Predicting Future Life Expectancy

December 10, 2024

[ ]:
```
!pip install --quiet optuna
```

[ ]:
```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
from matplotlib.ticker import MaxNLocator

from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

from sklearn.impute import KNNImputer
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
from xgboost import XGBRegressor
from sklearn.linear_model import Ridge
from sklearn.cluster import KMeans
from scipy.stats import pearsonr

from sklearn.model_selection import cross_val_score, GroupKFold

from matplotlib.ticker import MaxNLocator

import optuna

from google.colab import files
SAVE_FIGS = True
```

[ ]:
```
# Load and format data
url='https://drive.google.com/file/d/1p2uueb0ivqfmZmMUTQoGmHDtD44wGjH_/view?
↪usp=drive_link'
url='https://drive.google.com/uc?id=' + url.split('/')[-2]
dataset = pd.read_csv(url)

url='https://drive.google.com/file/d/1o2BeS77z1X6oyMT2qkaSzWzu3kMdweu0/view?
↪usp=sharing'
url='https://drive.google.com/uc?id=' + url.split('/')[-2]
dataset_with_gdp = pd.read_csv(url)
```

```
# Rename columns to not have trailing whitespace
dataset.rename(mapper=(lambda c: c.strip()), axis=1, inplace=True)

# Drop rows that don't have life expectancy data
dataset = dataset[dataset['Life expectancy'].notnull()]

# One-hot encode categorical data
dataset = pd.get_dummies(dataset, columns=['Status'])
```

[ ]:
```
dataset.isna().sum()
```

[ ]:
```
Country                              0
Year                                 0
Life expectancy                      0
Adult Mortality                      0
infant deaths                        0
Alcohol                            193
percentage expenditure               0
Hepatitis B                        553
Measles                              0
BMI                                 32
under-five deaths                    0
Polio                               19
Total expenditure                  226
Diphtheria                          19
HIV/AIDS                             0
GDP                                443
Population                         644
thinness  1-19 years                32
thinness 5-9 years                  32
Income composition of resources    160
Schooling                          160
Status_Developed                     0
Status_Developing                    0
dtype: int64
```

[ ]:
```
# Impute values with K-nearest-neighbors
imputer = KNNImputer(n_neighbors=5, weights='distance')
numeric_features = dataset.select_dtypes(include=['number']).columns
dataset[numeric_features] = imputer.fit_transform(dataset[numeric_features])
```

[ ]:
```
dataset.isna().sum()
```

[ ]:
```
Country                            0
Year                               0
Life expectancy                    0
```

```
Adult Mortality                    0
infant deaths                      0
Alcohol                            0
percentage expenditure             0
Hepatitis B                        0
Measles                            0
BMI                                0
under-five deaths                  0
Polio                              0
Total expenditure                  0
Diphtheria                         0
HIV/AIDS                           0
GDP                                0
Population                         0
thinness  1-19 years               0
thinness 5-9 years                 0
Income composition of resources    0
Schooling                          0
Status_Developed                   0
Status_Developing                  0
dtype: int64
```

[ ]:
```
dataset_with_gdp = dataset_with_gdp.sort_values(by=['Country', 'Year'])
dataset = dataset.sort_values(by=['Country', 'Year'])
dataset_with_gdp
```

[ ]:
```
          Country  Region  Year  Infant_deaths  Under_five_deaths  \
68    Afghanistan    Asia  2000           90.5              129.2
1693  Afghanistan    Asia  2001           87.9              125.2
679   Afghanistan    Asia  2002           85.3              121.1
1221  Afghanistan    Asia  2003           82.7              116.9
1147  Afghanistan    Asia  2004           80.0              112.6
…             …      …    …              …                  …
255      Zimbabwe  Africa  2011           50.8               80.8
1489     Zimbabwe  Africa  2012           46.5               72.2
1201     Zimbabwe  Africa  2013           44.8               66.3
1005     Zimbabwe  Africa  2014           42.9               62.7
1480     Zimbabwe  Africa  2015           42.1               61.3

      Adult_mortality  Alcohol_consumption  Hepatitis_B  Measles   BMI  …  \
68           310.8305                 0.02           62       12  21.7  …
1693         304.8580                 0.02           63       13  21.8  …
679          298.8855                 0.02           64       14  21.9  …
1221         292.0365                 0.02           65       15  22.0  …
1147         285.1880                 0.02           67       16  22.1  …
…                 …                    …          …        …    …   …
255          466.2650                 3.91           94       64  23.7  …
```

```
1489        423.4420                  3.93          97        64  23.7  …
1201        405.0080                  4.11          95        64  23.7  …
1005        386.5745                  4.22          91        64  23.8  …
1480        368.1410                  3.84          87        64  23.8  …

      Diphtheria  Incidents_HIV  GDP_per_capita  Population_mln  \
68            24           0.02             148           20.78
1693          33           0.02             163           21.61
679           36           0.02             320           22.60
1221          41           0.02             332           23.68
1147          50           0.02             323           24.73
…            …            …              …              …
255           93           6.05            1249           12.89
1489          95           5.13            1432           13.12
1201          95           4.77            1435           13.35
1005          91           4.29            1444           13.59
1480          87           3.86            1445           13.81

      Thinness_ten_nineteen_years  Thinness_five_nine_years  Schooling  \
68                            2.3                       2.5        2.2
1693                          2.1                       2.4        2.2
679                          19.9                       2.2        2.3
1221                         19.7                      19.9        2.4
1147                         19.5                      19.7        2.5
…                            …                        …         …
255                           6.8                       6.7        7.3
1489                          6.5                       6.4        7.9
1201                          6.2                       6.0        8.0
1005                          5.9                       5.7        8.2
1480                          5.6                       5.5        8.2

      Economy_status_Developed  Economy_status_Developing  Life_expectancy
68                           0                          1             55.8
1693                         0                          1             56.3
679                          0                          1             56.8
1221                         0                          1             57.3
1147                         0                          1             57.8
…                         …                         …               …
255                          0                          1             52.9
1489                         0                          1             55.0
1201                         0                          1             56.9
1005                         0                          1             58.4
1480                         0                          1             59.5

[2864 rows x 21 columns]
```

[ ]:
```python
# Find missing countries
countries = dataset['Country'].unique()
countries_with_gdp = dataset_with_gdp['Country'].unique()

# Replace changed country names
dataset_with_gdp = dataset_with_gdp.replace('Eswatini','Swaziland')
dataset_with_gdp = dataset_with_gdp.replace('Congo, Dem. Rep.', 'Democratic␣
↪Republic of the Congo')
replaced = ['Eswatini', 'Congo, Dem. Rep.']
for c in countries:
  if c in countries_with_gdp:
    continue
  for d in countries_with_gdp:
    if d in countries or d in replaced:
      continue
    if d in c:
      dataset_with_gdp = dataset_with_gdp.replace(d,c)
      replaced.append(d)
    elif d[:4] == c[:4]:
      if 'United' in c: # Should be caught and changed in previous step, if␣
↪not, skip
        continue
      else:
        dataset_with_gdp = dataset_with_gdp.replace(d,c)
        replaced.append(d)
    elif d[-4:] == c[-4:]:
      if d == 'Slovak Republic': # Edge case --> normally becomes Lao PRC␣
↪instead of Slovakia
        continue
      dataset_with_gdp = dataset_with_gdp.replace(d,c)
      replaced.append(d)

# Find missing countries
missing = []
countries_with_gdp = dataset_with_gdp['Country'].unique()

for c in countries:
  if c not in countries_with_gdp:
    missing.append(c)
print('Missing Countries: ',missing)


dataset_with_gdp = dataset_with_gdp.sort_values(by=['Country', 'Year'])
dataset = dataset.sort_values(by=['Country', 'Year'])
data = dataset.copy()
for i in missing: # Drop missing countries
  data = data[data['Country'] != i]
```

```
# Reindex and change GDP
dataset_with_gdp = dataset_with_gdp.reset_index(drop=True)
data = data.reset_index(drop=True)
data['GDP'] = dataset_with_gdp['GDP_per_capita']
```

```
Missing Countries:  ["Democratic People's Republic of Korea", 'Republic of
Korea', 'South Sudan', 'Sudan']
```

[ ]:
```
dataset = data
```

1 How well can we predict life expectancy in n years given a year of data?

[ ]:
```
n = 5
y_feature = 'Life expectancy'
```

[ ]:
```
dataset.columns
```

[ ]:
```
Index(['Country', 'Year', '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',
       'Status_Developed', 'Status_Developing'],
      dtype='object')
```

1.1 Setup

[ ]:
```
# I will regress for deviation of life expectancy from the mean in n years.
mean_life_exp = dataset.groupby('Year')[y_feature].mean()
# Trying to pick features that won't "reveal" too much? Something to work on
X_features = [
    y_feature, # The model needs to know where the feature began to predict␣
 ↪where it's going
    # 'Year', # Don't use the year
    'Alcohol',
    'percentage expenditure',
    'Hepatitis B',
    'Measles',
    'BMI',
    'Polio',
    'Total expenditure',
    'Diphtheria',
    'HIV/AIDS',
    'GDP',
```

```
    'Income composition of resources',
    'Schooling',
    'thinness  1-19 years',
    'thinness 5-9 years'
]
```

1.2 Add deviation from mean by year as a feature

[ ]:
```
y_feature_dev = f'{y_feature} deviation' # This will be the deviation of the␣
 ↪feature from the mean between all countries by year
y_feature_dev_in_n = f'{y_feature} deviation in n years' # Ditto, in n years␣
 ↪from the current year
y_feature_dev_change = f'{y_feature} deviation change' # y_feature_dev_in_n -␣
 ↪y_feature_dev
```

[ ]:
```
data_by_year = dataset.groupby(by='Year')
mean_by_year = data_by_year[y_feature].mean()
mean_by_year
```

