life
expectancy
    x_train_str["Life_expectancy"] = y_train

    country_corrmat = x_train_str.corr()
    cols = abs(country_corrmat).nlargest(10,
'Life_expectancy')['Life_expectancy'].index
    cm = np.corrcoef(x_train_str[cols].values.T)
    sns.set(font_scale=1.25)
    plt.subplots(figsize=(15, 12))
    hm = sns.heatmap(cm, cbar=True, annot=True, square=True, fmt='.2f',
annot_kws={'size': 10}, yticklabels=cols.values,
                     xticklabels=cols.values)
    bottom, top = hm.get_ylim()
    hm.set_ylim(bottom + 0.5, top - 0.5)
```

```
        plt.title('The country that is most related to life expectancy')
        plt.show()
        # Since the highest correlation between country and life expectancy is 0.17, we
decide not to use the feature "country"

        # standardize the data
        standar = preprocessing.StandardScaler().fit(x_train_num)
        x_train = standar.transform(x_train_num)
        x_val = standar.transform(x_val_num)
        x_train1 = pd.DataFrame(data=x_train, columns=x_train_num.columns)
        x_val1 = pd.DataFrame(data=x_val, columns=x_train_num.columns)
        # Correlation map to see how features are correlated with LifeExpectancy
        corrmat = x_train1.corr()
        plt.subplots(figsize=(18, 15))
        ax = sns.heatmap(corrmat, vmax=1, annot=True, square=True, vmin=0)
        bottom, top = ax.get_ylim()
        ax.set_ylim(bottom + 0.5, top - 0.5)
        plt.title('Correlation Heatmap Between Each Feature')
        plt.show()

        cols = abs(corrmat).nlargest(19, 'Life_expectancy')['Life_expectancy'].index
        cm = np.corrcoef(x_train1[cols].values.T)
        sns.set(font_scale=1.25)
        plt.subplots(figsize=(15, 12))
        plt.title('18 Features that are most related to Life Expectancy')
        hm = sns.heatmap(cm, cbar=True, annot=True, square=True, fmt='.2f',
annot_kws={'size': 10}, yticklabels=cols.values,
                         xticklabels=cols.values)
        bottom, top = hm.get_ylim()
        hm.set_ylim(bottom + 0.5, top - 0.5)

        plt.show()

        cols = abs(corrmat).nlargest(21, 'Life_expectancy')['Life_expectancy'].index
        related_col =
cols.drop(['Life_expectancy']).drop(['Status']).drop(['Hepatitis_B']).drop(['infant_deat
hs']).drop(
```

```
['GDP']).drop(['Measles']).drop(['Population']).drop(['percentage_expenditure']).drop
(['Diphtheria'])
    # related_col = related_col.drop(['Status'])

    # related_col = related_col.drop(['under-five_deaths'])
    # print("The columns most related to Life expectancy=", related_col)

    # transform the test data
    standar_all = preprocessing.StandardScaler().fit(x_num)
    x = standar_all.transform(x_num)
    x_test = standar_all.transform(x_test_num)
    x1 = pd.DataFrame(data=x, columns=x_train_num.columns)
    x_test1 = pd.DataFrame(data=x_test, columns=x_train_num.columns)

    x_train = x_train1[related_col]
    x_val = x_val1[related_col]
    x = x1[related_col]
    x_test = x_test1[related_col]
    '''
    # Choose the optimal no of features => 18 features (k=19, we need to deduct
'life expectancy')
    meanerror_NoFeature_val = []
    mse_NoFeature_val = []
    for k in range(5, 21):
        cols = abs(corrmat).nlargest(k, 'Life_expectancy')['Life_expectancy'].index
        related_col = cols.drop(['Life_expectancy'])
        x_train = x_train1[related_col]
        x_val = x_val1[related_col]
        mean_train = np.mean(y_train)
        mean_train_array = [x * mean_train for x in np.ones(y_train.shape[0])]
        np.ones(y_train.shape)
        # We found that the min MSE happened when n_estimators=25
        error_train = []
        error_val = []
        mse_train = []
        mse_val = []
        for j in range(10):
            pre_train, pre_X_train_pick, pre_y_train, pre_y_train_pick =
```

