Predicting life expectancy by economic and health factors

CS-C3240 Machine Learning D - Project Stage 2

1. Introduction

In this project, our goal is to predict average life expectancy by country, based on economic and health factors. These factors are often studied separately when assessing their impact on longevity, which is why combining them might provide useful unseen insight.

For instance, recent research shed light on health markers such as VO2max [1] and grip strength [2] being key predictors for longevity. On the economics side, research suggests education and wealth as being influential in average life expectancy [3][4][5]. Therefore, our aim is to take both health and economic factors into account, weigh them according to their importance, and predict average life expectancy by country. Knowing which factors contribute the most to life expectancy and being able to make predictions is of great importance in our domain of application, which is economic policy decision-making and resource allocation for governments. Therefore, external datasets are chosen to highlight the effects of different variables.

The report first presents the problem formulation in section 2, then in section 3 we discuss the methods used, followed by our results in section 4, and our conclusions in section 5. Section 6 includes our references, while section 7 is the appendix including all code used and the accompanying figures.

2. Problem formulation

The goal is to predict average life expectancy at the country level, so we need to consider reliable data that is available from a reasonable number of countries. This is why we selected a dataset from the World Health Organization (WHO) available on Kaggle [6], which tracked health and economical markers for 193 countries between 2000 and 2015.

For comparison purposes we also included datasets from Eurostat [7] containing data on alcohol consumption, education level, BMI, sex, and physical activity levels, which were easiest to combine. The Eurostat datasets needed thorough cleaning and indexing, and a python dictionary was made to handle variable acronyms.

In WHO dataset, each row of our final dataset represents a country, and each column a variable. There is a total of 20+ variables, which are averages by country, including (but not limited to): life expectancy, alcohol consumption, infant deaths, status (developing or developed nation), number of schooling years, gross domestic product (GDP), HIV/AIDS rates, and others. These are all numerical variables, apart from a select few like status variables, which were converted to numerical in both datasets.

The variable of interest (or label) that we want to predict is thus average life expectancy expressed in years, and we selected the following numerical features to predict it through a polynomial regression: alcohol consumption, years in education, HIV/AIDS rates, income composition of resources (ICOR), under-five deaths, adult mortality, BMI, GDP per capita in WHO dataset respectively, and other in Eurostat. These features were selected after visualizing the data through a matrix of correlations and a random forest test.

To run our linear and polynomial regressions, we aggregate the years taking median values in the WHO dataset, and we divide our set into a training set (80% of countries), a testing set (10% of countries), and a validation set (10% of countries) for both datasets. In addition, we also experimented with a 60/20/20 split on the WHO dataset. In both cases, the validation set is completely external to the training and testing sets, preventing overlapping, because there is no relation between countries. These are also commonly used splits for this type of supervised machine learning task, and we have enough data points to support it (193 countries). We're also going to use k-fold cross-validation during training. This trains the model on k-1 equal

parts (folds) of the dataset and tests it on the remaining folds k times, then the performance of all k test results is averaged out. This is done with the aim of preventing overfitting and improving the model robustness. Moreover, it allows us to have a reference to assess the performance and validation metrics of our own split.

3. Methods

3.1. Model selection

For our model, it was decided to use polynomial regression as our first method. This is because we wanted to avoid the underfitting problems that might have occurred with a linear regressor, and because we believe certain features like wealth (GDP per capita) or number of school years might have diminishing returns in increasing life expectancy. The idea is that, for example, small improvements in wealth at low levels might improve life expectancy by a lot, but only by a little if we're already at high levels of wealth. A polynomial model allows for these types of relations to be accounted for, because the curve would be non-linear.

Our second method is linear regression with ridge regularization. The goal of this choice is to have a way to test the assumptions that were made with polynomial regression. For instance, it's possible that a polynomial regression might actually be too complex and might overfit, and that there are no non-linear relations, or perhaps not to the extent we assumed. Having a linear regression as a second method allows to choose the most appropriate option and spot instances of underfitting or overfitting more easily.

To assess feature importance, the random forest method is used. This is an ensemble learning technique that builds multiple decision trees and averages their outputs for prediction accuracy. By doing so, it tells us which features influenced life expectancy the most, which is essential information in policy decision-making and resource allocation.

3.2. Regularization

Going with a regression model makes us face a variance-bias tradeoff: high bias is indicative of an underfitting (or too simplistic) model, and high variance is indicative of overfitting (when a model learns the training data noise rather than the underlying pattern). Ideally, we want to aim for low bias and low variance, but there's always a tradeoff between them.

Choosing a polynomial regression helps prevent underfitting, but we need to take steps to mitigate overfitting, which is something that can easily happen.

Moreover, it is important for the features to be independent, but some of our features might be correlated like wealth and schooling years. These concerns were addressed by using regularization. It consists in discouraging large coefficients by adding a penalty term to the loss function.

There are two main types of regularization: Lasso and Ridge. Lasso adds a penalty equal to the absolute value of coefficients to the loss function, which can set some coefficients to zero. On the other hand, Ridge adds a penalty equal to the square value of the coefficients, so it can minimize the impact of some less important features.

The ridge regularization was chosen, because it never sets the coefficients to exactly zero, and because it can help set appropriate weights to some of our features that might be slightly correlated (like schooling and wealth). Also, Lasso might remove some of our features, and it is believed all features have a degree of importance and should at best be minimized, but not removed. Therefore, Ridge is believed to be the best approach.

3.3. Loss function

The loss function measures how well the model predicts the label. In other words, how far off the predicted value is from the real value (hence why we have a supervising set). The goal is to minimize this function. The mean square error (MSE) was chosen because, thanks to the square element, it penalizes larger errors more than smaller ones. Moreover, MSE is convex finding its global minimum can be done efficiently with a gradient technique. It's also a widely used loss function in this context. Other options included mean absolute error (MAE) and Huber loss. They were not chosen because MAE doesn't penalize large errors as well as MSE, and Huber loss is more appropriate when dealing with outlier data points, and since the WHO is a reliable source, there is not a problem with outlier points, but Eurostat combined dataset might be. Therefore, MSE was the most appropriate choice for us.

The loss function (MSE + Ridge penalty) is formalized as: $\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda\sum_{j=1}^{p}\Theta_j{}^2$

The terms y_i and $\hat{y}_i$ are the actual and predicted values respectively, n the number of points, λ the parameter controlling the strength of the regularization, and Θ_j the weights of the features.

4. Results

```
Ridge Regression Avg MSE (k=5): 6.284689072645273
Polynomial Ridge Regression (degree=2) Avg MSE (k=5): 77.54516635509388
Ridge Regression Validation MSE: 4.556204431078149
Ridge Regression Test MSE: 10.414653186126133
Ridge Regression Test R-squared: 0.855523097554042
Polynomial Ridge Regression Validation MSE: 208.15754701805494
Polynomial Ridge Regression Test MSE: 5.930109869029756
Polynomial Ridge Regression Test R-squared: 0.9177347637285741
```

First, with a 60/20/20 split, our polynomial regression of degree 2 with ridge regularization gives us a test MSE of 5.93 and a validation MSE of 208.15. That is a low test MSE, which indicates good performance, but the very high validation MSE suggests strong overfitting. Our k-fold test also supports this conclusion, as the average MSE for the polynomial regression is very high at 77.54. We tested this with higher degrees and the validation MSE was even higher.

On the other hand, our ridge linear regression performs better. We have a validation MSE of 4.55, a test MSE of 10.41, and a k-fold average MSE of 6.28. We notice that the validation of MSE is smaller compared to the polynomial, this indicates that the linear model performs better on unseen data. Moreover, our linear model has an R2 value of 0.85 which means the model can explain 85% of the variance in life expectancy. This is quite good, considering that the polynomial could explain 91%, all while being extremely overfitting.

The EUROSTAT and WHO datasets were also compared between each other without the k folding due to inconsistent dimensions in years and variables. As can be seen from the figures included in the appendixes, the polynomial regression overfitted when dimension was increased over 5 and underfit when dimension was below 2. There was a division between training, testing and validation between countries with 80%,10%,10%. In EUROSTAT, the year 2019 data was used to as validation. The ridge and linear model were compared between the real statistical data. The figures can be seen in the appendixes and errors below table. Polynomial regression is excluded because of huge errors which can be observed in figures.

Dataset model \ error type	Training error	Testing error	Validation error
Eurostat dataset linear	3.36580653e-28	1.24251739e-1	39.24808579
Who dataset linear	9.65805957	6.54076359e0	1.23212798e1
Eurostat dataset ridge	0.0025334816222466945	0.031547048313643015	0.031547048313643015
Who dataset ridge	11.435299253389392	6.4836394156240305	6.4836394156240305

Therefore, the final chosen method is linear regression with ridge in both comparisons.

5. Conclusion

Our assumptions about the non-linearity of the problem revealed to be false, as the linear model performed the most accurately in both datasets, as shown by the lower validation MSE. Interestingly, according to our random forest test, ICOR (income composition of resources) is the most important feature (after the obvious adult mortality) with a score of 0.3, nearly 3 times higher than the next most important feature (HIV/AIDS rates) at 0.09. ICOR is an index indicating how efficient a country is at utilizing their resources and shows how equally the incomes are distributed in the population. A high ICOR is correlated with high life expectancy. This suggests that to improve life expectancy, a government should prioritize addressing income inequalities. However, our model still has room for improvement. For instance, the linear model contains errors which could be excluded. An error in measurement in data also affects quite heavily the polynomial regression, which the linear model outperforms. Therefore, a model with more robust adjustment to measurement error would be suitable.

It could also be useful in future research to make use of a more developed random forest method to compare the results with our regressions, rather than only using it for feature importance.

Nonetheless, we still find the average loss of the linear model satisfying, as we have a value of 6.28, against 77.5 for the polynomial. This tells us that the linear model has considerably better accuracy, that it's less prone to overfitting, and that it generalizes to unseen data quite reasonably.

The supporting figures and plots are provided as evidence in the appendixes.

6. References

[1] Strasser, Barbara, and Martin Burtscher. "Survival of the fittest: VO2max, a key predictor of longevity." Front Biosci (Landmark Ed) 23.8 (2018): 1505-1516.