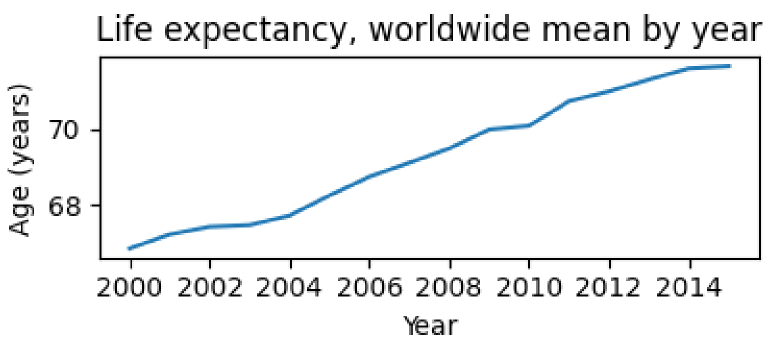
[ ]:
```
Year
2000.0    66.851397
2001.0    67.222346
2002.0    67.425698
2003.0    67.469274
2004.0    67.718436
2005.0    68.246927
2006.0    68.741899
2007.0    69.110615
2008.0    69.487709
2009.0    69.987151
2010.0    70.086034
2011.0    70.732402
2012.0    70.992179
2013.0    71.305028
2014.0    71.596648
2015.0    71.654749
Name: Life expectancy, dtype: float64
```

[ ]:
```
# Plot the worldwide mean of life expectancy
plt.figure(figsize=(4.2, 2))
plt.plot(mean_by_year.index, mean_by_year)
plt.title("Life expectancy, worldwide mean by year")
plt.xlabel('Year')
plt.ylabel('Age (years)')

# Force integer x-axis labels
```

```python
plt.gca().xaxis.set_major_locator(MaxNLocator(integer=True))

plt.tight_layout()

# Save plot
filename = 'worldwide_mean.pdf'
plt.savefig(filename)
if SAVE_FIGS:
    files.download(filename)

plt.show()
```

<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>



[ ]:
```python
dataset['Country'].unique()
```

[ ]:
```
array(['Afghanistan', 'Albania', 'Algeria', 'Angola',
       'Antigua and Barbuda', 'Argentina', 'Armenia', 'Australia',
       'Austria', 'Azerbaijan', 'Bahamas', 'Bahrain', 'Bangladesh',
       'Barbados', 'Belarus', 'Belgium', 'Belize', 'Benin', 'Bhutan',
       'Bolivia (Plurinational State of)', 'Bosnia and Herzegovina',
       'Botswana', 'Brazil', 'Brunei Darussalam', 'Bulgaria',
       'Burkina Faso', 'Burundi', 'Cabo Verde', 'Cambodia', 'Cameroon',
       'Canada', 'Central African Republic', 'Chad', 'Chile', 'China',
       'Colombia', 'Comoros', 'Congo', 'Costa Rica', 'Croatia', 'Cuba',
       'Cyprus', 'Czechia', "Côte d'Ivoire",
       'Democratic Republic of the Congo', 'Denmark', 'Djibouti',
       'Dominican Republic', 'Ecuador', 'Egypt', 'El Salvador',
       'Equatorial Guinea', 'Eritrea', 'Estonia', 'Ethiopia', 'Fiji',
       'Finland', 'France', 'Gabon', 'Gambia', 'Georgia', 'Germany',
       'Ghana', 'Greece', 'Grenada', 'Guatemala', 'Guinea',
       'Guinea-Bissau', 'Guyana', 'Haiti', 'Honduras', 'Hungary',
```

```
'Iceland', 'India', 'Indonesia', 'Iran (Islamic Republic of)',
'Iraq', 'Ireland', 'Israel', 'Italy', 'Jamaica', 'Japan', 'Jordan',
'Kazakhstan', 'Kenya', 'Kiribati', 'Kuwait', 'Kyrgyzstan',
"Lao People's Democratic Republic", 'Latvia', 'Lebanon', 'Lesotho',
'Liberia', 'Libya', 'Lithuania', 'Luxembourg', 'Madagascar',
'Malawi', 'Malaysia', 'Maldives', 'Mali', 'Malta', 'Mauritania',
'Mauritius', 'Mexico', 'Micronesia (Federated States of)',
'Mongolia', 'Montenegro', 'Morocco', 'Mozambique', 'Myanmar',
'Namibia', 'Nepal', 'Netherlands', 'New Zealand', 'Nicaragua',
'Niger', 'Nigeria', 'Norway', 'Oman', 'Pakistan', 'Panama',
'Papua New Guinea', 'Paraguay', 'Peru', 'Philippines', 'Poland',
'Portugal', 'Qatar', 'Republic of Moldova', 'Romania',
'Russian Federation', 'Rwanda', 'Saint Lucia',
'Saint Vincent and the Grenadines', 'Samoa',
'Sao Tome and Principe', 'Saudi Arabia', 'Senegal', 'Serbia',
'Seychelles', 'Sierra Leone', 'Singapore', 'Slovakia', 'Slovenia',
'Solomon Islands', 'Somalia', 'South Africa', 'Spain', 'Sri Lanka',
'Suriname', 'Swaziland', 'Sweden', 'Switzerland',
'Syrian Arab Republic', 'Tajikistan', 'Thailand',
'The former Yugoslav republic of Macedonia', 'Timor-Leste', 'Togo',
'Tonga', 'Trinidad and Tobago', 'Tunisia', 'Turkey',
'Turkmenistan', 'Uganda', 'Ukraine', 'United Arab Emirates',
'United Kingdom of Great Britain and Northern Ireland',
'United Republic of Tanzania', 'United States of America',
'Uruguay', 'Uzbekistan', 'Vanuatu',
'Venezuela (Bolivarian Republic of)', 'Viet Nam', 'Yemen',
'Zambia', 'Zimbabwe'], dtype=object)
```

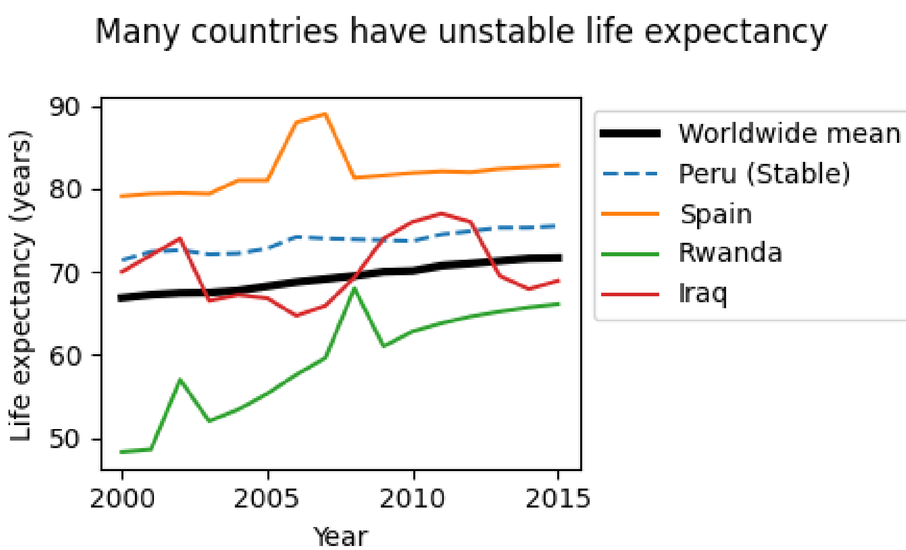
[ ]:

```
plt.figure(figsize=(5, 3))
# Plot outliers against the mean
plt.plot(mean_by_year.index, mean_by_year, label='Worldwide mean',␣
 ↪color='black', lw=3)

stable_data = dataset[dataset['Country'] == 'Peru']
plt.plot(stable_data['Year'], stable_data[y_feature], '--', label='Peru␣
 ↪(Stable)')

countries_to_plot = ['Spain', 'Rwanda', 'Iraq']
## Fetch country data
for country in countries_to_plot:
    country_data = dataset[dataset['Country'] == country]
    plt.plot(country_data['Year'], country_data[y_feature], label=country)

plt.suptitle("Many countries have unstable life expectancy")
plt.xlabel('Year')
plt.ylabel('Life expectancy (years)')
plt.legend(bbox_to_anchor=(1, 1))
```

```
plt.tight_layout()
# Save plot
filename = 'unstable.pdf'
plt.savefig(filename)
if SAVE_FIGS:
    files.download(filename)

plt.show()
```

<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>




```
[ ]: def get_deviation(row):
       year = row['Year']
       return row[y_feature] - mean_by_year[year]
     dataset[y_feature_dev] = dataset.apply(get_deviation, axis=1)
```

```
[ ]: max_year = dataset['Year'].max()
     def get_deviation_in_n_years(row):
       country = row['Country']
       year = row['Year']
       if year + n > max_year:
         return None
       else:
         return dataset[(dataset['Country'] == country) & (dataset['Year'] == year +
      ↪n)][y_feature_dev].item()

     dataset[y_feature_dev_in_n] = dataset.apply(get_deviation_in_n_years, axis=1)
```

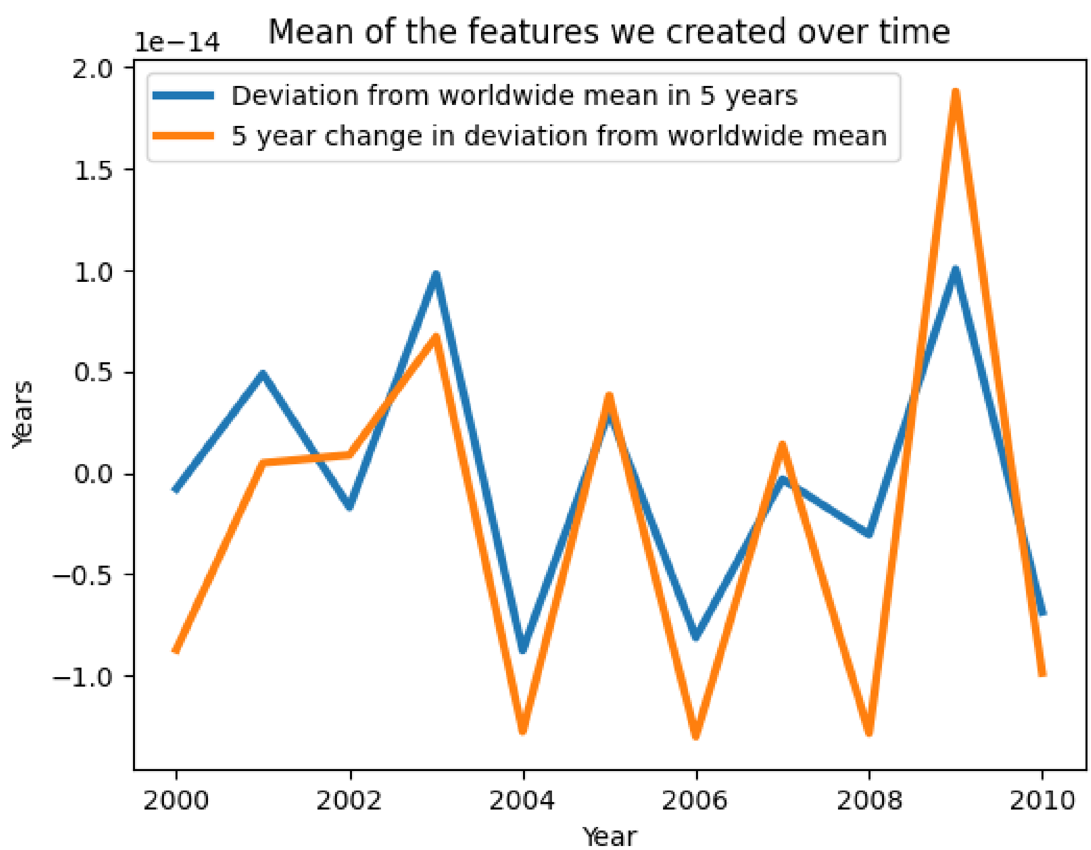
```
dataset[y_feature_dev_change] = dataset[y_feature_dev_in_n] -␣
↪dataset[y_feature_dev]
# Drop the rows that don't have data in n years, save a copy first though
dataset_full = dataset.copy(deep=True)
dataset = dataset[dataset[y_feature_dev_in_n].notna()]
```