```
train_test_split(x_train, y_train,

test_size=1 / 3)
                RF = RandomForestRegressor(n_estimators=25, bootstrap=True,
max_features=3)
                RF.fit(pre_X_train_pick, pre_y_train_pick.values.ravel())
                predict_train = RF.predict(x_train)
                predict_val = RF.predict(x_val)
                acc_train = RF.score(x_train, y_train)
                acc_val = RF.score(x_val, y_val)
                error_train = np.append(error_train, 1 - acc_train)
                error_val = np.append(error_val, 1 - acc_val)
                mse_train = np.append(mse_train,
mean_squared_error(predict_train, y_train))
                mse_val = np.append(mse_val, mean_squared_error(predict_val,
y_val))
            meanerror_val = np.mean(error_val)
            mean_mse_val = np.mean(mse_val)
            meanerror_NoFeature_val = np.append(meanerror_NoFeature_val,
meanerror_val)
            mse_NoFeature_val = np.append(mse_NoFeature_val, mean_mse_val)
        # print('mean error in validation set when 4~19 features')
        # print(meanerror_NoFeature_val)
        # print(' ')
        # print('mse in validation set when 4~19 features')
        # print(mse_NoFeature_val)
        print('When we set the number of features from 4-19,')
        print('min mean error = %.2f and min MSE = %.2f in validation set when we
chose %.0f correlated features' % (
        min(meanerror_NoFeature_val), min(mse_NoFeature_val),
np.argmin(meanerror_NoFeature_val) + 4))
        X = np.arange(4, 20)
        plt.plot(X, meanerror_NoFeature_val, label='Mean Error')
        plt.title('Number of Features vs Mean Error')
        plt.ylabel('Mean Error')
        plt.xlabel('Number of features')
        plt.show()
        plt.plot(X, mse_NoFeature_val, label='MSE')
```

```
    plt.title('Number of features vs MSE')
    plt.ylabel('MSE')
    plt.xlabel('Number of features')
    plt.show()
    '''
    ################Regression Tree#########################
    # Find the best way to fill the missing data
    print('###################Random Forest
Regression########################')
    meanerror_train = []
    meanerror_val = []
    mean_mse_train = []
    mean_mse_val = []
    for i in range(50):
        error_train = []
        error_val = []
        mse_train = []
        mse_val = []
        for j in range(10):
            pre_train, pre_X_train_pick, pre_y_train, pre_y_train_pick =
train_test_split(x_train, y_train,

test_size=1 / 3)
            RF = RandomForestRegressor(n_estimators=i + 1, bootstrap=True,
random_state=0)
            RF.fit(pre_X_train_pick, pre_y_train_pick.values.ravel())
            predict_train = RF.predict(x_train)
            predict_val = RF.predict(x_val)
            acc_train = RF.score(x_train, y_train)
            acc_val = RF.score(x_val, y_val)
            error_train = np.append(error_train, 1 - acc_train)
            error_val = np.append(error_val, 1 - acc_val)
            mse_train = np.append(mse_train,
mean_squared_error(predict_train, y_train))
            mse_val = np.append(mse_val, mean_squared_error(predict_val,
y_val))
        meanerror_train = np.append(meanerror_train, np.mean(error_train))
        meanerror_val = np.append(meanerror_val, np.mean(error_val))
```

```
            mean_mse_train = np.append(mean_mse_train, np.mean(mse_train))
            mean_mse_val = np.append(mean_mse_val, np.mean(mse_val))
        # print('When fill the missing data with mean:')
        # print('mean error in training set =', meanerror_train)
        # print('mean error in validation set =', meanerror_val)
        # print('MSE in training set =',mean_mse_train)
        # print('MSE in validation set =',mean_mse_val)
        print("we got the min MSE value=%.3f and error rate=%.3f in validation set
when there are %.0f trees" % (
        min(mean_mse_val), min(meanerror_val), np.argmin(mean_mse_val) + 1))

        fi = pd.DataFrame({'feature': list(x_train.columns),
                                    'importance': RF.feature_importances_}). \
            sort_values('importance', ascending=False)
        #print('importance=', fi)