[2] Oksuzyan A, Demakakos P, Shkolnikova M, Thinggaard M, Vaupel JW, et al. (2017) Handgrip strength and its prognostic value for mortality in Moscow, Denmark, and England. PLOS ONE 12(9): e0182684. https://doi.org/10.1371/journal.pone.0182684

[3] Chetty R, Stepner M, Abraham S, et al. The Association Between Income and Life Expectancy in the United States, 2001-2014. JAMA. 2016;315(16):1750–1766. doi:10.1001/jama.2016.4226

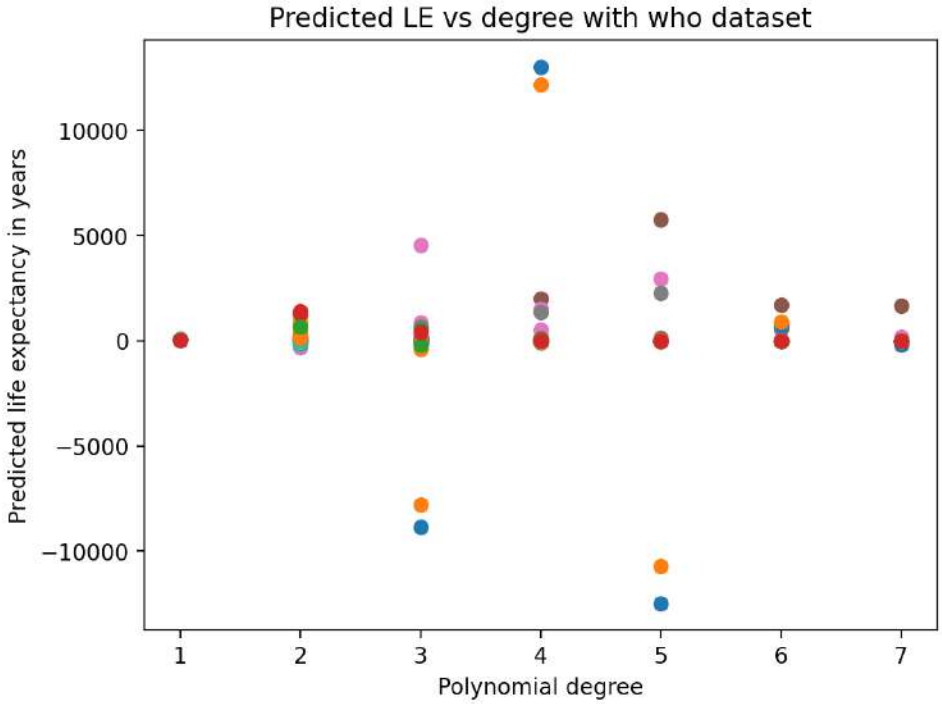
[4] Feng Ji, "Analysis of life expectancy," Proc. SPIE 12163, International Conference on Statistics, Applied Mathematics, and Computing Science (CSAMCS 2021), 121630I (22 April 2022); https://doi.org/10.1117/12.2628054

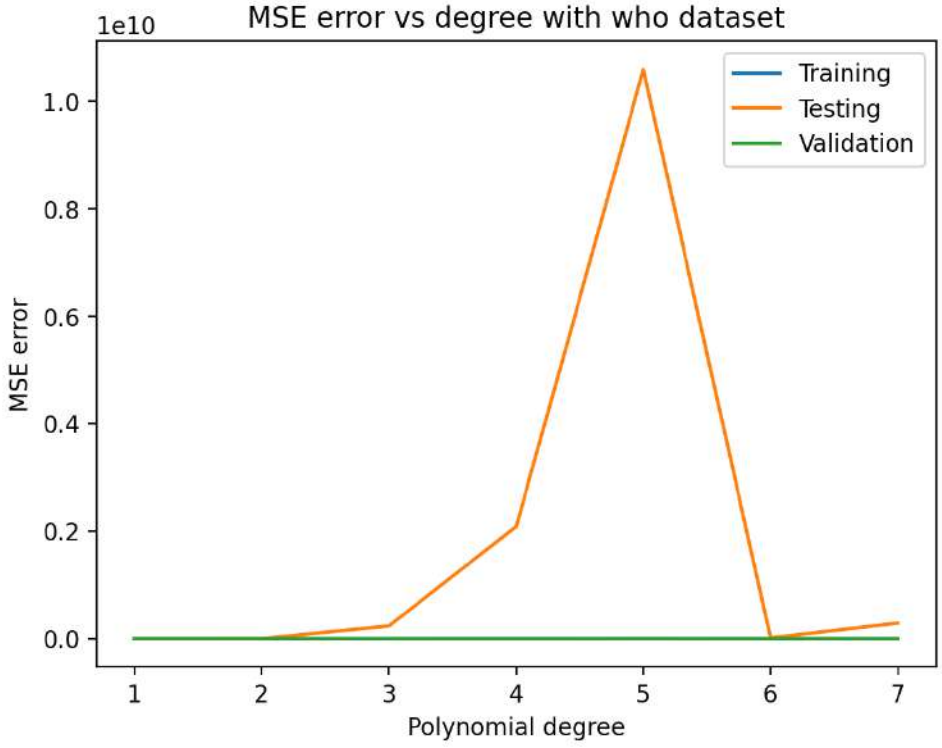
[5] Y. Wang, "The Greatest Factors Affecting Life Expectancy: A Research based on Different Continents and Countries," 2021 3rd International Conference on Machine Learning, Big Data and Business Intelligence (MLBDBI), Taiyuan, China, 2021, pp. 531-541, doi: 10.1109/MLBDBI54094.2021.00107.

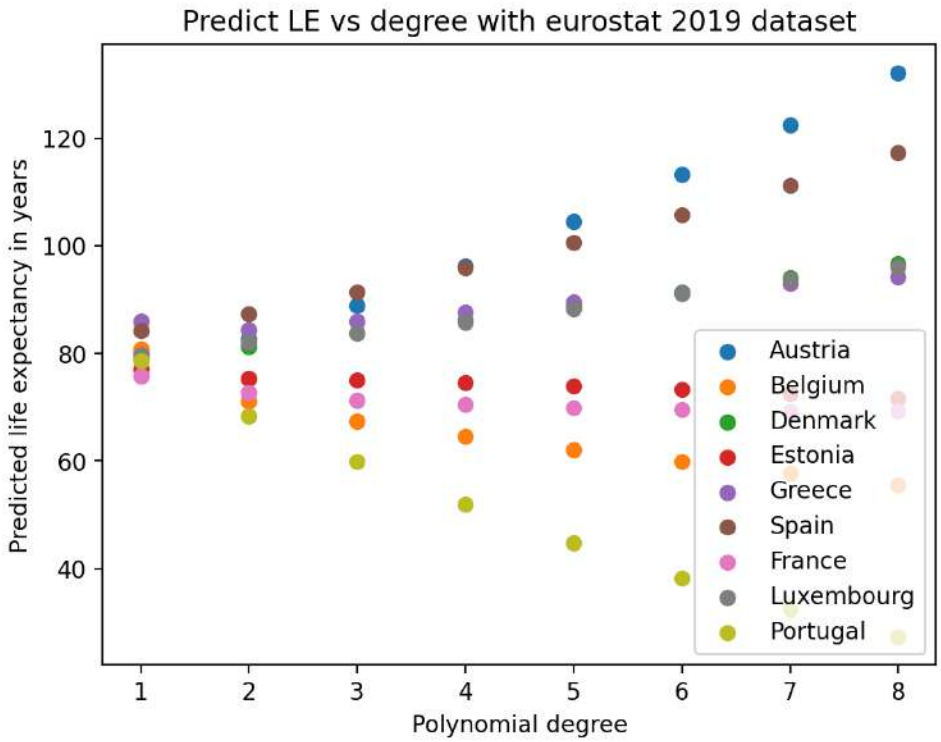
[6] Life expectancy dataset (WHO), Kaggle: https://www.kaggle.com/datasets/kumarajarshi/life-expectancy-who

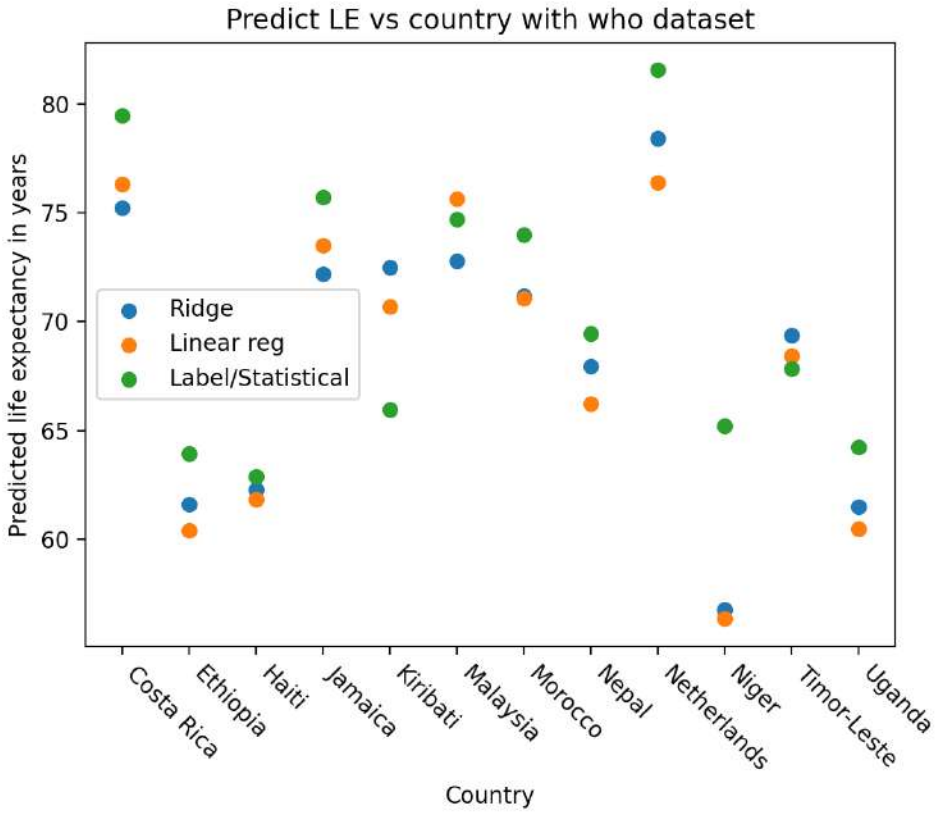
[7] Eurostat datasets used for comparison (health determinants folder):
https://ec.europa.eu/eurostat/databrowser/explore/all/popul?lang=en&subtheme=hlth.hlth_det&display=list&sort=category