[ ]:
```
dataset[['Country', 'Year', y_feature, y_feature_dev, y_feature_dev_in_n,␣
↪y_feature_dev_change]].head(15)
```

[ ]:
```
        Country    Year  Life expectancy  Life expectancy deviation  \
0   Afghanistan  2000.0             54.8                 -12.051397
1   Afghanistan  2001.0             55.3                 -11.922346
2   Afghanistan  2002.0             56.2                 -11.225698
3   Afghanistan  2003.0             56.7                 -10.769274
4   Afghanistan  2004.0             57.0                 -10.718436
5   Afghanistan  2005.0             57.3                 -10.946927
6   Afghanistan  2006.0             57.3                 -11.441899
7   Afghanistan  2007.0             57.5                 -11.610615
8   Afghanistan  2008.0             58.1                 -11.387709
9   Afghanistan  2009.0             58.6                 -11.387151
10  Afghanistan  2010.0             58.8                 -11.286034
16      Albania  2000.0             72.6                   5.748603
17      Albania  2001.0             73.6                   6.377654
18      Albania  2002.0             73.3                   5.874302
19      Albania  2003.0             72.8                   5.330726

    Life expectancy deviation in n years  Life expectancy deviation change
0                             -10.946927                          1.104469
1                             -11.441899                          0.480447
2                             -11.610615                         -0.384916
3                             -11.387709                         -0.618436
4                             -11.387151                         -0.668715
5                             -11.286034                         -0.339106
6                             -11.532402                         -0.090503
7                             -11.492179                          0.118436
8                             -11.405028                         -0.017318
9                             -11.696648                         -0.309497
10                             -6.654749                          4.631285
16                              5.253073                         -0.495531
17                              5.458101                         -0.919553
18                              6.789385                          0.915084
19                              5.812291                          0.481564
```

1.3 Basic analysis

[ ]:
```python
# Plotting the features we created
years_range = dataset['Year'].unique()
plt.plot(
    years_range,
    dataset.groupby('Year')[y_feature_dev_in_n].mean(),
    label=f'Deviation from worldwide mean in {n} years',
    lw=3
)
plt.plot(
    years_range,
    dataset.groupby('Year')[y_feature_dev_change].mean(),
    label=f'{n} year change in deviation from worldwide mean',
    lw=3
)
plt.xlabel('Year')
plt.ylabel('Years')
plt.title('Mean of the features we created over time')
plt.legend()
plt.show()
```



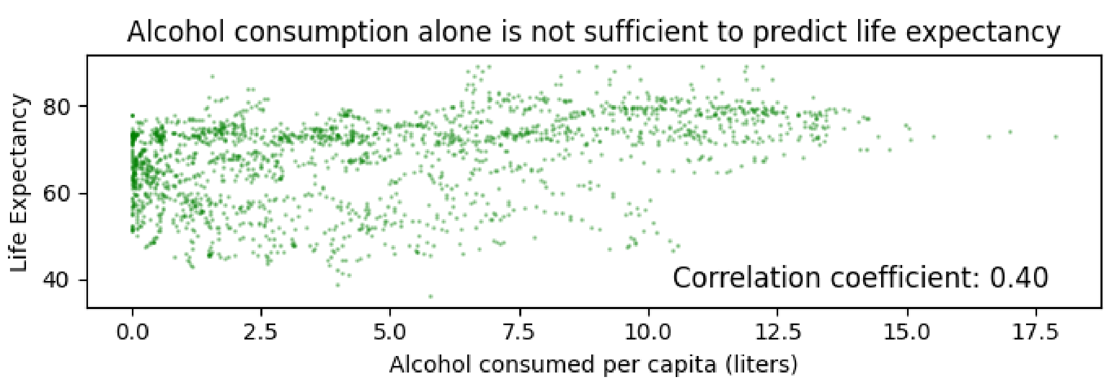
[ ]:
```
# Calculate correlation coefficient between population and life expectancy
corr_feature = 'Alcohol'
corr_coeff = pearsonr(dataset[corr_feature], dataset[y_feature])
corr_coeff.statistic
```

[ ]: 0.3955694243703762

[ ]:
```
plt.figure(figsize=(7, 2.5))
plt.scatter(dataset[corr_feature], dataset[y_feature], s=1, alpha=0.3,␣
 ↪color='green')
#plt.xscale('log')
plt.xlabel('Alcohol consumed per capita (liters)')
plt.ylabel('Life Expectancy')
plt.title('Alcohol consumption alone is not sufficient to predict life␣
 ↪expectancy')
# Add correlation coefficient text at the bottom right
plt.text(0.95, 0.05, f'Correlation coefficient: {corr_coeff.statistic:.2f}',␣
 ↪ha='right', va='bottom', transform=plt.gca().transAxes, fontsize=12,␣
 ↪bbox=dict(facecolor='white', edgecolor='none', alpha=0.7))

# Save plot
filename = 'alcohol.pdf'
plt.tight_layout()
plt.savefig(filename)
if SAVE_FIGS:
    files.download(filename)

plt.show()
```

<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>



1.4 Model selection

We will select the best model with cross-validation with various models.

The split in each fold here will be a little funky. In a given fold, test and train can't have overlap in country, unless they also have no overlap in time, otherwise there will be information spill. Guaranteeing no overlap in time is tricky, so I'll start by setting apart countries for test.

In each fold, the validation set will have 20% of the countries.

There will also be a test set containing 15% of all the countries—the other 85% is what will be used in the validation set.

[ ]:
```
# Random number generator
gen = np.random.default_rng()
```

[ ]:
```
n_folds = 5
all_countries = dataset['Country'].unique()

# Reserve countries for test set
n_countries_test = int(0.15 * len(dataset['Country'].unique()))
test_countries = gen.choice(dataset['Country'], n_countries_test)

# Train-test split
data_test  = dataset[ dataset['Country'].isin(test_countries)]
data_train = dataset[~dataset['Country'].isin(test_countries)]

# Group-based splitter for cross-validation
group_kfold = GroupKFold(n_splits=n_folds)
groups = data_train['Country']

# Generate folds by country
folds = []
for train_idx, val_idx in group_kfold.split(data_train, groups=groups):
    train_data = dataset.iloc[train_idx]
    val_data = dataset.iloc[val_idx]
    folds.append((train_data, val_data))
```

[ ]:
```
def X_y_split(data):
    """ Split a dataset or a fold into X (features) and y (labels). """
    return data[X_features], data[y_feature_dev_change]

X_train, y_train = X_y_split(data_train)
X_test, y_test = X_y_split(data_test)
```

```
[ ]: optuna.logging.set_verbosity(optuna.logging.WARNING)
     n_trials = 50
```

```
[ ]: ## Random forest
     best_rf = None
     best_rf_score = -np.inf

     def objective(trial):
         global best_rf, best_rf_score

         # Select parameters. If log=True, prefer smaller numbers
         n_estimators = trial.suggest_int('n_estimators', 25, 500, log=True)
         max_depth = trial.suggest_int('max_depth', 2, 32)
         bootstrap = trial.suggest_categorical('bootstrap', [True, False])
         regressor = RandomForestRegressor(
             n_estimators=n_estimators,
             max_depth=max_depth,
             bootstrap=bootstrap,
             n_jobs=-1
         )

         # Train and evalate for each fold using sklearn utilities
         scores = cross_val_score(
             regressor,
             X_train,
             y_train,
             cv=group_kfold,
             groups=groups,
             scoring='r2'
         )
         score = np.mean(scores)

         # Save best model
         if score > best_rf_score:
             best_rf = regressor
             best_rf_score = score

         return score

     study_rf = optuna.create_study(direction='maximize')
     study_rf.optimize(objective, n_trials=n_trials, show_progress_bar=True,␣
      ↪n_jobs=-1)
```

```
  0%|          | 0/50 [00:00<?, ?it/s]
```

```
[ ]: print("Random Forest, Best Trial:")
     print(f"  Score: {study_rf.best_trial.value:.4f}")
```

```
print("  Parameters:")
for key, value in study_rf.best_trial.params.items():
    print(f"    {key}: {value}")
```

```
Random Forest, Best Trial:
  Score: 0.3049
  Parameters:
    n_estimators: 172
    max_depth: 6
    bootstrap: True
```