        # plot the figure
        X = np.arange(1, 51)
        fig, (ax1, ax2) = plt.subplots(nrows=2, ncols=1)
        fig.suptitle('Random Forest Regression')
        ax1.plot(X, meanerror_train, label='train data')
        ax1.plot(X, meanerror_val, color='r', label='val data')
        ax1.set_ylabel('Error Rate')
        ax1.plot(np.argmin(meanerror_val) + 1, min(meanerror_val), '*',
label='minimum', color='b', markersize=15)
        ax1.legend(loc='best')

        ax2.plot(X, mean_mse_train, label='train data')
        ax2.plot(X, mean_mse_val, color='r', label='val data')
        ax2.set_ylabel('MSE')
        ax2.set_xlabel('Number of trees')
        ax2.plot(np.argmin(mean_mse_val) + 1, min(mean_mse_val), '*',
label='minimum', color='b', markersize=15)
        ax2.legend(loc='best')
        plt.show()
        '''
        #We found that the min MSE happened when n_estimators=25
        error_train = []
```

```
        error_val = []
        mse_train = []
        mse_val = []
        for j in range(10):
            pre_train, pre_X_train_pick, pre_y_train, pre_y_train_pick =
train_test_split(x_train, y_train, test_size=1 / 3)
            RF = RandomForestRegressor(n_estimators=25, bootstrap=True,
max_features=3)
            RF.fit(pre_X_train_pick, pre_y_train_pick.values.ravel())
            predict_train = RF.predict(x_train)
            predict_val = RF.predict(x_val)
            acc_train = RF.score(x_train, y_train)
            acc_val = RF.score(x_val, y_val)
            error_train = np.append(error_train, 1 - acc_train)
            error_val = np.append(error_val, 1 - acc_val)
            mse_train = np.append(mse_train, mean_squared_error(predict_train,
y_train))
            mse_val = np.append(mse_val, mean_squared_error(predict_val, y_val))
        meanerror_train = np.mean(error_train)
        meanerror_val = np.mean(error_val)
        mean_mse_train = np.mean(mse_train)
        mean_mse_val = np.mean(mse_val)
        print('When fill the missing data with 0 and n_estimator = 25:')
        print('mean error in training set =', meanerror_train)
        print('mean error in validation set =', meanerror_val)
        print('MSE in training set =',mean_mse_train)
        print('MSE in validation set =',mean_mse_val)
        '''
        ##############Linear Regression################
        # linear regression (general)
        print('####################linear regression
(general)########################')
        lin_mse_val = []
        lin_error_val = []
        lin_mse_train = []
        lin_error_train = []
        corrmat = x_train1.corr()
        for k in range(5, 21):
```

```
            cols = abs(corrmat).nlargest(k, 'Life_expectancy')['Life_expectancy'].index
            related_col = cols.drop(['Life_expectancy'])
            x_train = x_train1[related_col]
            x_val = x_val1[related_col]
            reg = linear_model.LinearRegression()
            reg.fit(x_train, y_train)
            reg_predict_val = reg.predict(x_val)
            reg_predict_train = reg.predict(x_train)
            reg_acc_val = reg.score(x_val, y_val)
            reg_acc_train = reg.score(x_train, y_train)
            lin_mse_val = np.append(lin_mse_val, mean_squared_error(y_val,
reg_predict_val))
            lin_mse_train = np.append(lin_mse_train, mean_squared_error(y_train,
reg_predict_train))
            lin_error_val = np.append(lin_error_val, 1 - reg_acc_val)
            lin_error_train = np.append(lin_error_train, 1 - reg_acc_train)
        print('Linear regression when featrues=4-19')
        # print('error rate in validation set =')
        # print(lin_error_val)
        # print('MSE in validation set')
        # print(lin_mse_val)
        print('We can find the min error rate= %.4f and min MSE= %.4f when there are
%.0f features' % (
        min(lin_error_val), min(lin_mse_val), np.argmin(lin_mse_val) + 4))