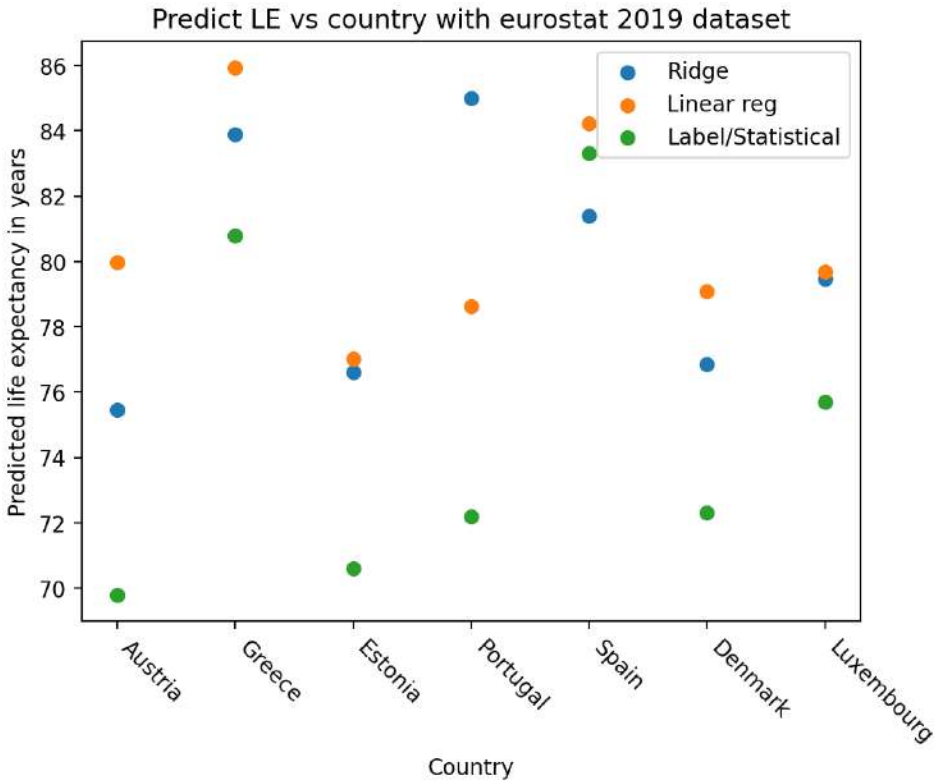
7.1. Appendix - Figures

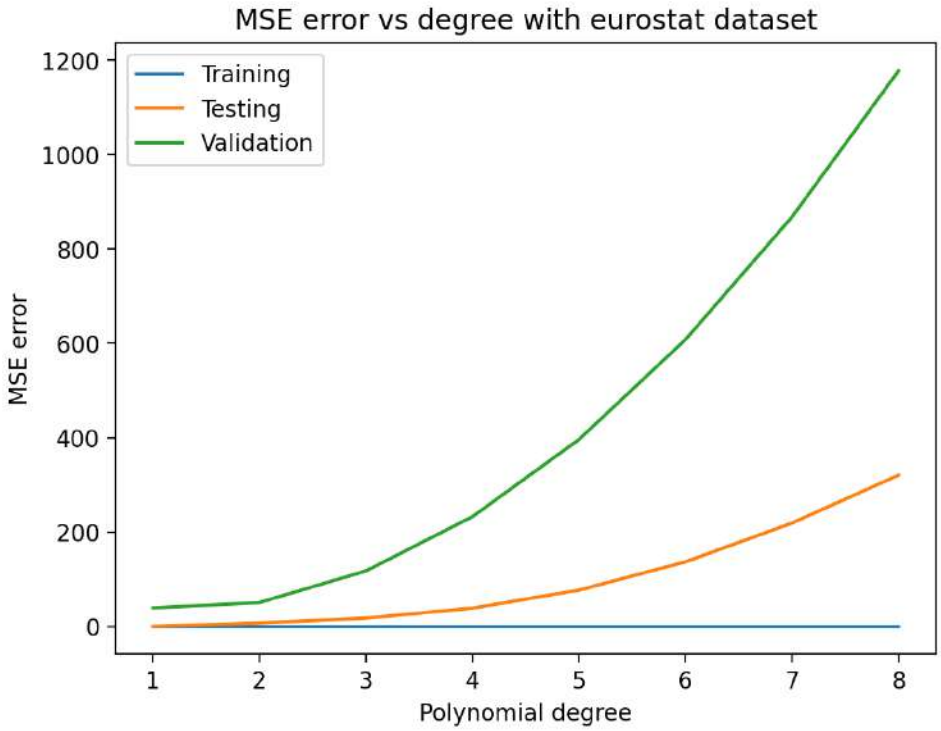





Predict LE vs degree with eurostat 2019 dataset
Predicted life expectancy in years
Polynomial degree
Austria
Belgium
Denmark
Estonia
Greece
Spain
France
Luxembourg
Portugal


Predict LE vs country with who dataset
Predicted life expectancy in years
Country
Ridge
Linear reg
Label/Statistical
Costa Rica
Ethiopia
Haiti
Jamaica
Kiribati
Malaysia
Morocco
Nepal
Netherlands
Niger
Timor-Leste
Uganda





7.2. Appendix - Notebook codes

project_notebook

October 9, 2024

[1]:
```
# import the important libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
# plotting for heatmaps of missing data
import seaborn as sns

# This python file contains the acronyms for the EUROSTAT acronyms
# File contains every eurostat quantity used in the project and corresponding␣
 ↪dictionary
# with explanations.
# Example quantity_exp[quantity_acronym]
import acronyms
```

[2]:
```
# The following block contains constant that are used for cleaning EUROSTAT data

replace_values = [':',': ',': z',': u',': bu']
filter_values_countries = [
    'Euro area - 18 countries (2014)',
    'European Union (EU6-1958, EU9-1973, EU10-1981, EU12-1986, EU15-1995,␣
 ↪EU25-2004, EU27-2007, EU28-2013, EU27-2020)',
    'European Union - 27 countries (2007-2013)',
    'European Union - 27 countries (from 2020)',
    'European Union - 28 countries (2013-2020)',
    'European Union - 15 countries (1995-2004)',
    'Euro area (EA11-1999, EA12-2001, EA13-2007, EA15-2008, EA16-2009,␣
 ↪EA17-2011, EA18-2014, EA19-2015, EA20-2023)',
    'Euro area - 20 countries (from 2023)',
    'Euro area - 19 countries  (2015-2022)',
    'Euro area - 12 countries (2001-2006)'
]

create_labels = lambda x,y=0 : dict(zip(list(x.
 ↪values())+[-1],range(y,len(x)+1+y)))
colum_values = {
    'BMI description' : create_labels(acronyms.bmi_exp),
    'Drinking freq' : create_labels(acronyms.frequenc_exp),
```

```
    'Smoking levels' : create_labels(acronyms.smoking_exp),
    'Exercise level' : create_labels(acronyms.physact_exp),
    'Education level' : create_labels(acronyms.isced11_exp),
    'Sex' : create_labels(acronyms.sex_exp),
}

dataset_paths = {
    "Healthy life years by sex" : "./data/estat_hlth_hlye.tsv.gz",
    "Body mass index (BMI) by sex, age and educational attainment level" : "./
 ↪data/estat_hlth_ehis_bm1e.tsv.gz",
    "Daily smokers of cigarettes by sex, age and educational attainment level" ␣
 ↪: "./data/estat_hlth_ehis_sk3e.tsv.gz",
    "Frequency of heavy episodic drinking by sex, age and educational␣
 ↪attainment level" : "./data/estat_hlth_ehis_al3e.tsv.gz",
    "Performing (non-work-related) physical activities by sex, age and␣
 ↪educational attainment level" : "./data/estat_hlth_ehis_pe3e.tsv.gz",
}
```

[3]:
```
def read_bmi_dataset():
    # returns pandas dataset of bmi based on education
    # contains frequency,unit, bmi, education level according to standard, sex,␣
 ↪age, country and time year 2014 or 2019
    # https://ec.europa.eu/eurostat/databrowser/view/hlth_ehis_bm1e/default/
 ↪table?lang=en
    df = pd.read_csv(dataset_paths["Body mass index (BMI) by sex, age and␣
 ↪educational attainment level" ],sep="\t",compression='gzip')
    df['Country'] = df[df.columns[0]].apply(lambda x : acronyms.geo_exp[x.
 ↪split(',')[-1]])
    df['Time Frequency'] = df[df.columns[0]].apply(lambda x : acronyms.
 ↪freq_exp[x.split(',')[0]])
    df['Unit of measure'] = df[df.columns[0]].apply(lambda x : acronyms.
 ↪unit_exp[x.split(',')[1]])
    df['BMI description'] = df[df.columns[0]].apply(lambda x : acronyms.
 ↪bmi_exp[x.split(',')[2]])
    df['Education level'] = df[df.columns[0]].apply(lambda x : acronyms.
 ↪isced11_exp[x.split(',')[3]])
    df['Sex'] = df[df.columns[0]].apply(lambda x : acronyms.sex_exp[x.
 ↪split(',')[4]])
    df['Age class'] = df[df.columns[0]].apply(lambda x : acronyms.age_exp[x.
 ↪split(',')[5]])

    df = df.replace(to_replace=replace_values,value=np.nan)

    df['2014 '] = df['2014 '].apply(lambda x : float(x.split(' ')[0]) if␣
 ↪type(x) == type(str()) else x)
```

```
    df['2019 '] = df['2019 '].apply(lambda x : float(x.split(' ')[0]) if␣
 ↪type(x) == type(str()) else x)
    drop_rows = df.loc[
        (df['Country'].isin(filter_values_countries)) | # filter only countries
        (df['Age class'] != 'Total') |
        (df['Sex'] != 'Total') |
        (df['Education level'] != 'All ISCED 2011 levels')
    ]
    df = df.drop(drop_rows.index)
    df = df.drop(columns=[df.columns[0],'Age class','Time Frequency','Unit of␣
 ↪measure','Sex','Education level'])
    df = df.rename(columns={"2014 " : f"2014_bmi_percent" , "2019 " :␣
 ↪f"2019_bmi_percent"})
    df = df.reset_index(drop=True)
    df = df.set_index(['Country'])
    return df
#print(read_bmi_dataset())
```