[ ]:
```
## XGBoost
best_xg = None
best_xg_score = -np.inf
def objective(trial):
    global best_xg, best_xg_score

    # Select parameters. If log=True, prefer smaller numbers
    n_estimators = trial.suggest_int('n_estimators', 25, 500, log=True)
    max_depth = trial.suggest_int('max_depth', 2, 32)
    learning_rate = trial.suggest_float('learning_rate', 0.01, 0.1)
    subsample = trial.suggest_float('subsample', 0.5, 1.0)
    colsample_bytree = trial.suggest_float('colsample_bytree', 0.5, 1.0)
    gamma = trial.suggest_float('gamma', 0, 10)

    regressor = XGBRegressor(
        n_estimators=n_estimators,
        max_depth=max_depth,
        learning_rate=learning_rate,
        subsample=subsample,
        colsample_bytree=colsample_bytree,
        gamma=gamma
    )

    # Train and evalate for each fold using sklearn utilities
    scores = cross_val_score(
        regressor,
        X_train,
        y_train,
        cv=group_kfold,
        groups=groups,
        scoring='r2'
    )

    score = np.mean(scores)

    # Save best model
```

```
    if score > best_xg_score:
        best_xg = regressor
        best_xg_score = score

    return score


study_xg = optuna.create_study(direction='maximize')
study_xg.optimize(objective, n_trials=n_trials, show_progress_bar=True,␣
 ↪n_jobs=-1)
```

```
  0%|          | 0/50 [00:00<?, ?it/s]
```

[ ]:
```
print("XGBoost, Best Trial:")
print(f"  Score: {study_xg.best_trial.value:.4f}")
print("  Parameters:")
for key, value in study_xg.best_trial.params.items():
    print(f"    {key}: {value}")
```

```
XGBoost, Best Trial:
  Score: 0.3191
  Parameters:
    n_estimators: 243
    max_depth: 12
    learning_rate: 0.015164812218471683
    subsample: 0.8899350531401716
    colsample_bytree: 0.8608917725519883
    gamma: 7.443110257368188
```

[ ]:
```
## Gradient boosting regressor
best_gb = None
best_gb_score = -np.inf
def objective(trial):
    global best_gb, best_gb_score

    # Select parameters. If log=True, prefer smaller numbers
    n_estimators = trial.suggest_int('n_estimators', 25, 300, log=True)
    learning_rate = trial.suggest_float('learning_rate', 0.01, 1)
    subsample = trial.suggest_float('subsample', 0.5, 1)
    regressor = GradientBoostingRegressor(
        n_estimators=n_estimators,
        learning_rate=learning_rate,
        subsample=subsample
    )

    # Train and evalate for each fold using sklearn utilities
    scores = cross_val_score(
```

```
        regressor,
        X_train,
        y_train,
        cv=group_kfold,
        groups=groups,
        scoring='r2'
    )
    score = np.mean(scores)

    # Save best model
    if score > best_gb_score:
        best_gb = regressor
        best_gb_score = score

    return score


study_gb = optuna.create_study(direction='maximize')
study_gb.optimize(objective, n_trials=n_trials, show_progress_bar=True,␣
 ↪n_jobs=-1)
```

```
  0%|          | 0/50 [00:00<?, ?it/s]
```

[ ]:
```
print("Gradient Boosting, Best Trial:")
print(f"  Score: {study_gb.best_trial.value:.4f}")
print("  Parameters:")
for key, value in study_gb.best_trial.params.items():
    print(f"    {key}: {value}")
```

```
Gradient Boosting, Best Trial:
  Score: 0.3079
  Parameters:
    n_estimators: 58
    learning_rate: 0.07164775493579317
    subsample: 0.8312091757491455
```

[ ]:
```
## Linear regression (ridge regression)
best_lr = None
best_lr_score = -np.inf

def objective(trial):
    global best_lr, best_lr_score

    # Select parameters. If log=True, prefer smaller numbers
    alpha = trial.suggest_float('alpha', 1e-5, 100.0, log=True)

    regressor = Ridge(alpha=alpha)
```

```
    # Train and evalate for each fold using sklearn utilities
    scores = cross_val_score(
        regressor,
        X_train,
        y_train,
        cv=group_kfold,
        groups=groups,
        scoring='r2'
    )
    score = np.mean(scores)

    # Save best model
    if score > best_lr_score:
        best_lr = regressor
        best_lr_score = score

    return score


study_lr = optuna.create_study(direction='maximize')
study_lr.optimize(objective, n_trials=n_trials, show_progress_bar=True,␣
↪n_jobs=-1)
```

```
  0%|          | 0/50 [00:00<?, ?it/s]
```

[ ]:
```
print("Linear Regression (Ridge), Best Trial:")
print(f"  Score: {study_lr.best_trial.value:.4f}")
print("  Parameters:")
for key, value in study_lr.best_trial.params.items():
    print(f"    {key}: {value}")
```

```
Linear Regression (Ridge), Best Trial:
  Score: 0.1878
  Parameters:
    alpha: 99.8658341047081
```

[ ]:
```
models = [best_rf, best_gb, best_xg, best_lr]
scores = [best_rf_score, best_gb_score, best_xg_score, best_lr_score]
best_index = np.argmax(scores)

print(f'Best score: {scores[best_index]}')
best_regressor = models[best_index]
best_regressor.fit(X_train, y_train)
```

```
Best score: 0.3191344243213058
```

[ ]:
```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=0.8608917725519883, device=None,
             early_stopping_rounds=None, enable_categorical=False,
             eval_metric=None, feature_types=None, gamma=7.443110257368188,
             grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=0.015164812218471683,
             max_bin=None, max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=12, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             multi_strategy=None, n_estimators=243, n_jobs=None,
             num_parallel_tree=None, random_state=None, …)
```

[ ]:
```python
best_model_df = pd.DataFrame([{
    'Model': best_regressor.__class__.__name__,
    'Best Score': scores[best_index],
    **best_regressor.get_params()  # add the parameters of the model as columns
}])
best_model_df.T
```

[ ]:

	0
Model	XGBRegressor
Best Score	0.319134
objective	reg:squarederror
base_score	None
booster	None
callbacks	None
colsample_bylevel	None
colsample_bynode	None
colsample_bytree	0.860892
device	None
early_stopping_rounds	None
enable_categorical	False
eval_metric	None
feature_types	None
gamma	7.44311
grow_policy	None
importance_type	None
interaction_constraints	None
learning_rate	0.015165
max_bin	None
max_cat_threshold	None
max_cat_to_onehot	None
max_delta_step	None
max_depth	12
max_leaves	None
min_child_weight	None

```
missing                                    NaN
monotone_constraints                      None
multi_strategy                            None
n_estimators                               243
n_jobs                                    None
num_parallel_tree                         None
random_state                              None
reg_alpha                                 None
reg_lambda                                None
sampling_method                           None
scale_pos_weight                          None
subsample                             0.889935
tree_method                               None
validate_parameters                       None
verbosity                                 None
```

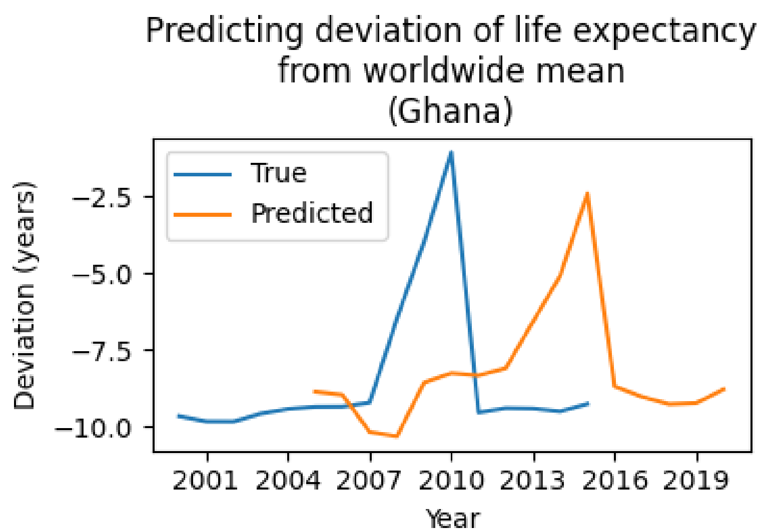
It works. Let's visualize the results.

[ ]:
```python
regressor = best_regressor

# Randomly pick some countries from the test set
n_countries = 12
countries = gen.choice(test_countries, n_countries, replace=False)
# Plot for each the predictions vs the truth
for country in countries:
    country_data = dataset_full[dataset_full['Country'] == country]

    plt.figure(figsize=(4, 3))
    plt.title(f'Predicting deviation of {y_feature.lower()}\nfrom worldwide␣
↪mean\n({country})')
    plt.xlabel('Year')
    plt.ylabel('Deviation (years)')

    plt.plot(
        country_data['Year'],
        country_data[y_feature_dev],
        label=f'True'
    )

    # The model predicts change in life expectancy.
    plt.plot(
        [y + n for y in country_data['Year']],
        [dev + delta_dev
            for dev, delta_dev
```

```
            in zip(
                country_data[y_feature_dev],
                regressor.predict(country_data[X_features])
                )
        ],
        label=f'Predicted'
    )

    plt.legend()
    # Make year ticks on the bottom integers
    plt.gca().xaxis.set_major_locator(MaxNLocator(integer=True))
    plt.tight_layout()
    if SAVE_FIGS:
        plt.savefig(f'pred_{country}.pdf')
        files.download(f'pred_{country}.pdf')
    plt.show()
    print()
```

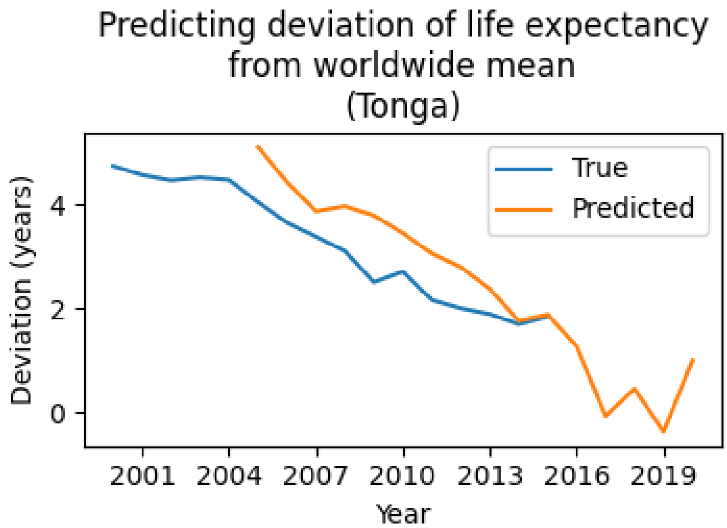
```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.Javascript object>
```