        X = np.arange(4, 20, 1)
        fig, (ax1, ax2) = plt.subplots(nrows=2, ncols=1)
        fig.suptitle('Linear Regression')
        ax1.plot(X, lin_error_val, color='r', label='Validation set')
        ax1.plot(X, lin_error_train, label='Training set')
        ax1.set_ylabel('Error Rate')
        ax1.plot(np.argmin(lin_error_val) + 4, min(lin_error_val), '*', label='minimum',
color='b', markersize=15)
        ax1.legend(loc='best')

        ax2.plot(X, lin_mse_val, color='r', label='Validation set')
        ax2.plot(X, lin_mse_train, label='Training set')
        ax2.set_ylabel('MSE')
```

```
    ax2.set_xlabel('Number of Features')
    ax2.plot(np.argmin(lin_mse_val) + 4, min(lin_mse_val), '*', label='minimum',
color='b', markersize=15)
    ax2.legend(loc='best')
    plt.show()
    #############Ridge Regression################
    print('###############Ridge Regression(without CV)################')

    X = np.linspace(-3, 1, 30)
    cols = abs(corrmat).nlargest(19, 'Life_expectancy')[
        'Life_expectancy'].index    # Select 7 features that are the most related to
life expectancy
    related_col = cols.drop(['Life_expectancy'])
    x_train = x_train1[related_col]
    x_val = x_val1[related_col]
    rid_mse_val = []
    rid_mse_train = []
    rid_error_val = []
    rid_error_train = []
    for i in X:
        ridge = linear_model.Ridge(alpha=10 ** i, normalize=True)
        ridge.fit(x_train, y_train)
        ridge_predict_val = ridge.predict(x_val)
        ridge_predict_train = ridge.predict(x_train)
        ridge_acc_val = ridge.score(x_val, y_val)
        ridge_acc_train = ridge.score(x_train, y_train)
        rid_mse_val = np.append(rid_mse_val, mean_squared_error(y_val,
ridge_predict_val))
        rid_mse_train = np.append(rid_mse_train, mean_squared_error(y_train,
ridge_predict_train))
        rid_error_val = np.append(rid_error_val, 1 - ridge_acc_val)
        rid_error_train = np.append(rid_error_train, 1 - ridge_acc_train)
    # print('error rate in validation set =')
    # print(rid_error_val)
    # print('MSE in validation set')
    # print(rid_mse_val)
    print('We can find the min error rate= %.4f and min MSE= %.4f when alpha=
%.6f ' % (
```

```
        min(rid_error_val), min(rid_mse_val), 10 ** X[np.argmin(rid_mse_val)]))
        # print('alpha=',X)
        fig, (ax1, ax2) = plt.subplots(nrows=2, ncols=1)
        fig.suptitle('Ridge Regression (Without CV)')
        ax1.plot(X, rid_error_train, label='Training set')
        ax1.plot(X, rid_error_val, color='r', label='Validation set')
        ax1.set_ylabel('Error Rate')
        ax1.set_xlabel('log(alpha)')
        ax1.plot(X[np.argmin(rid_error_val)], min(rid_error_val), '*', label='minimum',
color='b', markersize=15)
        ax1.legend(loc='best')

        ax2.plot(X, rid_mse_train, label='Training set')
        ax2.plot(X, rid_mse_val, color='r', label='Validation set')
        ax2.set_ylabel('MSE')
        ax2.set_xlabel('log(alpha)')
        ax2.plot(X[np.argmin(rid_mse_val)], min(rid_mse_val), '*', label='minimum',
color='b', markersize=15)
        ax2.legend(loc='best')
        plt.show()

        print('###############Ridge Regression(with CV)################')
        X = np.linspace(-3, 1, 30)
        cols = abs(corrmat).nlargest(19, 'Life_expectancy')[
            'Life_expectancy'].index    # Select 7 features that are the most related to
life expectancy
        related_col = cols.drop(['Life_expectancy']).drop(['Status'])