[4]:
```
def read_smoking_dataset():
    # returns pandas dataset of bmi based on education
    # contains frequency,unit, smoking, education level according to standard,␣
 ↪sex, age, country and time year 2014 or 2019
    # https://ec.europa.eu/eurostat/databrowser/view/hlth_ehis_sk3e/default/
 ↪table?lang=en
    df = pd.read_csv(dataset_paths["Daily smokers of cigarettes by sex, age and␣
 ↪educational attainment level"],sep="\t",compression='gzip')
    df['Time Frequency'] = df[df.columns[0]].apply(lambda x : acronyms.
 ↪freq_exp[x.split(',')[0]])
    df['Country'] = df[df.columns[0]].apply(lambda x : acronyms.geo_exp[x.
 ↪split(',')[-1]])
    df['Smoking levels'] = df[df.columns[0]].apply(lambda x : acronyms.
 ↪smoking_exp[x.split(',')[2]])
    df['Education level'] = df[df.columns[0]].apply(lambda x : acronyms.
 ↪isced11_exp[x.split(',')[3]])
    df['Unit of measure'] = df[df.columns[0]].apply(lambda x : acronyms.
 ↪unit_exp[x.split(',')[1]])
    df['Sex'] = df[df.columns[0]].apply(lambda x : acronyms.sex_exp[x.
 ↪split(',')[4]])
    df['Age class'] = df[df.columns[0]].apply(lambda x : acronyms.age_exp[x.
 ↪split(',')[5]])

    df = df.replace(to_replace=replace_values,value=np.nan)
    # fix some data to from Number Char type to numbers
    df['2014 '] = df['2014 '].apply(lambda x : float(x.split(' ')[0]) if␣
 ↪type(x) == type(str()) else x)
```

```
    df['2019 '] = df['2019 '].apply(lambda x : float(x.split(' ')[0]) if␣
↪type(x) == type(str()) else x)

    drop_rows = df.loc[
        (df['Country'].isin(filter_values_countries)) | # filter only countries
        (df['Age class'] != 'Total') | # the age doesn't mater
        (df['Sex'] != 'Total') |
        (df['Smoking levels'] != 'Total')|
        (df['Education level'] != 'All ISCED 2011 levels')
    ]
    drop_cols = df.columns.drop(['2014 ','2019 ','Country'])
    df = df.drop(index=drop_rows.index,columns=drop_cols)
    df = df.rename(columns={"2014 " : f"2014_smoking_percent" , "2019 " :␣
↪f"2019_smoking_percent"})
    df = df.reset_index(drop=True)
    df = df.set_index(['Country'])
    return df

#print(read_smoking_dataset())
```

[5]:
```
def read_alcohol_dataset():
    # Frequency of heavy episodic drinking by sex, age and educational␣
↪attainment level
    # https://ec.europa.eu/eurostat/databrowser/view/hlth_ehis_al3e/default/
↪table?lang=en
    df = pd.read_csv(dataset_paths["Frequency of heavy episodic drinking by␣
↪sex, age and educational attainment level"],sep="\t",compression='gzip')
    df['Time Frequency'] = df[df.columns[0]].apply(lambda x : acronyms.
↪freq_exp[x.split(',')[0]])
    df['Country'] = df[df.columns[0]].apply(lambda x : acronyms.geo_exp[x.
↪split(',')[-1]])
    df['Unit of measure'] = df[df.columns[0]].apply(lambda x : acronyms.
↪unit_exp[x.split(',')[1]])
    df['Sex'] = df[df.columns[0]].apply(lambda x : acronyms.sex_exp[x.
↪split(',')[4]])
    df['Drinking freq'] = df[df.columns[0]].apply(lambda x : acronyms.
↪frequenc_exp[x.split(',')[2]])
    df['Education level'] = df[df.columns[0]].apply(lambda x : acronyms.
↪isced11_exp[x.split(',')[3]])
    df['Age class'] = df[df.columns[0]].apply(lambda x : acronyms.age_exp[x.
↪split(',')[5]])

    drop_rows = df.loc[
        (df['Country'].isin(filter_values_countries)) | # filter only countries
        (df['Age class'] != 'Total') | # the age doesn't mater
        (df['Sex'] != 'Total') | # the expectancion depends on sex
```

```
        (df['Education level'] != 'All ISCED 2011 levels') # there is assumed␣
↪to be correlation between healthy lifehabits and education
    ]
    drop_cols = df.columns.drop(['2014 ','2019 ','Country','Drinking freq'])
    df = df.drop(index=drop_rows.index,columns=drop_cols)
    df = df.replace(to_replace=replace_values,value=np.nan)
    # fix some data to from Number Char type to numbers
    df['2014 '] = df['2014 '].apply(lambda x : float(x.split(' ')[0]) if␣
↪type(x) == type(str()) else x)
    df['2019 '] = df['2019 '].apply(lambda x : float(x.split(' ')[0]) if␣
↪type(x) == type(str()) else x)
    df = df.rename(columns={"2014 " : f"2014_alcohol_percent" , "2019 " :␣
↪f"2019_alcohol_percent"})
    df = df.reset_index(drop=True)
    df = df.set_index(['Country'])
    return df
#print(read_alcohol_dataset())
```

[6]:
```
def read_exercise_dataset():
    #freq,unit,physact,isced11,sex,age,geo\TIME_PERIOD
    # https://ec.europa.eu/eurostat/databrowser/view/hlth_ehis_pe3e/default/
↪table?lang=en
    df = pd.read_csv(dataset_paths["Performing (non-work-related) physical␣
↪activities by sex, age and educational attainment␣
↪level"],sep="\t",compression='gzip')
    df['Time Frequency'] = df[df.columns[0]].apply(lambda x : acronyms.
↪freq_exp[x.split(',')[0]])
    df['Country'] = df[df.columns[0]].apply(lambda x : acronyms.geo_exp[x.
↪split(',')[-1]])
    df['Unit of measure'] = df[df.columns[0]].apply(lambda x : acronyms.
↪unit_exp[x.split(',')[1]])
    df['Education level'] = df[df.columns[0]].apply(lambda x : acronyms.
↪isced11_exp[x.split(',')[3]])
    df['Exercise level'] = df[df.columns[0]].apply(lambda x : acronyms.
↪physact_exp[x.split(',')[2]])
    df['Sex'] = df[df.columns[0]].apply(lambda x : acronyms.sex_exp[x.
↪split(',')[4]])
    df['Age class'] = df[df.columns[0]].apply(lambda x : acronyms.age_exp[x.
↪split(',')[5]])

    df = df.replace(to_replace=replace_values,value=np.nan)
    # fix some data to from Number Char type to numbers
    df['2014 '] = df['2014 '].apply(lambda x : float(x.split(' ')[0]) if␣
↪type(x) == type(str()) else x)
    df['2019 '] = df['2019 '].apply(lambda x : float(x.split(' ')[0]) if␣
↪type(x) == type(str()) else x)
```

```
    drop_rows = df.loc[
        (df['Country'].isin(filter_values_countries)) |
        (df['Age class'] != 'Total') |
        (df['Sex'] != 'Total') |
        (df['Education level'] != 'All ISCED 2011 levels')
    ]
    df = df.drop(drop_rows.index)
    df = df.drop(columns=[df.columns[0],'Age class','Time␣
↪Frequency','Sex','Education level','Unit of measure'])
    df = df.rename(columns={"2014 " : f"2014_exercise_percent" , "2019 " :␣
↪f"2019_exercise_percent"})
    df = df.reset_index(drop=True)
    df = df.set_index(['Country'])
    return df

#print(read_exercise_dataset())
```

[7]:
```
def read_healthy_life_years_dataset():
    # This dataset reads infromation about frequency,unit,indicator of␣
↪health,sex, and country.
    # https://ec.europa.eu/eurostat/databrowser/view/hlth_hlye/default/table?
↪lang=en
    df = pd.read_csv(dataset_paths["Healthy life years by sex"␣
↪],sep="\t",compression='gzip')
    df['Country'] = df[df.columns[0]].apply(lambda x : acronyms.geo_exp[x.
↪split(',')[-1]])
    df['Time Frequency'] = df[df.columns[0]].apply(lambda x : acronyms.
↪freq_exp[x.split(',')[0]])
    df['Indicator health'] = df[df.columns[0]].apply(lambda x : acronyms.
↪indic_he_exp[x.split(',')[3]])
    df['Unit of measure'] = df[df.columns[0]].apply(lambda  x : acronyms.
↪unit_exp[x.split(',')[1]])
    df['Sex'] = df[df.columns[0]].apply(lambda  x : acronyms.sex_exp[x.
↪split(',')[2]])

    df = df.replace(to_replace=replace_values,value=np.nan)
    # drop nan values
    df = df.dropna()
    drop_rows = df.loc[
        (df['Country'].isin(filter_values_countries)) | # filter only countries
        (df['Sex'] != 'Total') |
        (df['Unit of measure'] == 'year') |
        (df['Indicator health'] != 'Healthy life years at birth in percentage␣
↪of the total life expectancy') # since we are only interested by lf of birth
    ]
```

```
    drop_columns = df.columns.drop(['2014 ','2019 ','Country'])
    df = df.drop(index=drop_rows.index,columns=drop_columns)
    # fix some data to from Number Char type to numbers
    df['2014 '] = df['2014 '].apply(lambda x : float(x.split(' ')[0]) if␣
 ↪type(x) == type(str()) else x)
    df['2019 '] = df['2019 '].apply(lambda x : float(x.split(' ')[0]) if␣
 ↪type(x) == type(str()) else x)
    # take just the years 2014 and 2019
    df = df.reset_index(drop=True)

    df = df.set_index(['Country'])
    return df
#print(read_healthy_life_years_dataset())
```