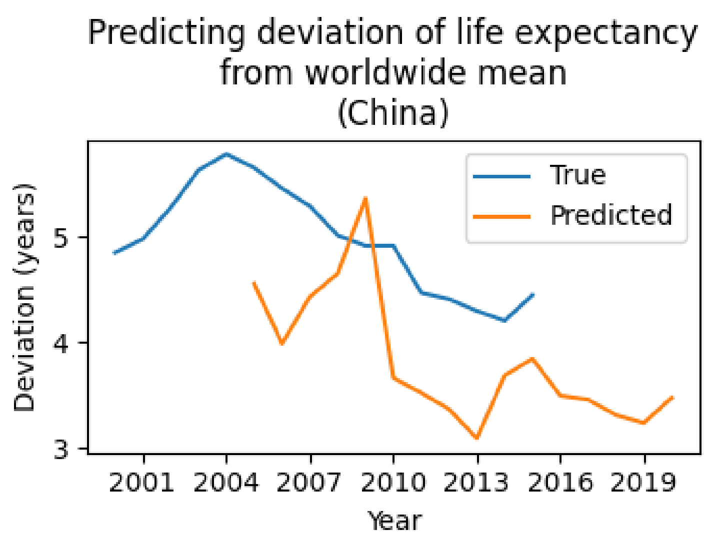

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.Javascript object>
```



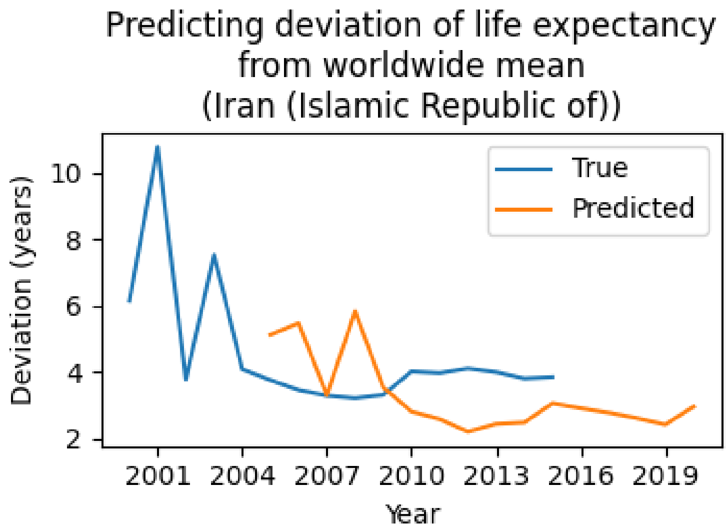
<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>



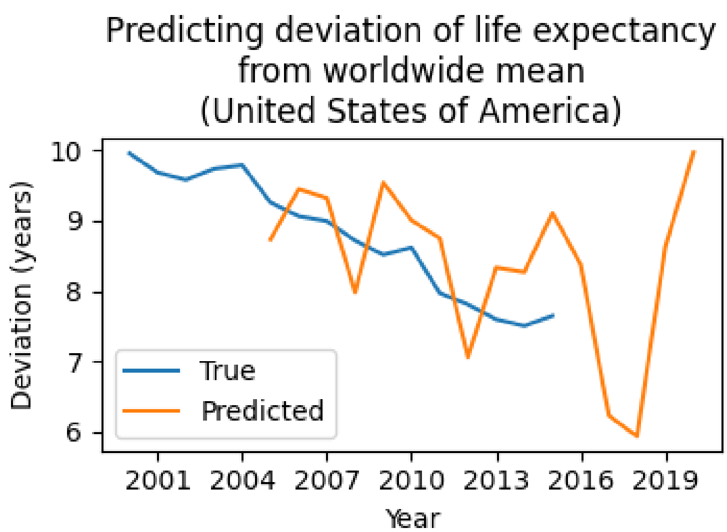
<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>



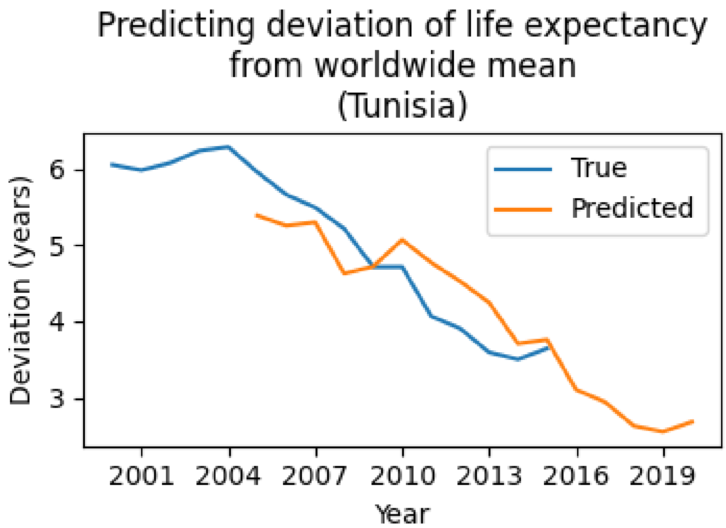
<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>



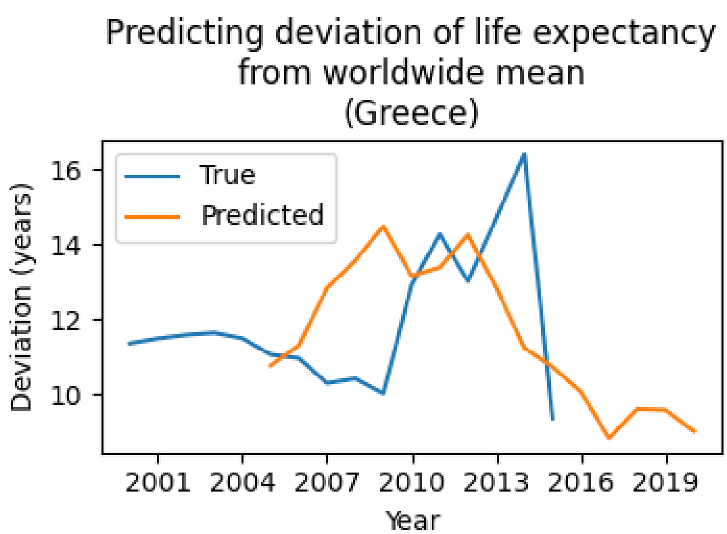
<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>



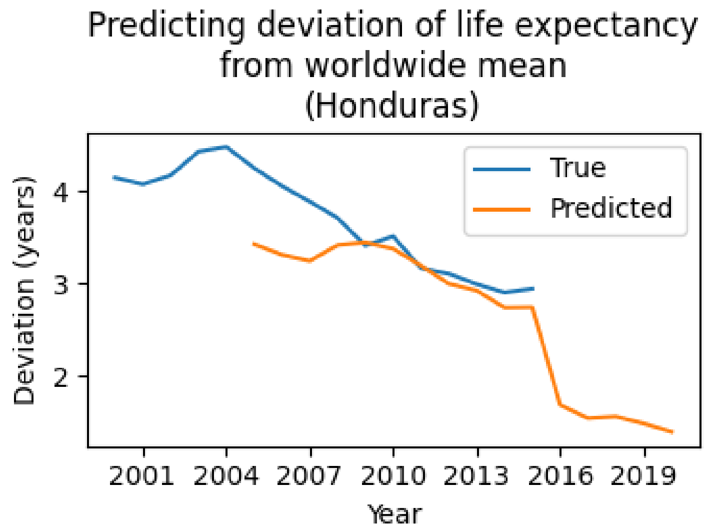
<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>



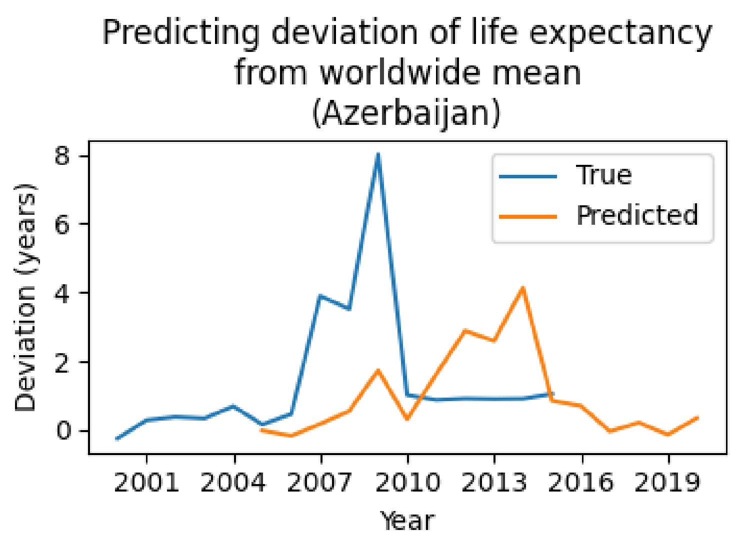
<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>



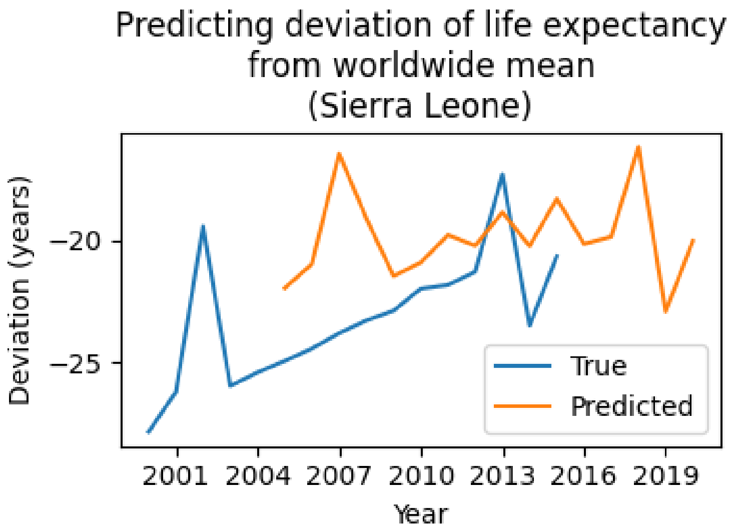

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.Javascript object>
```