        #print('in ridge regression')
        #print('related col=', related_col)
        xx = x1[related_col]
        ridCV_err = np.zeros([len(X), 6])
        ridCV_mse = np.zeros([len(X), 6])
        for i in range(5, 11):    # column
            kfold = KFold(n_splits=i, shuffle=True)
            for j in range(len(X)):    # row
                ridgeCV = linear_model.RidgeCV(alphas=10 ** X, normalize=True)
                ridCV_neg_mse = cross_val_score(ridgeCV, xx, y, cv=kfold,
```

```
scoring='neg_mean_squared_error')
                ridCV_score = cross_val_score(ridgeCV, xx, y, cv=kfold, scoring='r2')
                # ridCV_err[j][i-5] = 1- np.mean(ridCV_score)
                ridCV_mse[j][i - 5] = np.mean(ridCV_neg_mse) * (-1)
    min_err_index = np.unravel_index(ridCV_err.argmin(), ridCV_err.shape)
    min_mse_index = np.unravel_index(ridCV_mse.argmin(), ridCV_mse.shape)
    print('When we use Ridge Regression with cross validation')
    print('We got the min MSE value= %.3f when we applied %.0f fold and alpha =
%.5f' % (
    ridCV_mse.min(), min_mse_index[1] + 5, 10 ** X[min_mse_index[0]]))
    print('')
    bestK_alpha_mse = ridCV_mse[:, min_mse_index[1]].reshape((ridCV_mse[:,
min_mse_index[1]].shape[0], 1))
    # bestK_alpha_err =
ridCV_err[:,min_err_index[1]].reshape((ridCV_err[:,min_err_index[1]].shape[0],1))
    fig, ax2 = plt.subplots(nrows=1, ncols=1)
    fig.suptitle('Ridge Regression (With CV when K= %.0f)' % (min_mse_index[1] +
5))

    # ax1.plot(X, bestK_alpha_err)
    ax1.set_ylabel('Error Rate')
    ax1.set_xlabel('log(alpha)')
    # ax1.plot(X[min_err_index[0]],bestK_alpha_err.min(),'*',
label='minimum',color='b',markersize=15)
    ax1.legend(loc='best')

    ax2.plot(X, bestK_alpha_mse)
    ax2.set_ylabel('MSE')
    ax2.set_xlabel('log(alpha)')
    ax2.plot(X[min_mse_index[0]], bestK_alpha_mse.min(), '*', label='minimum',
color='b', markersize=15)
    ax2.legend(loc='best')
    # plt.show()

    ############Lasso Regression################
    print('###############Lasso Regression(without CV)################')
    X = np.linspace(-3, 1, 30)
    x_train = x_train1[related_col]
```

```
    x_val = x_val1[related_col]
    lasso_mse_val = []
    lasso_mse_train = []
    lasso_error_val = []
    lasso_error_train = []
    for i in X:
        lasso = linear_model.Lasso(alpha=10 ** i, normalize=True)
        lasso.fit(x_train, y_train)
        lasso_predict_val = lasso.predict(x_val)
        lasso_predict_train = lasso.predict(x_train)
        lasso_acc_val = lasso.score(x_val, y_val)
        lasso_acc_train = lasso.score(x_train, y_train)
        lasso_mse_val = np.append(lasso_mse_val, mean_squared_error(y_val,
lasso_predict_val))
        lasso_mse_train = np.append(lasso_mse_train,
mean_squared_error(y_train, lasso_predict_train))
        lasso_error_val = np.append(lasso_error_val, 1 - lasso_acc_val)
        lasso_error_train = np.append(lasso_error_train, 1 - lasso_acc_train)
    # print('error rate in validation set =')
    # print(lasso_error_val)
    # print('MSE in validation set')
    # print(lasso_mse_val)
    print('We can find the min error rate= %.4f and min MSE= %.4f when alpha=
%.6f ' % (
    min(lasso_error_val), min(lasso_mse_val), 10 ** X[np.argmin(lasso_mse_val)]))