[8]:
```
def plot_heatmap(df,country):
    # label the values
    cpy = df.loc[country]
    cpy = cpy.fillna(-1)
    for col in cpy.columns:
        cpy[col] = cpy[col].apply(lambda x : -1 if x < 0 else 1)
    plt.figure(figsize=(8,4))
    sns.heatmap(cpy, cmap='coolwarm')
    plt.title(f'{country} heatmap (NaNs filled with -1)')
    plt.show()

def plot_missing_counts(df):
    missing_counts = df.isnull().sum()
    # Plot missing values
    plt.figure(figsize=(12, 8))
    missing_counts[missing_counts > 0].plot(kind='bar')
    plt.title('Number of Missing Values per Column')
    plt.xlabel('Column')
    plt.ylabel('Missing Values')
    plt.xticks(rotation=90)
    plt.show()
```

[9]:
```
# combine the data into a big matrix
datasets = [read_smoking_dataset(),
            read_alcohol_dataset(),
            read_bmi_dataset(),
            read_exercise_dataset()]

# next take the labels for the life_expectancy for 0y old (aka. labels)
df = read_healthy_life_years_dataset()
y14 = df['2014 ']
y19 = df['2019 ']
```

[10]:
```
# create a datasets by year so all dimensions could be placed like a nice table
def dataset_by_year(year,list_of_df,countries):
    df = pd.
 ↪DataFrame(index=countries,columns=[f'{year}_smoking_percent','Drinking at␣
 ↪least once a week',
                                                'Drinking every month','Drinking␣
 ↪less than once a month',
                                                'Drinking never or not in the last␣
 ↪12 months', 'Bmi underweight',
                                                'Bmi normal','Bmi overweight','Bmi␣
 ↪pre-obese','Bmi obese',
                                                 'Walking to get to and from␣
 ↪place','Cycling to get to and from place',
                                                 'Aerobic␣
 ↪sports','Muscle-strengthening'])

    for country in countries:
        df.loc[country,f'{year}_smoking_percent'] = datasets[0].
 ↪loc[country,f'{year}_smoking_percent']

        drink_freq =  datasets[1].loc[country,['Drinking␣
 ↪freq',f'{year}_alcohol_percent']]
        df.loc[country,'Drinking at least once a week'] = drink_freq.
 ↪loc[drink_freq['Drinking freq'] == 'At least once a week'].
 ↪at[country,f'{year}_alcohol_percent']
        df.loc[country,'Drinking every month'] = drink_freq.
 ↪loc[drink_freq['Drinking freq'] == 'Every month'].
 ↪at[country,f'{year}_alcohol_percent']
        df.loc[country,'Drinking less than once a month'] = drink_freq.
 ↪loc[drink_freq['Drinking freq'] == 'Less than once a month'].
 ↪at[country,f'{year}_alcohol_percent']
        df.loc[country,'Drinking never or not in the last 12 months'] =␣
 ↪drink_freq.loc[drink_freq['Drinking freq'] == 'Never or not in the last 12␣
 ↪months'].at[country,f'{year}_alcohol_percent']

        bmis = datasets[2].loc[country,['BMI␣
 ↪description',f'{year}_bmi_percent']]
        df.loc[country,'Bmi underweight'] = bmis.loc[bmis['BMI description'] ==␣
 ↪'Underweight'].at[country,f'{year}_bmi_percent']
        df.loc[country,'Bmi normal'] = bmis.loc[bmis['BMI description'] ==␣
 ↪'Normal'].at[country,f'{year}_bmi_percent']
        df.loc[country,'Bmi overweight'] = bmis.loc[bmis['BMI description'] ==␣
 ↪'Overweight'].at[country,f'{year}_bmi_percent']
        df.loc[country,'Bmi pre-obese'] = bmis.loc[bmis['BMI description'] ==␣
 ↪'Pre-obese'].at[country,f'{year}_bmi_percent']
```

```
        df.loc[country,'Bmi obese'] = bmis.loc[bmis['BMI description'] ==␣
 ↪'Obese'].at[country,f'{year}_bmi_percent']

        exercise = datasets[3].loc[country,['Exercise␣
 ↪level',f'{year}_exercise_percent']]
        df.loc[country,'Walking to get to and from place'] = exercise.
 ↪loc[exercise['Exercise level'] == 'Walking to get to and from place'].
 ↪at[country,f'{year}_exercise_percent']
        df.loc[country,'Cycling to get to and from place'] = exercise.
 ↪loc[exercise['Exercise level'] == 'Cycling to get to and from place'].
 ↪at[country,f'{year}_exercise_percent']
        df.loc[country,'Aerobic sports'] = exercise.loc[exercise['Exercise␣
 ↪level'] == 'Aerobic sports'].at[country,f'{year}_exercise_percent']
        df.loc[country,'Muscle-strengthening'] = exercise.
 ↪loc[exercise['Exercise level'] == 'Muscle-strengthening'].
 ↪at[country,f'{year}_exercise_percent']
    # remove NaN values
    df = df.dropna()
    return df

df14 = dataset_by_year(year=2014,list_of_df=datasets,countries=df.index)
df19 = dataset_by_year(year=2019,list_of_df=datasets,countries=df.index)
```

[11]:
```
common_countries = list(set(y14.index).intersection(set(y19.index)).
 ↪intersection(set(df14.index)).intersection(set(df19.index)))
print(common_countries)
```

```
['Austria', 'Greece', 'Estonia', 'Portugal', 'Spain', 'Denmark', 'Luxembourg']
```

[12]:
```
df14
```

[12]:

Country	2014_smoking_percent	Drinking at least once a week	\
Austria	23.9	2.4	
Denmark	12.3	9.5	
Estonia	22.7	5.2	
Greece	27.0	1.4	
Spain	22.2	2.6	
Ireland	12.7	13.3	
Luxembourg	13.8	11.2	
Portugal	16.3	2.5	

Country	Drinking every month	Drinking less than once a month	\
Austria	16.3	22.2	
Denmark	27.9	35.1	
Estonia	18.1	29.4	

Greece	8.9	12.1
Spain	6.7	13.1
Ireland	18.9	23.2
Luxembourg	23.3	21.7
Portugal	7.7	13.1

Country	Drinking never or not in the last 12 months	Bmi underweight
Austria	59.1	2.7
Denmark	27.5	2.9
Estonia	47.3	2.7
Greece	77.7	2.3
Spain	77.5	2.6
Ireland	44.5	2.1
Luxembourg	43.8	3.2
Portugal	76.7	2.5

Country	Bmi normal	Bmi overweight	Bmi pre-obese	Bmi obese
Austria	50.4	46.9	32.6	14.3
Denmark	51.1	46.0	31.6	14.4
Estonia	45.0	52.3	32.6	19.7
Greece	42.2	55.5	38.6	16.9
Spain	46.4	51.0	34.8	16.2
Ireland	43.4	54.4	36.2	18.2
Luxembourg	50.4	46.4	31.3	15.1
Portugal	45.3	52.2	36.1	16.1

Country	Walking to get to and from place	Cycling to get to and from place
Austria	81.0	24.9
Denmark	79.1	47.1
Estonia	80.1	24.2
Greece	81.2	9.0
Spain	81.7	10.5
Ireland	86.2	13.6
Luxembourg	88.6	17.4
Portugal	60.7	5.8

Country	Aerobic sports	Muscle-strengthening
Austria	72.0	44.3
Denmark	73.8	48.3
Estonia	35.2	15.4
Greece	22.8	12.6
Spain	46.3	14.9
Ireland	45.8	34.3

```
Luxembourg           65.4                36.8
Portugal             35.0                13.0
```

[13]:
```
df19
```

[13]:
```
           2019_smoking_percent Drinking at least once a week  \
Country
Austria                    20.2                          2.3
Belgium                    14.6                          7.6
Denmark                    11.7                          9.1
Estonia                    18.9                          2.9
Greece                     23.6                          0.3
Spain                      19.7                          1.6
France                     17.8                          4.1
Luxembourg                 10.5                         10.5
Portugal                   11.5                          4.0

           Drinking every month Drinking less than once a month  \
Country
Austria                    14.1                           24.6
Belgium                    19.9                           19.0
Denmark                    28.7                           33.9
Estonia                    11.3                           22.9
Greece                      5.7                            9.3
Spain                       4.4                           10.8
France                     16.4                           17.3
Luxembourg                 23.8                           22.2
Portugal                   10.6                           15.8

           Drinking never or not in the last 12 months Bmi underweight  \
Country
Austria                                          59.1              2.6
Belgium                                          53.4              3.2
Denmark                                          28.3              2.3
Estonia                                          62.9              2.5
Greece                                           84.7              1.4
Spain                                            83.1              2.4
France                                           62.3              4.3
Luxembourg                                       43.5              3.7
Portugal                                         69.6              2.0

           Bmi normal Bmi overweight Bmi pre-obese Bmi obese  \
Country
Austria          46.3           51.1          34.4      16.7
Belgium          48.0           48.8          32.9      15.9
Denmark          48.8           48.8          32.9      15.9
Estonia          42.4           55.1          34.0      21.1
```

```
Greece          42.3          56.2          40.1      16.2
Spain           45.3          52.3          36.9      15.4
France          50.3          45.4          31.0      14.4
Luxembourg      49.2          47.1          31.0      16.1
Portugal        43.4          54.5          37.3      17.2

           Walking to get to and from place Cycling to get to and from place  \
Country
Austria                                 89.1                               30.4
Belgium                                 76.9                               29.5
Denmark                                 83.1                               47.5
Estonia                                 79.6                               20.3
Greece                                  82.9                               13.6
Spain                                   88.6                                7.9
France                                  81.9                               15.4
Luxembourg                              89.6                               17.2
Portugal                                68.1                                6.1

           Aerobic sports Muscle-strengthening
Country
Austria              62.2                  42.7
Belgium              42.0                  15.9
Denmark              75.8                  53.0
Estonia              43.5                  21.8
Greece               26.0                  12.1
Spain                50.0                  20.7
France               45.2                  23.9
Luxembourg           66.8                  43.0
Portugal             33.2                  15.2
```