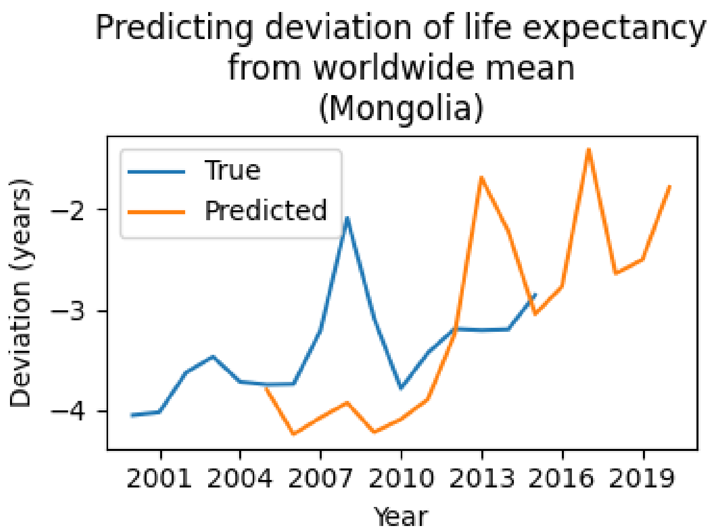

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.Javascript object>
```



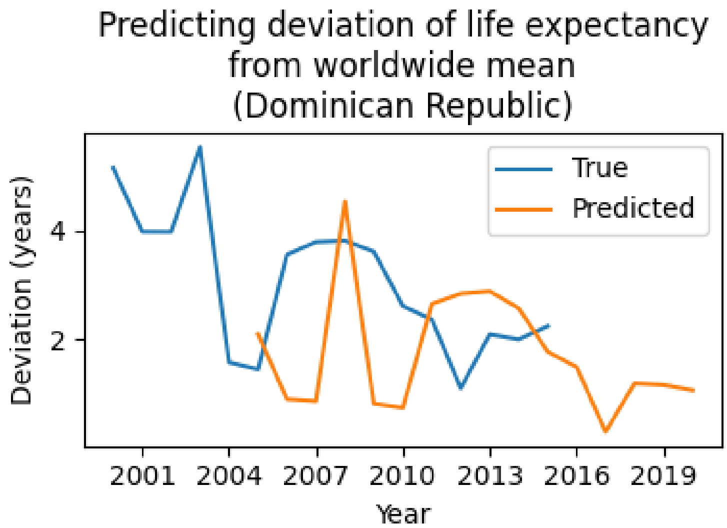
<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>



<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>



[ ]:
```
# Testing coefficient of correlation on test set
regressor.score(X_test, y_test)
```

[ ]: 0.29924327032546183

[ ]:
```
regressor_1 = regressor # Save for later
```

[ ]:
```
feature_importances = reversed(np.argsort(regressor.feature_importances_))
# Most important features
print('~~~ Feature importance, from most to least ~~~')
for place, i in enumerate(feature_importances):
    print(f'{place+1}.\t{X_train.columns[i]}')
```

```
~~~ Feature importance, from most to least ~~~
1.      Life expectancy
2.      HIV/AIDS
3.      Polio
4.      Income composition of resources
5.      Diphtheria
6.      GDP
7.      BMI
8.      Alcohol
9.      thinness 5-9 years
10.     Schooling
11.     Total expenditure
12.     thinness  1-19 years
13.     percentage expenditure
```

```
14.     Hepatitis B
15.     Measles
```

We suspect that the model is implicitly clustering countries.

2 What if we make the test set countries from a single cluster?

[ ]:
```python
# Features on which to cluster
X_features = [
    'Alcohol',
    'percentage expenditure',
    'Hepatitis B',
    'Measles',
    'BMI',
    'Polio',
    'Diphtheria',
    'HIV/AIDS',
    'Income composition of resources',
    'Schooling'
]

means_by_country = dataset.groupby('Country')[X_features].mean()
means_by_country.dropna(axis=1, inplace=True)
means_by_country.sample(5)
```

[ ]:
```
                                     Alcohol  percentage expenditure  \
Country
Uruguay                             6.272727              749.577108
Kiribati                            0.550000               74.633137
Guinea-Bissau                       2.754545               22.883904
Dominican Republic                  6.070000              262.198044
Venezuela (Bolivarian Republic of)  7.698182                0.000000

                                    Hepatitis B     Measles        BMI  \
Country
Uruguay                               94.090909    0.000000  48.400000
Kiribati                              69.272727    0.000000  66.145455
Guinea-Bissau                         70.412261  467.909091  16.909091
Dominican Republic                    71.090909   33.272727  44.200000
Venezuela (Bolivarian Republic of)    59.454545  239.909091  56.445455

                                        Polio  Diphtheria  HIV/AIDS  \
Country
Uruguay                             94.000000   86.545455  0.100000
Kiribati                            76.272727   68.909091  0.100000
Guinea-Bissau                       69.363636   55.454545  5.000000
Dominican Republic                  78.454545   82.090909  1.881818
```

```
Venezuela (Bolivarian Republic of)  72.363636   62.727273  0.100000

                                    Income composition of resources  Schooling
Country
Uruguay                                                     0.755727  15.109091
Kiribati                                                    0.262000  11.600000
Guinea-Bissau                                               0.180545   7.909091
Dominican Republic                                          0.673818  12.563636
Venezuela (Bolivarian Republic of)                          0.708727  12.154545
```

[ ]:
```
# Standardize the data
scaler = StandardScaler()
means_by_country_scaled = scaler.fit_transform(means_by_country)

# Perform PCA
n_components = len(means_by_country.columns)
pca = PCA(n_components=n_components)
pca_result = pca.fit_transform(means_by_country_scaled)

# Create a DataFrame for PCA results
pca_labels = [f'PC{i+1}' for i in range(n_components)]
pca_df = pd.DataFrame(
    pca_result,
    columns=pca_labels,
    index=means_by_country.index
)
```

[ ]:
```
# Perform KMeans clustering on the PCA results
n_clusters = 8
kmeans = KMeans(n_clusters=n_clusters)
pca_df['Cluster'] = kmeans.fit_predict(pca_df[pca_labels])

# Merge small clusters
min_cluster_size = 10
cluster_sizes = pca_df['Cluster'].value_counts()

# Step 3: Merge small clusters
for cluster, size in cluster_sizes.items():
    if size < min_cluster_size:
        # Find nearest cluster
        distances = np.linalg.norm(kmeans.cluster_centers_ - kmeans.
 ↪cluster_centers_[cluster], axis=1)
        nearest_cluster = np.argmin(distances[distances > 0])  # Exclude self␣
 ↪distance

        # Merge clusters
        pca_df.loc[pca_df['Cluster'] == cluster, 'Cluster'] = nearest_cluster
```

```
[ ]: # Renumber clusters to begin with 1 and be contiguous
pca_df['Cluster'] = pd.factorize(pca_df['Cluster'])[0] + 1
```

```
[ ]: pca_df['Cluster']
```

```
[ ]: Country
Afghanistan                         1
Albania                             2
Algeria                             3
Angola                              1
Antigua and Barbuda                 2
                                   ..
Venezuela (Bolivarian Republic of)  3
Viet Nam                            2
Yemen                               6
Zambia                              6
Zimbabwe                            6
Name: Cluster, Length: 179, dtype: int64
```

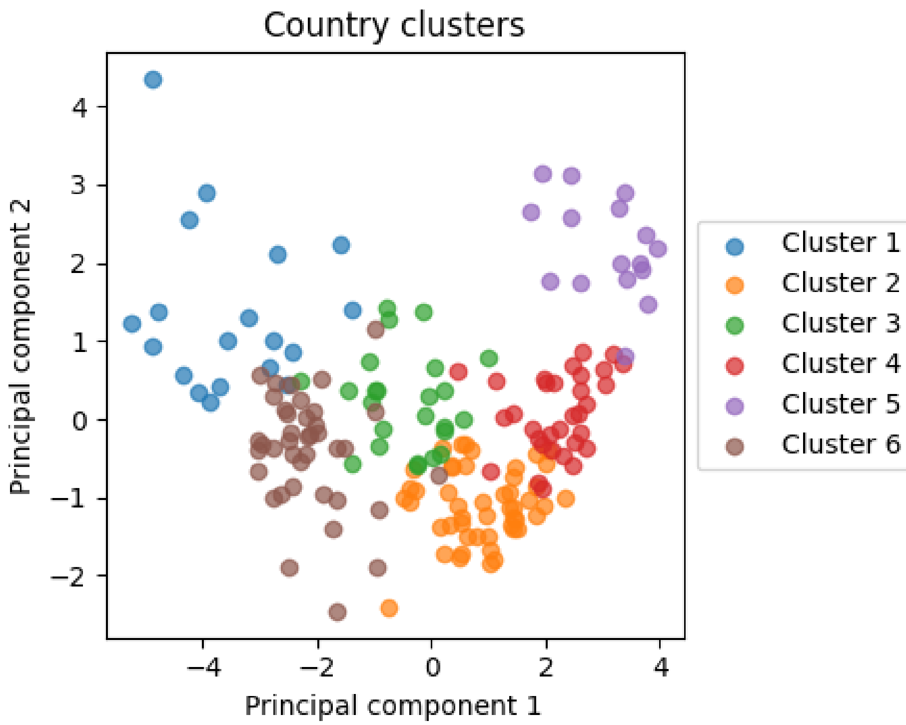
```
[ ]: pca_df['Cluster'].unique()
```

```
[ ]: array([1, 2, 3, 4, 5, 6])
```

```
[ ]: cluster_indices = sorted(pca_df['Cluster'].unique())
```

```
[ ]: # Visualize clusters in 2D
fig, ax = plt.subplots(figsize=(5, 4))

for cluster in cluster_indices:
    # Plot data
    cluster_data = pca_df[pca_df['Cluster'] == cluster]
    ax.scatter(cluster_data.iloc[:, 0], cluster_data.iloc[:, 1],␣
 ↪label=f'Cluster {cluster}', alpha=0.7)
ax.set_title(f'Country clusters')
ax.set_xlabel('Principal component 1')
ax.set_ylabel('Principal component 2')
ax.legend(loc='center left', bbox_to_anchor=(1, 0.5))
fig.tight_layout()

if SAVE_FIGS:
    fig.savefig('clusters.pdf')
    files.download('clusters.pdf')
plt.show()
```

```
<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>
```



[ ]:
```
# List countries in each cluster
for cluster in cluster_indices:
    print(f'~~~ Countries in Cluster {cluster} ~~~')
    # Fetch data
    cluster_data = pca_df[pca_df['Cluster'] == cluster]
    for country in cluster_data.index:
        print(country)
    print()
```

```
~~~ Countries in Cluster 2 ~~~
Afghanistan
Angola
Central African Republic
Chad
China
Congo
Democratic Republic of the Congo
Equatorial Guinea
Ethiopia
Gabon
Guinea
Haiti
India
Lao People's Democratic Republic
```