    fig, (ax1, ax2) = plt.subplots(nrows=2, ncols=1)
    fig.suptitle('Lasso Regression (Without CV)')
    ax1.plot(X, lasso_error_train, label='Training set')
    ax1.plot(X, lasso_error_val, color='r', label='Validation set')
    ax1.set_ylabel('Error Rate')
    ax1.plot(X[np.argmin(lasso_error_val)], min(lasso_error_val), '*',
label='minimum', color='b', markersize=15)
    ax1.legend(loc='lower right')

    ax2.plot(X, lasso_mse_train, label='Training set')
    ax2.plot(X, lasso_mse_val, color='r', label='Validation set')
    ax2.set_ylabel('MSE')
```

```
    ax2.set_xlabel('log(alpha)')
    ax2.plot(X[np.argmin(lasso_mse_val)], min(lasso_mse_val), '*',
label='minimum', color='b', markersize=15)
    plt.show()

    print('###############Lasso Regression(with CV)###############')
    X = np.linspace(-3, 1, 30)
    xx = x1[related_col]
    lassoCV_err = np.zeros([len(X), 6])
    lassoCV_mse = np.zeros([len(X), 6])
    for i in range(5, 11):    # column
        kfold = KFold(n_splits=i, shuffle=True)
        for j in range(len(X)):    # row
            lassoCV = linear_model.LassoCV(alphas=10 ** X, normalize=True)
            lassoCV_neg_mse = cross_val_score(lassoCV, xx, y, cv=kfold,
scoring='neg_mean_squared_error')
            lassoCV_score = cross_val_score(lassoCV, xx, y, cv=kfold, scoring='r2')
            # lassoCV_err[j][i-5] = 1- np.mean(lassoCV_score)
            lassoCV_mse[j][i - 5] = np.mean(lassoCV_neg_mse) * (-1)
    # min_err_index=np.unravel_index(lassoCV_err.argmin(), lassoCV_err.shape)
    min_mse_index = np.unravel_index(lassoCV_mse.argmin(), lassoCV_mse.shape)
    print('When we use Lasso Regression with cross validation')
    print('We got the min MSE value= %.3f when we applied %.0f fold and alpha =
%.5f' % (
    lassoCV_mse.min(), min_mse_index[1] + 5, 10 ** X[min_mse_index[0]]))

    bestK_alpha_mse = lassoCV_mse[:, min_mse_index[1]].reshape((lassoCV_mse[:,
min_mse_index[1]].shape[0], 1))
    # bestK_alpha_err =
lassoCV_err[:,min_err_index[1]].reshape((lassoCV_err[:,min_err_index[1]].shape[0],
1))
    fig, ax2 = plt.subplots(nrows=1, ncols=1)
    fig.suptitle('Lasso Regression (With CV when K= %.0f)' % (min_mse_index[1] +
5))

    # ax1.plot(X, bestK_alpha_err)
    ax1.set_ylabel('Error Rate')
    ax1.set_xlabel('log(alpha)')
```

```
        # ax1.plot(X[min_err_index[0]],bestK_alpha_err.min(),'*',
label='minimum',color='b',markersize=15)
        ax1.legend(loc='best')

        ax2.plot(X, bestK_alpha_mse)
        ax2.set_ylabel('MSE')
        ax2.set_xlabel('log(alpha)')
        ax2.plot(X[min_mse_index[0]], bestK_alpha_mse.min(), '*', label='minimum',
color='b', markersize=15)
        ax2.legend(loc='best')