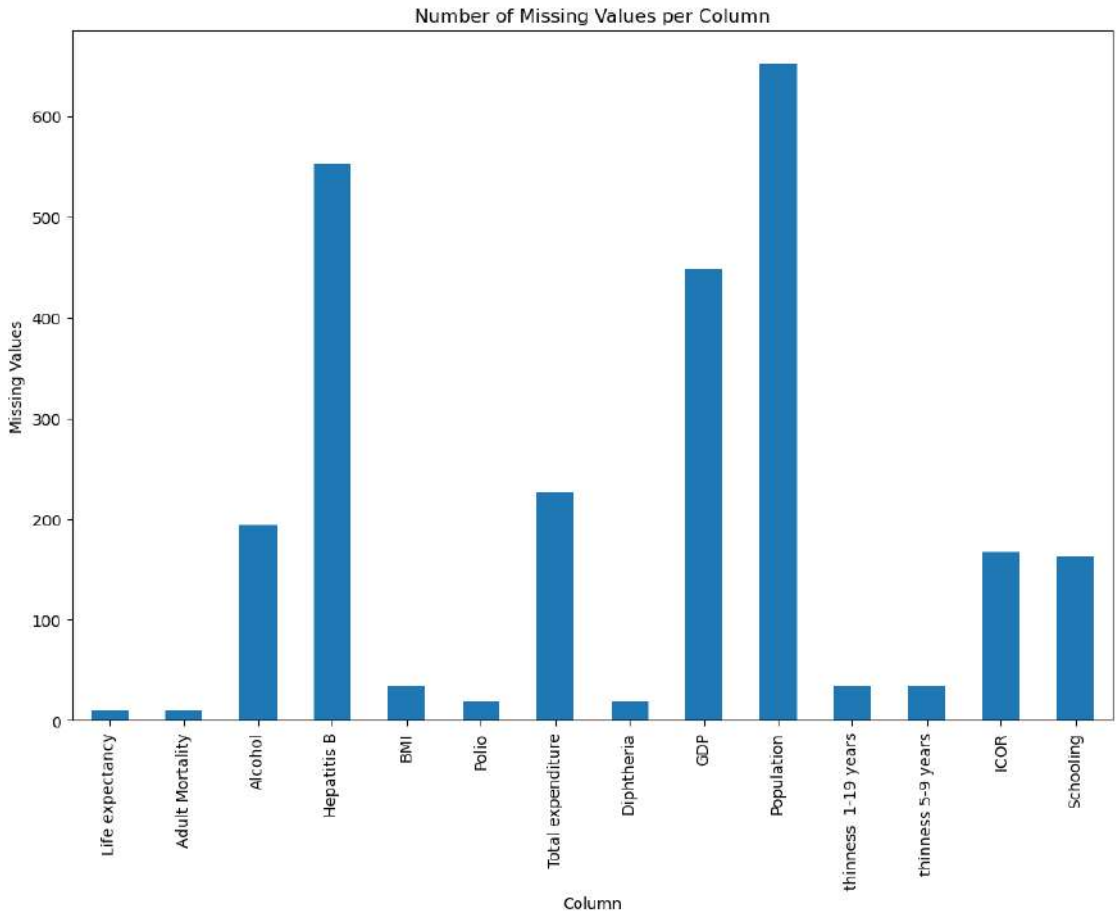
Below I'm loading the life expectancy dataset from the World Health Organization (WHO). It's obtained from Kaggle (which was marked as a recommended dataset source on MyCourses): https://www.kaggle.com/datasets/kumarajarshi/life-expectancy-who It contains data of 193 countries from 2000 to 2015 included.

[14]:
```python
who = pd.read_csv("data/WHO_life_exp.csv")
who.columns = who.columns.str.strip() #Removing extra spaces
who = who.rename(columns={'Income composition of resources': 'ICOR'}) #Renaming␣
 ↪this column to something shorter
# ICOR measures how effectively a country utilizes their resources. A high ICOR␣
 ↪score is believed to be highly correlated with higher life expectancy
print(who.info())
start_year = who['Year'].min()
end_year = who['Year'].max()
print(f"The dataset starts in {start_year} and ends in {end_year}.")
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2938 entries, 0 to 2937
```

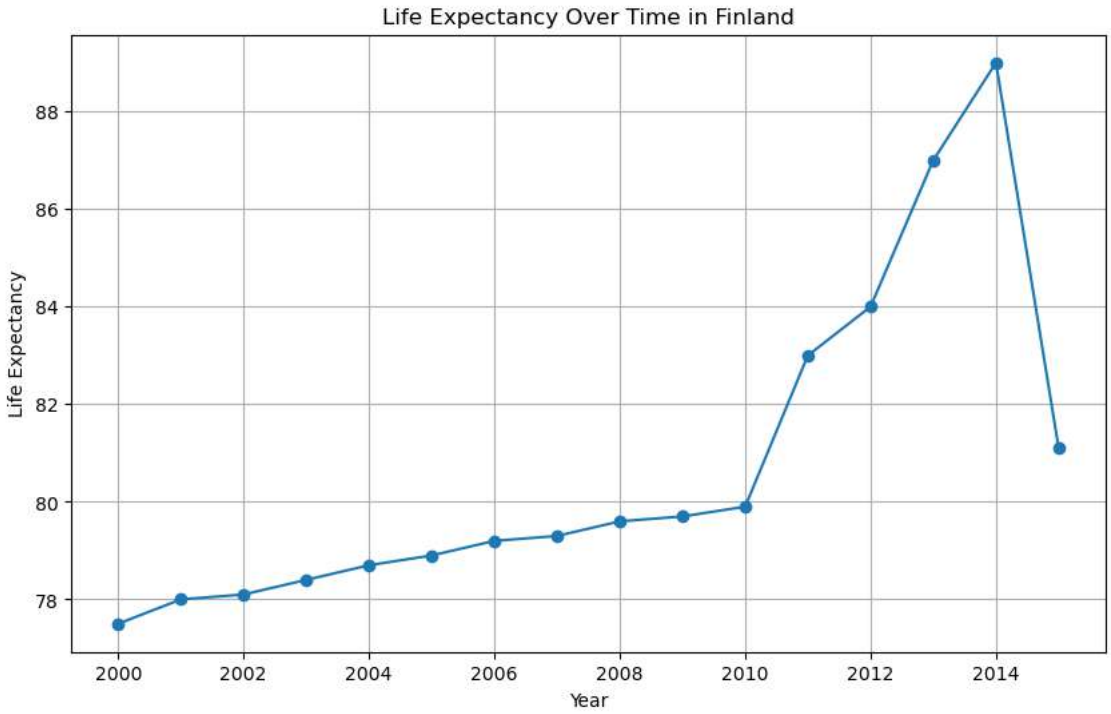
```
Data columns (total 22 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Country                 2938 non-null   object
 1   Year                    2938 non-null   int64
 2   Status                  2938 non-null   object
 3   Life expectancy         2928 non-null   float64
 4   Adult Mortality         2928 non-null   float64
 5   infant deaths           2938 non-null   int64
 6   Alcohol                 2744 non-null   float64
 7   percentage expenditure  2938 non-null   float64
 8   Hepatitis B             2385 non-null   float64
 9   Measles                 2938 non-null   int64
 10  BMI                     2904 non-null   float64
 11  under-five deaths       2938 non-null   int64
 12  Polio                   2919 non-null   float64
 13  Total expenditure       2712 non-null   float64
 14  Diphtheria              2919 non-null   float64
 15  HIV/AIDS                2938 non-null   float64
 16  GDP                     2490 non-null   float64
 17  Population              2286 non-null   float64
 18  thinness  1-19 years    2904 non-null   float64
 19  thinness 5-9 years      2904 non-null   float64
 20  ICOR                    2771 non-null   float64
 21  Schooling               2775 non-null   float64
dtypes: float64(16), int64(4), object(2)
memory usage: 505.1+ KB
None
The dataset starts in 2000 and ends in 2015.
```

[15]:
```
# Count missing values for each column
plot_missing_counts(who)
```



[16]:
```python
def life_exp_overtime(country_name):
    # Plot life expectancy over time for a specific country
    # Filter the data for the specified country
    country_data = who[who['Country'] == country_name]

    # Check if the country exists in the dataset
    if country_data.empty:
        print(f"No data found for {country_name}.")
        return

    # Plot the data
    plt.figure(figsize=(10, 6))
    plt.plot(country_data['Year'], country_data['Life expectancy'], marker='o')
    plt.title(f'Life Expectancy Over Time in {country_name}')
    plt.xlabel('Year')
    plt.ylabel('Life Expectancy')
    plt.grid(True)
    plt.show()
```

[17]:
```
life_exp_overtime('Finland')
```



[18]:
```
# We want to know how many countries changed status (Developing/Developed) over␣
 ↪the years
# Reminder: we have a total of 193 countries in the dataset

# Group by 'Country' and get unique statuses for each country
status_per_country = who.groupby('Country')['Status'].nunique()

# Filter countries that have only one unique status
countries_with_unchanged_status = status_per_country[status_per_country == 1]

# Display the number of such countries
num_countries_with_unchanged_status = countries_with_unchanged_status.shape[0]
print(f"Number of countries that did not change status over the years:␣
 ↪{num_countries_with_unchanged_status}")
```

```
Number of countries that did not change status over the years: 193
```

[19]:
```
from sklearn.preprocessing import LabelEncoder

# Create a LabelEncoder instance
le = LabelEncoder()
```

```
# Fit and transform the country names
who['Country_encoded'] = le.fit_transform(who['Country'])

# Encode 'Status' column as 0 for Developing and 1 for Developed
who['Status_encoded'] = who['Status'].map({'Developing': 0, 'Developed': 1})
```

[20]:
```
# Comparing developing (0) and developed (1) countries on some variables in the␣
 ↪year 2015
who_2015 = who[who['Year'] == 2015]

# Pivot table for the year 2015
pivot_table_2015 = pd.pivot_table(who_2015,
                                  values=['Life expectancy', 'Adult Mortality',␣
 ↪'Alcohol', 'Schooling', 'ICOR'],
                                  index='Status_encoded',
                                  aggfunc='mean')

print(pivot_table_2015)
```