```
Liberia
Niger
Nigeria
Somalia
Uganda

~~~ Countries in Cluster 3 ~~~
Albania
Antigua and Barbuda
Armenia
Bahrain
Belize
Brunei Darussalam
Cabo Verde
Colombia
Cuba
Egypt
El Salvador
Fiji
Grenada
Guatemala
Guyana
Honduras
Iran (Islamic Republic of)
Israel
Jamaica
Jordan
Kuwait
Kyrgyzstan
Libya
Malaysia
Maldives
Mauritius
Mexico
Mongolia
Morocco
Nicaragua
Oman
Panama
Paraguay
Peru
Qatar
Saint Vincent and the Grenadines
Sao Tome and Principe
Saudi Arabia
Seychelles
Singapore
Sri Lanka
```

```
Tajikistan
Thailand
The former Yugoslav republic of Macedonia
Tunisia
Turkmenistan
United Arab Emirates
Uzbekistan
Viet Nam

~~~ Countries in Cluster 4 ~~~
Algeria
Azerbaijan
Bolivia (Plurinational State of)
Bosnia and Herzegovina
Costa Rica
Dominican Republic
Ecuador
Georgia
Iraq
Kiribati
Lebanon
Micronesia (Federated States of)
Montenegro
Philippines
Samoa
Solomon Islands
Suriname
Syrian Arab Republic
Tonga
Trinidad and Tobago
Turkey
Ukraine
Vanuatu
Venezuela (Bolivarian Republic of)

~~~ Countries in Cluster 5 ~~~
Argentina
Bahamas
Barbados
Belarus
Brazil
Bulgaria
Chile
Croatia
Cyprus
Czechia
Estonia
Finland
```

```
Greece
Hungary
Italy
Kazakhstan
Latvia
Lithuania
Malta
Poland
Portugal
Republic of Moldova
Romania
Russian Federation
Saint Lucia
Serbia
Slovakia
Slovenia
Spain
United Kingdom of Great Britain and Northern Ireland
United States of America
Uruguay

~~~ Countries in Cluster 6 ~~~
Australia
Austria
Belgium
Canada
Denmark
France
Germany
Iceland
Ireland
Japan
Luxembourg
Netherlands
New Zealand
Norway
Sweden
Switzerland

~~~ Countries in Cluster 7 ~~~
Bangladesh
Benin
Bhutan
Botswana
Burkina Faso
Burundi
Cambodia
Cameroon
```

```
Comoros
Côte d'Ivoire
Djibouti
Eritrea
Gambia
Ghana
Guinea-Bissau
Indonesia
Kenya
Lesotho
Madagascar
Malawi
Mali
Mauritania
Mozambique
Myanmar
Namibia
Nepal
Pakistan
Papua New Guinea
Rwanda
Senegal
Sierra Leone
South Africa
Swaziland
Timor-Leste
Togo
United Republic of Tanzania
Yemen
Zambia
Zimbabwe
```

[ ]:
```python
# Random number generator
gen = np.random.default_rng()

# Pick a random cluster
test_cluster = gen.choice(cluster_indices)

# Reserve countries from that cluster for test set
test_countries = pca_df[pca_df['Cluster'] == test_cluster].index
test_countries
```

[ ]:
```
Index(['Albania', 'Antigua and Barbuda', 'Armenia', 'Bahrain', 'Belize',
       'Brunei Darussalam', 'Cabo Verde', 'Colombia', 'Cuba', 'Egypt',
       'El Salvador', 'Fiji', 'Grenada', 'Guatemala', 'Guyana', 'Honduras',
       'Iran (Islamic Republic of)', 'Israel', 'Jamaica', 'Jordan', 'Kuwait',
```

```
       'Kyrgyzstan', 'Libya', 'Malaysia', 'Maldives', 'Mauritius', 'Mexico',
       'Mongolia', 'Morocco', 'Nicaragua', 'Oman', 'Panama', 'Paraguay',
       'Peru', 'Qatar', 'Saint Vincent and the Grenadines',
       'Sao Tome and Principe', 'Saudi Arabia', 'Seychelles', 'Singapore',
       'Sri Lanka', 'Tajikistan', 'Thailand',
       'The former Yugoslav republic of Macedonia', 'Tunisia', 'Turkmenistan',
       'United Arab Emirates', 'Uzbekistan', 'Viet Nam'],
      dtype='object', name='Country')
```

[ ]:
```
# Train-test split
data_test  = dataset[ dataset['Country'].isin(test_countries)]
data_train = dataset[~dataset['Country'].isin(test_countries)]
```

2.1 Machine learning analysis for clustered data

[ ]:
```
n_folds = 5
all_countries = dataset['Country'].unique()

# Reserve countries for test set
n_countries_test = int(0.15 * len(dataset['Country'].unique()))
test_countries = gen.choice(dataset['Country'], n_countries_test)

# Train-test split
data_test  = dataset[ dataset['Country'].isin(test_countries)]
data_train = dataset[~dataset['Country'].isin(test_countries)]

# Group-based splitter for cross-validation
group_kfold = GroupKFold(n_splits=n_folds)
groups = data_train['Country']

# Generate folds by country
folds = []
for train_idx, val_idx in group_kfold.split(data_train, groups=groups):
    train_data = dataset.iloc[train_idx]
    val_data = dataset.iloc[val_idx]
    folds.append((train_data, val_data))
```

[ ]:
```
def X_y_split(data):
    """ Split a dataset or a fold into X (features) and y (labels). """
    return data[X_features], data[y_feature_dev_change]

X_train, y_train = X_y_split(data_train)
X_test, y_test = X_y_split(data_test)
```

```
[ ]: optuna.logging.set_verbosity(optuna.logging.WARNING)
     n_trials = 50
```

```
[ ]: ## Random forest
     best_rf = None
     best_rf_score = -np.inf

     def objective(trial):
         global best_rf, best_rf_score

         # Select parameters. If log=True, prefer smaller numbers
         n_estimators = trial.suggest_int('n_estimators', 25, 500, log=True)
         max_depth = trial.suggest_int('max_depth', 2, 32)
         bootstrap = trial.suggest_categorical('bootstrap', [True, False])
         regressor = RandomForestRegressor(
             n_estimators=n_estimators,
             max_depth=max_depth,
             bootstrap=bootstrap,
             n_jobs=-1
         )

         # Train and evalate for each fold using sklearn utilities
         scores = cross_val_score(
             regressor,
             X_train,
             y_train,
             cv=group_kfold,
             groups=groups,
             scoring='r2'
         )
         score = np.mean(scores)

         # Save best model
         if score > best_rf_score:
             best_rf = regressor
             best_rf_score = score

         return score

     study_rf = optuna.create_study(direction='maximize')
     study_rf.optimize(objective, n_trials=n_trials, show_progress_bar=True,␣
      ↪n_jobs=-1)
```

```
  0%|          | 0/50 [00:00<?, ?it/s]
```

```
[ ]: print("Random Forest, Best Trial:")
     print(f"  Score: {study_rf.best_trial.value:.4f}")
```

```
print("  Parameters:")
for key, value in study_rf.best_trial.params.items():
    print(f"    {key}: {value}")
```

```
Random Forest, Best Trial:
  Score: 0.0686
  Parameters:
    n_estimators: 185
    max_depth: 2
    bootstrap: True
```

[ ]:
```
## XGBoost
best_xg = None
best_xg_score = -np.inf
def objective(trial):
    global best_xg, best_xg_score

    # Select parameters. If log=True, prefer smaller numbers
    n_estimators = trial.suggest_int('n_estimators', 25, 500, log=True)
    max_depth = trial.suggest_int('max_depth', 2, 32)
    learning_rate = trial.suggest_float('learning_rate', 0.01, 0.1)
    subsample = trial.suggest_float('subsample', 0.5, 1.0)
    colsample_bytree = trial.suggest_float('colsample_bytree', 0.5, 1.0)
    gamma = trial.suggest_float('gamma', 0, 10)

    regressor = XGBRegressor(
        n_estimators=n_estimators,
        max_depth=max_depth,
        learning_rate=learning_rate,
        subsample=subsample,
        colsample_bytree=colsample_bytree,
        gamma=gamma
    )

    # Train and evalate for each fold using sklearn utilities
    scores = cross_val_score(
        regressor,
        X_train,
        y_train,
        cv=group_kfold,
        groups=groups,
        scoring='r2'
    )

    score = np.mean(scores)

    # Save best model
```

```
    if score > best_xg_score:
        best_xg = regressor
        best_xg_score = score

    return score


study_xg = optuna.create_study(direction='maximize')
study_xg.optimize(objective, n_trials=n_trials, show_progress_bar=True,␣
 ↪n_jobs=-1)
```

```
  0%|          | 0/50 [00:00<?, ?it/s]
```

[ ]:
```
print("XGBoost, Best Trial:")
print(f"  Score: {study_xg.best_trial.value:.4f}")
print("  Parameters:")
for key, value in study_xg.best_trial.params.items():
    print(f"    {key}: {value}")
```

```
XGBoost, Best Trial:
  Score: 0.0736
  Parameters:
    n_estimators: 158
    max_depth: 2
    learning_rate: 0.03347635438978973
    subsample: 0.784084009373734
    colsample_bytree: 0.5404658977838952
    gamma: 5.69600994503886
```

[ ]:
```
## Gradient boosting regressor
best_gb = None
best_gb_score = -np.inf
def objective(trial):
    global best_gb, best_gb_score

    # Select parameters. If log=True, prefer smaller numbers
    n_estimators = trial.suggest_int('n_estimators', 25, 300, log=True)
    learning_rate = trial.suggest_float('learning_rate', 0.01, 1)
    subsample = trial.suggest_float('subsample', 0.5, 1)
    regressor = GradientBoostingRegressor(
        n_estimators=n_estimators,
        learning_rate=learning_rate,
        subsample=subsample
    )

    # Train and evalate for each fold using sklearn utilities
    scores = cross_val_score(
```

```
        regressor,
        X_train,
        y_train,
        cv=group_kfold,
        groups=groups,
        scoring='r2'
    )
    score = np.mean(scores)

    # Save best model
    if score > best_gb_score:
        best_gb = regressor
        best_gb_score = score

    return score


study_gb = optuna.create_study(direction='maximize')
study_gb.optimize(objective, n_trials=n_trials, show_progress_bar=True,␣
 ↪n_jobs=-1)
```

```
  0%|          | 0/50 [00:00<?, ?it/s]
```