        # plt.show()

        avg_ytest = np.mean(y_test)
        one_array = np.ones([len(y_test), 1])
        mean_arr = avg_ytest['Life_expectancy'] * one_array
        baseline_mse = mean_squared_error(y_test, mean_arr)
        baseline_err = 1 - r2_score(y_test, mean_arr)

        print('Number of training data (original):', len(y))
        print('Number of test data:', len(y_test))
        print('Number of training data (New):', len(y_train))
        print('Number of validation data:', len(y_val))

        print('####################Final model: Random Forest
Regression########################')
        cols = abs(corrmat).nlargest(21, 'Life_expectancy')['Life_expectancy'].index
        related_col =
cols.drop(['Life_expectancy']).drop(['Status']).drop(['Hepatitis_B']).drop(['infant_deat
hs']).drop(

['GDP']).drop(['Measles']).drop(['Population']).drop(['percentage_expenditure']).drop
(['Diphtheria'])

        error_x = []
        error_test = []
        mse_x = []
        mse_test = []
```

```
    oob_error = []
    for j in range(10):
        pre_train, pre_X_train_pick, pre_y_train, pre_y_train_pick =
train_test_split(x, y, test_size=1 / 3)
        RF = RandomForestRegressor(n_estimators=38, bootstrap=True,
random_state=0, oob_score=True)
        RF.fit(pre_X_train_pick, pre_y_train_pick.values.ravel())
        predict_x = RF.predict(x)
        predict_test = RF.predict(x_test)
        acc_x = RF.score(x, y)
        acc_test = RF.score(x_test, y_test)
        error_x = np.append(error_x, 1 - acc_x)
        error_test = np.append(error_test, 1 - acc_test)
        oob_error = np.append(oob_error, 1 - RF.oob_score_)
        mse_x = np.append(mse_x, mean_squared_error(predict_x, y))
        mse_test = np.append(mse_test, mean_squared_error(predict_test,
y_test))
    mean_oob_err = np.mean(oob_error)
    meanerror_x = np.mean(error_x)
    meanerror_test = np.mean(error_test)
    mean_mse_x = np.mean(mse_x)
    mean_mse_test = np.mean(mse_test)
    var_mse_test = np.var(mse_test)
    var_err_test = np.var(error_test)
    print('In our final model (38 trees):')
    print('In the whole training data')
    print('Mean MSE =', mean_mse_x)
    print('Mean Error Rate =', meanerror_x)
    print('In the test data')
    print('Mean MSE = %.3f with variance = %.3f ' % (mean_mse_test,
var_mse_test))
    print('Mean Error Rate = %.5f with variance = %.5f ' % (meanerror_test,
var_err_test))
    print('Out Of Sample Error = ', mean_oob_err)
    print('####################Baseline#######################')
    print('The baseline for the test data:')
    print('MSE = ', baseline_mse)
    print('Error Rate=', baseline_err)
```

```
    #Draw the 2D plot
    feature = ['HIV_AIDS', 'Income_composition_of_resources', 'Adult_Mortality',
'Schooling']
    for i in feature:
        for j in range(10):
            pre_train, pre_X_train_pick, pre_y_train, pre_y_train_pick =
train_test_split(x, y, test_size=1 / 3)
            RF = RandomForestRegressor(n_estimators=38, bootstrap=True,
random_state=0)
            RF.fit(pre_X_train_pick[[i]], pre_y_train_pick.values.ravel())
            predict_x = RF.predict(x[[i]])
            predict_test = RF.predict(x_test[[i]])
        X_grid = np.arange(min(x[i]), max(x[i]), 0.001)
        # reshape for reshaping the data into a len(X_grid)*1 array,
        # i.e. to make a column out of the X_grid value
        X_grid = X_grid.reshape((len(X_grid), 1))

        # Scatter plot for original data
        plt.scatter(x[i], y, color='blue', label='training data points')
        # plot predicted data
        plt.plot(X_grid, RF.predict(X_grid), color='green', label='regression
function')
        plt.title('Random Forest Regression')
        plt.xlabel(i)
        plt.ylabel('Life expectancy')
        plt.legend(loc='best')
        plt.show()


if __name__ == "__main__":
    main()
```