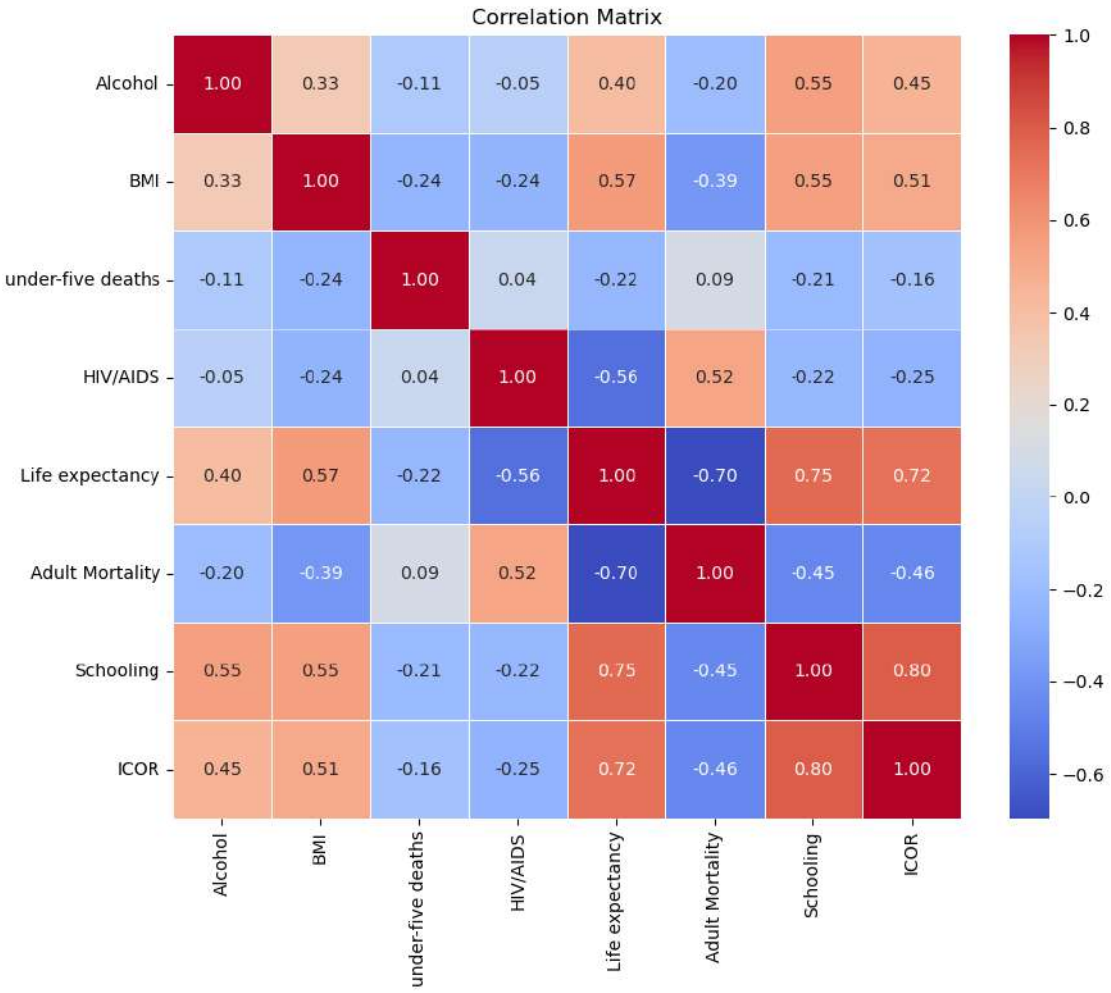
```
                 Adult Mortality  Alcohol      ICOR  Life expectancy  Schooling
Status_encoded
0                     169.390728    2.305  0.653431        69.690066  12.200000
1                      74.875000    6.780  0.881966        80.709375  16.537931
```

[21]:
```
# Compute correlation matrix
correlation_matrix = who[['Alcohol', 'BMI', 'under-five deaths', 'HIV/AIDS',␣
 ↪'Life expectancy', 'Adult Mortality', 'Schooling', 'ICOR']].corr()

# Print the correlation matrix
print(correlation_matrix)
```

```
                     Alcohol       BMI  under-five deaths  HIV/AIDS  \
Alcohol             1.000000  0.330408          -0.112370 -0.048845
BMI                 0.330408  1.000000          -0.237669 -0.243717
under-five deaths  -0.112370 -0.237669           1.000000  0.038062
HIV/AIDS           -0.048845 -0.243717           0.038062  1.000000
Life expectancy     0.404877  0.567694          -0.222529 -0.556556
Adult Mortality    -0.195848 -0.387017           0.094146  0.523821
Schooling           0.547378  0.546961          -0.209373 -0.220429
ICOR                0.450040  0.508774          -0.163305 -0.249519

                   Life expectancy  Adult Mortality  Schooling      ICOR
Alcohol                   0.404877        -0.195848   0.547378  0.450040
BMI                       0.567694        -0.387017   0.546961  0.508774
under-five deaths        -0.222529         0.094146  -0.209373 -0.163305
HIV/AIDS                 -0.556556         0.523821  -0.220429 -0.249519
Life expectancy           1.000000        -0.696359   0.751975  0.724776
```

```
Adult Mortality          -0.696359        1.000000  -0.454612 -0.457626
Schooling                 0.751975       -0.454612   1.000000  0.800092
ICOR                      0.724776       -0.457626   0.800092  1.000000
```

[22]:
```
# Plot heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f',␣
 ↪linewidths=0.5)
plt.title('Correlation Matrix')
plt.show()
```



[23]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
# Drop rows with missing values to ensure a clean dataset
who_cleaned = who.dropna()
```

```
# Aggregate data by country using median values
aggregated_data = who_cleaned.groupby('Country_encoded').agg({
'Life expectancy': 'median',
'Alcohol': 'median',
'Adult Mortality': 'median',
'Schooling': 'median',
'ICOR': 'median',
'HIV/AIDS': 'median',
'BMI': 'median',
'GDP': 'median',
'percentage expenditure': 'median',
'under-five deaths': 'median',
'Polio': 'median',
'Population': 'median',
'Measles': 'median',
'infant deaths': 'median',
'Diphtheria': 'median',
'Hepatitis B': 'median',
}).reset_index()
# Separate features and target
X = aggregated_data.drop(columns=['Life expectancy', 'Country_encoded'])
y = aggregated_data['Life expectancy']
# Convert all columns to numeric if they are not already
X = X.apply(pd.to_numeric, errors='coerce')
# Drop any remaining rows with NaN values after conversion
X = X.dropna()
y = y[X.index] # Align y with the cleaned X
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.
 ↪2,random_state=42)
# Initialize and train Random Forest
model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
# Predict and evaluate
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")
# Feature importance
importances = model.feature_importances_
feature_importance_df = pd.DataFrame({
'Feature': X.columns,
'Importance': importances
}).sort_values(by='Importance', ascending=False)
print(feature_importance_df)
```

```
Mean Squared Error: 6.3449132592592505
                   Feature  Importance
```

```
1          Adult Mortality    0.505150
3                     ICOR    0.270556
4                  HIV/AIDS    0.094272
6                      GDP    0.071533
7   percentage expenditure    0.017500
5                      BMI    0.006775
2                Schooling    0.006135
11                 Measles    0.004885
8        under-five deaths    0.004707
0                  Alcohol    0.004552
12           infant deaths    0.004043
10              Population    0.002817
13              Diphtheria    0.002595
9                    Polio    0.002330
14             Hepatitis B    0.002151
```

[24]:
```
""" The phase 2 section begins here. The correct models are importet."""
# polynomial features
from sklearn.preprocessing import PolynomialFeatures
# linear regression
from sklearn.linear_model import LinearRegression
# ridge
from sklearn.linear_model import Ridge
# MSE loss function
from sklearn.metrics import mean_squared_error
```

[25]:
```
print(who_cleaned.columns)
```

```
Index(['Country', 'Year', 'Status', 'Life expectancy', 'Adult Mortality',
       'infant deaths', 'Alcohol', 'percentage expenditure', 'Hepatitis B',
       'Measles', 'BMI', 'under-five deaths', 'Polio', 'Total expenditure',
       'Diphtheria', 'HIV/AIDS', 'GDP', 'Population', 'thinness  1-19 years',
       'thinness 5-9 years', 'ICOR', 'Schooling', 'Country_encoded',
       'Status_encoded'],
      dtype='object')
```

[26]:
```
years_of_interest = list(range(2013,2016))
who_cleaned_yoi = who_cleaned.loc[who_cleaned['Year'].isin(years_of_interest)]
who_countries = who_cleaned_yoi['Country'].unique()
X = who_cleaned_yoi[['Country','Country_encoded','Life expectancy','Adult␣
 ↪Mortality',
                     'Alcohol','infant deaths',
                      'percentage expenditure','Hepatitis␣
 ↪B','Measles','Diphtheria',
                     'HIV/AIDS','GDP','Population','thinness  1-19 years',
                     'thinness 5-9 years','ICOR','Schooling']]
```

```
# division to training,testing, and validation by countries
n_countries = len(who_countries)
# the division is done by 80% 10% 10%
training_len = int(np.round(0.8*n_countries))
testing_len = int(n_countries - training_len)//2
validation_len = testing_len
random_state = 60
np.random.state = random_state
np.random.seed = random_state
np.random.shuffle(who_countries)
countries_tr = who_countries[:training_len]
countries_tst = who_countries[training_len:training_len+testing_len]
countries_val = who_countries[training_len+testing_len:-1]
```

[27]:
```
Xtr = X.loc[X['Country'].isin(countries_tr)]
ytr = Xtr[['Life expectancy']].to_numpy()
Xtr = Xtr.drop(columns=['Country','Country_encoded','Life expectancy']).
 ↪to_numpy()

Xtst = X.loc[X['Country'].isin(countries_tst)]
ytst = Xtst[['Life expectancy']].to_numpy()
Xtst = Xtst.drop(columns=['Country','Country_encoded','Life expectancy']).
 ↪to_numpy()