[ ]:
```
print("Gradient Boosting, Best Trial:")
print(f"  Score: {study_gb.best_trial.value:.4f}")
print("  Parameters:")
for key, value in study_gb.best_trial.params.items():
    print(f"    {key}: {value}")
```

```
Gradient Boosting, Best Trial:
  Score: 0.0697
  Parameters:
    n_estimators: 75
    learning_rate: 0.017719964914993426
    subsample: 0.676705825175306
```

[ ]:
```
## Linear regression (ridge regression)
best_lr = None
best_lr_score = -np.inf

def objective(trial):
    global best_lr, best_lr_score

    # Select parameters. If log=True, prefer smaller numbers
    alpha = trial.suggest_float('alpha', 1e-5, 100.0, log=True)

    regressor = Ridge(alpha=alpha)
```

```
    # Train and evalate for each fold using sklearn utilities
    scores = cross_val_score(
        regressor,
        X_train,
        y_train,
        cv=group_kfold,
        groups=groups,
        scoring='r2'
    )
    score = np.mean(scores)

    # Save best model
    if score > best_lr_score:
        best_lr = regressor
        best_lr_score = score

    return score


study_lr = optuna.create_study(direction='maximize')
study_lr.optimize(objective, n_trials=n_trials, show_progress_bar=True,␣
  ↪n_jobs=-1)
```

```
  0%|          | 0/50 [00:00<?, ?it/s]
```

[ ]:
```
print("Linear Regression (Ridge), Best Trial:")
print(f"  Score: {study_lr.best_trial.value:.4f}")
print("  Parameters:")
for key, value in study_lr.best_trial.params.items():
    print(f"    {key}: {value}")
```

```
Linear Regression (Ridge), Best Trial:
  Score: 0.0706
  Parameters:
    alpha: 99.81574190801281
```

[ ]:
```
models = [best_rf, best_gb, best_xg, best_lr]
scores = [best_rf_score, best_gb_score, best_xg_score, best_lr_score]
best_index = np.argmax(scores)

print(f'Best score: {scores[best_index]}')
best_regressor = models[best_index]
```

```
Best score: 0.07364257864681734
```

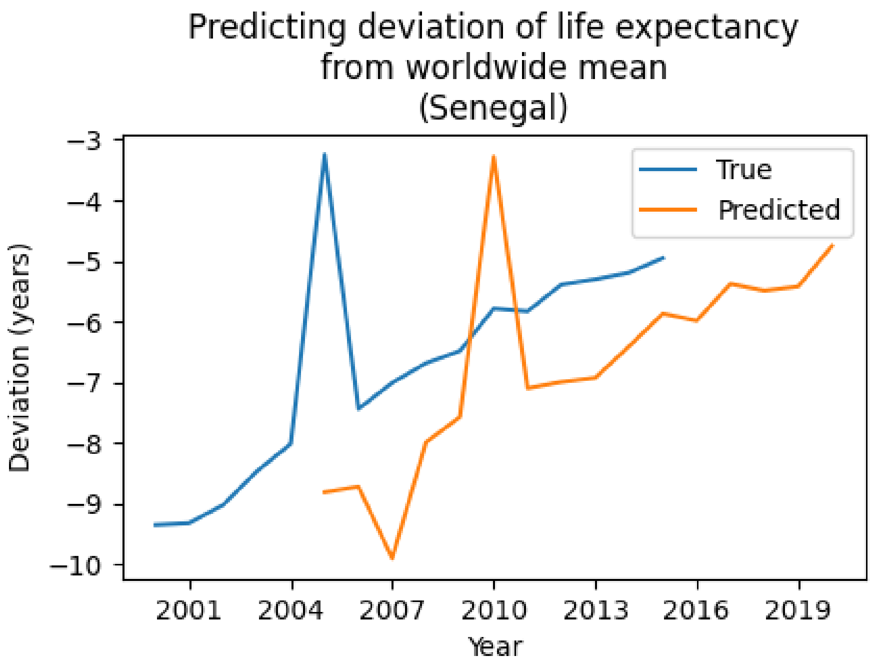
```
[ ]: regressor = best_regressor
     regressor.fit(X_train, y_train)
```

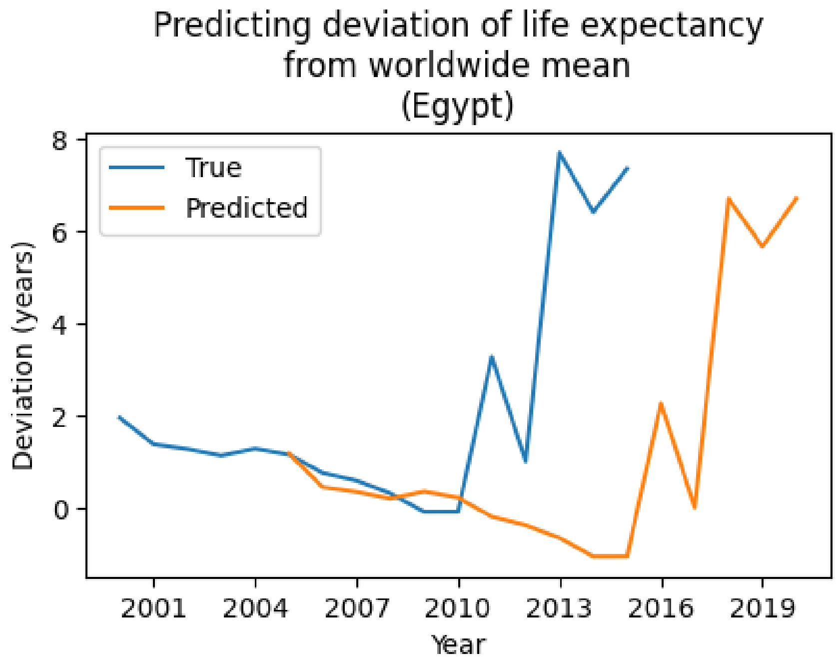
```
[ ]: XGBRegressor(base_score=None, booster=None, callbacks=None,
                  colsample_bylevel=None, colsample_bynode=None,
                  colsample_bytree=0.5404658977838952, device=None,
                  early_stopping_rounds=None, enable_categorical=False,
                  eval_metric=None, feature_types=None, gamma=5.69600994503886,
                  grow_policy=None, importance_type=None,
                  interaction_constraints=None, learning_rate=0.03347635438978973,
                  max_bin=None, max_cat_threshold=None, max_cat_to_onehot=None,
                  max_delta_step=None, max_depth=2, max_leaves=None,
                  min_child_weight=None, missing=nan, monotone_constraints=None,
                  multi_strategy=None, n_estimators=158, n_jobs=None,
                  num_parallel_tree=None, random_state=None, …)
```

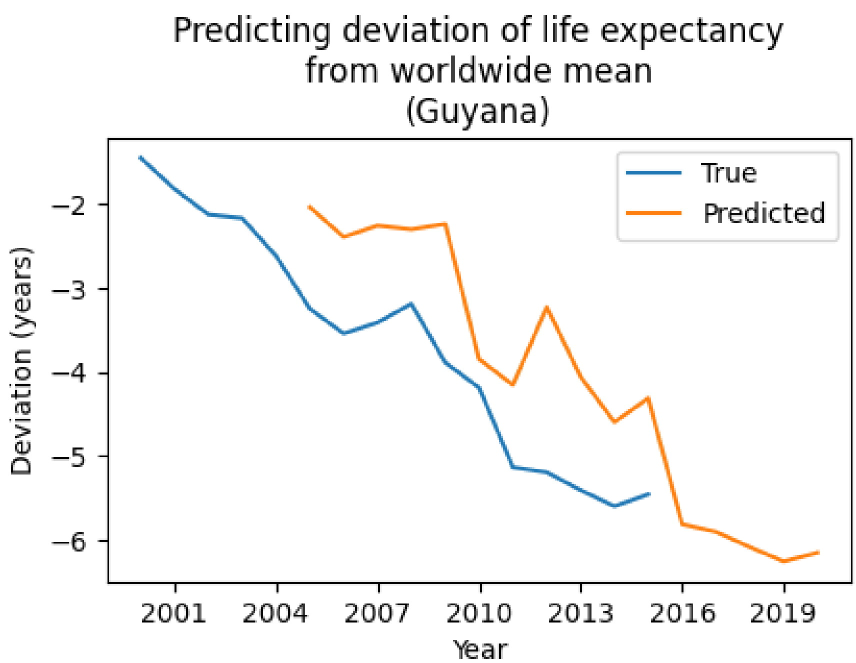
Let's visualize.

```
[ ]: # Randomly pick some countries from the test set
     n_countries = 3
     countries = gen.choice(test_countries, n_countries, replace=False)
     # Plot for each the predictions vs the truth
     for country in countries:
         country_data = dataset_full[dataset_full['Country'] == country]

         plt.figure(figsize=(5, 3))
         plt.title(f'Predicting deviation of {y_feature.lower()}\nfrom worldwide␣
      ↪mean\n({country})')
         plt.xlabel('Year')
         plt.ylabel('Deviation (years)')

         plt.plot(
             country_data['Year'],
             country_data[y_feature_dev],
             label=f'True'
         )
         # The following may seem confusing. I'll explain.
         # The model predicts change in life expectancy.
         plt.plot(
             [y + n for y in country_data['Year']],
             [dev + delta_dev
                 for dev, delta_dev
                 in zip(
                     country_data[y_feature_dev],
                     regressor.predict(country_data[X_features])
                     )
             ],
```

```
        label=f'Predicted'
    )

    plt.legend()
    # Make year ticks on the bottom integers
    plt.gca().xaxis.set_major_locator(MaxNLocator(integer=True))
    plt.show()
    print()
```




Predicting deviation of life expectancy
from worldwide mean
(Egypt)
True
Predicted
Deviation (years)
Year
8
6
4
2
0
2001
2004
2007
2010
2013
2016
2019


Predicting deviation of life expectancy
from worldwide mean
(Guyana)
True
Predicted
Deviation (years)
Year
−2
−3
−4
−5
−6
2001
2004
2007
2010
2013
2016
2019