Xval = X.loc[X['Country'].isin(countries_val)]
yval = Xval[['Life expectancy']].to_numpy()
Xval = Xval.drop(columns=['Country','Country_encoded','Life expectancy']).
 ↪to_numpy()

degrees = list(range(1,8))
model_dict_who =␣
 ↪dict(zip(degrees,[(PolynomialFeatures(degree=i),LinearRegression(fit_intercept=False))␣
 ↪for i in degrees]))
training_error_who = np.zeros(len(degrees))
testing_error_who = np.zeros(len(degrees))
validation_error_who = np.zeros(len(degrees))
for key in model_dict_who.keys():
    X_poly = model_dict_who[key][0].fit_transform(Xtr)
    model_dict_who[key][1].fit(X_poly,ytr)
    y_pred = model_dict_who[key][1].predict(X_poly)
    training_error_who[key-1] = mean_squared_error(y_true=ytr,y_pred=y_pred)

    X_poly = model_dict_who[key][0].transform(Xtst)
    y_pred = model_dict_who[key][1].predict(X_poly)
    testing_error_who[key-1] = mean_squared_error(y_true=ytst,y_pred=y_pred)
```

```
    X_poly = model_dict_who[key][0].transform(Xval)
    y_pred = model_dict_who[key][1].predict(X_poly)
    validation_error_who[key-1] = mean_squared_error(y_true=yval,y_pred=y_pred)
    print(f"degree : {key} ; tr error : {training_error_who[key-1]} ; tst error␣
 ↪: {testing_error_who[key-1]} ; val error : {validation_error_who[key-1]}")
```

```
degree : 1 ; tr error : 7.970307909255271 ; tst error : 17.002710261854336 ; val
error : 19.79466286564975
degree : 2 ; tr error : 187.70733489996115 ; tst error : 460906.2948567874 ; val
error : 11143.39723302472
degree : 3 ; tr error : 2882.736658551505 ; tst error : 237134215.39622843 ; val
error : 62582.91144387924
degree : 4 ; tr error : 4155.20345579403 ; tst error : 2087369209.3866994 ; val
error : 249049.30992941352
degree : 5 ; tr error : 4437.866422328527 ; tst error : 10591517557.18011 ; val
error : 1701826.41323643
degree : 6 ; tr error : 4575.547482449568 ; tst error : 15263229.040003201 ; val
error : 137993.38599511606
degree : 7 ; tr error : 4732.2338831015895 ; tst error : 290545096.89533883 ;
val error : 112162.8929450405
```

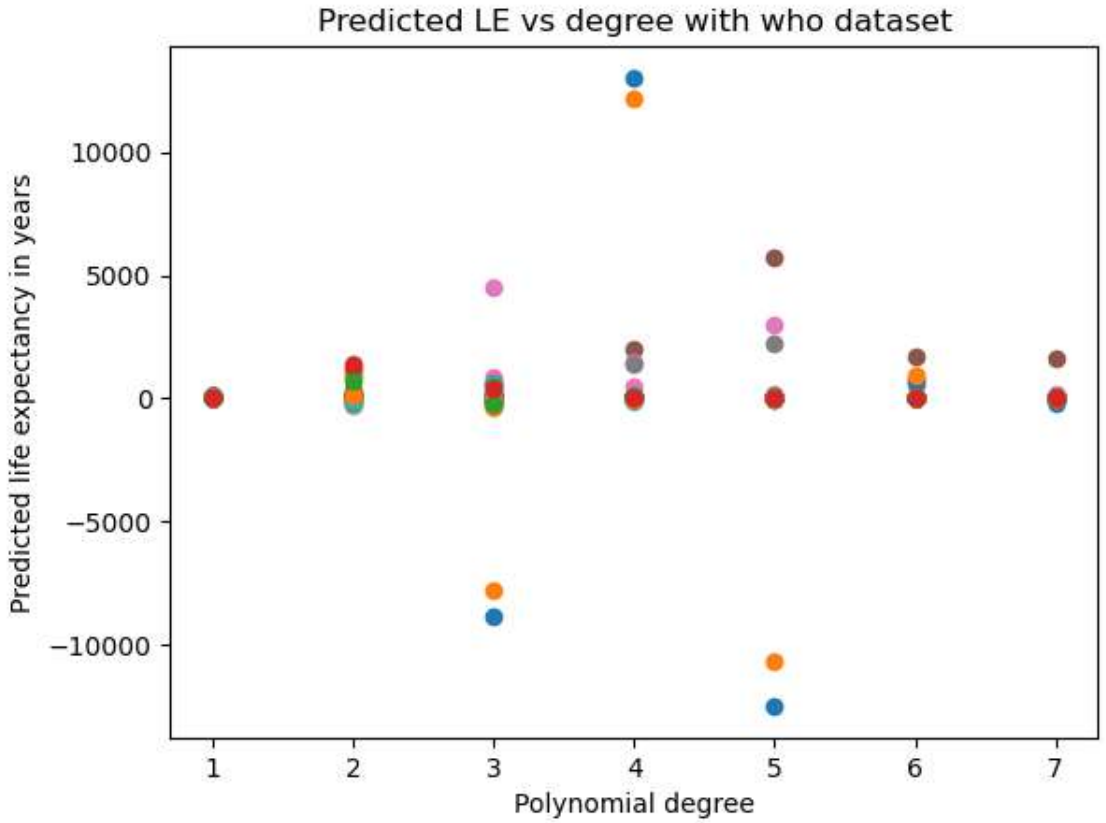
[28]:
```
# Training the ridge model
alpha = 1
rdg_who = Ridge(alpha=alpha)
rdg_who.fit(Xtr,ytr)
tr_error = mean_squared_error(y_pred=rdg_who.predict(Xtr),y_true=ytr)
tst_error = mean_squared_error(y_pred=rdg_who.predict(Xtst),y_true=ytst)
val_error = mean_squared_error(y_pred=rdg_who.predict(Xtst),y_true=ytst)
print(f"tr error : {tr_error} ; tst error : {tst_error} ; val error␣
 ↪{val_error}")
```

```
tr error : 9.379763369420333 ; tst error : 24.77934278641026 ; val error
24.77934278641026
```

```
/opt/software/lib/python3.10/site-packages/sklearn/linear_model/_ridge.py:200:
LinAlgWarning: Ill-conditioned matrix (rcond=7.63031e-19): result may not be
accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
```

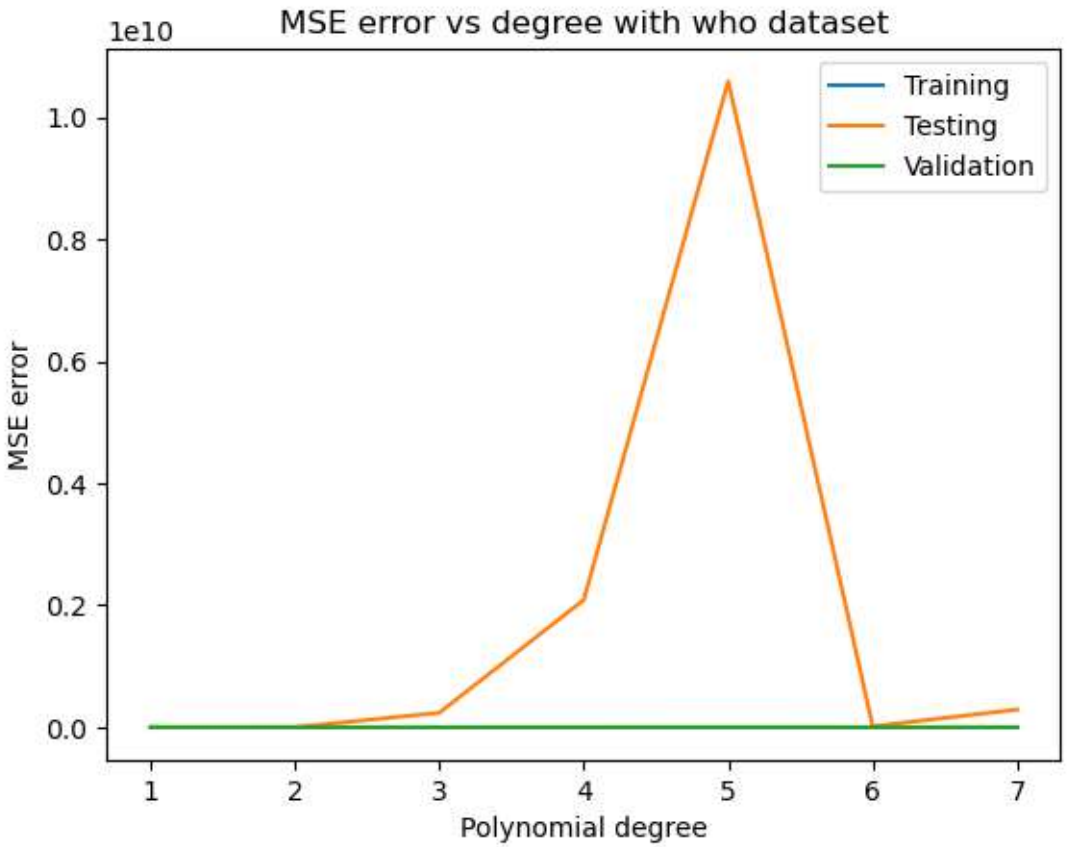
[29]:
```
# plot the degrees in x-axis and predicted outcome to y-axis
df = who_cleaned.loc[who_cleaned['Country'].isin(countries_val)]
predictions = np.zeros((len(df.index),len(degrees)))
legends = list(df.index)
fig, ax = plt.subplots()
for ix,country in enumerate(df.index):
    ts = df.loc[country,['Adult Mortality','Alcohol','infant deaths',
                'percentage expenditure','Hepatitis B','Measles','Diphtheria',
                'HIV/AIDS','GDP','Population','thinness  1-19 years',
```

```
                'thinness 5-9 years','ICOR','Schooling']].to_numpy()
    ts = np.expand_dims(ts,axis=0)
    for i in range(len(degrees)):
        predictions[ix,i] = model_dict_who[i+1][1].
↪predict(model_dict_who[i+1][0].transform(ts))[0]
    plt.scatter(x=degrees,y=predictions[ix,:])

#ax.legend(legends)
ax.set_title("Predicted LE vs degree with who dataset")
ax.set_ylabel('Predicted life expectancy in years')
ax.set_xlabel('Polynomial degree')
plt.savefig('Predicted_LE_vs_degree_who.png', bbox_inches='tight', dpi=200)
plt.show()
```



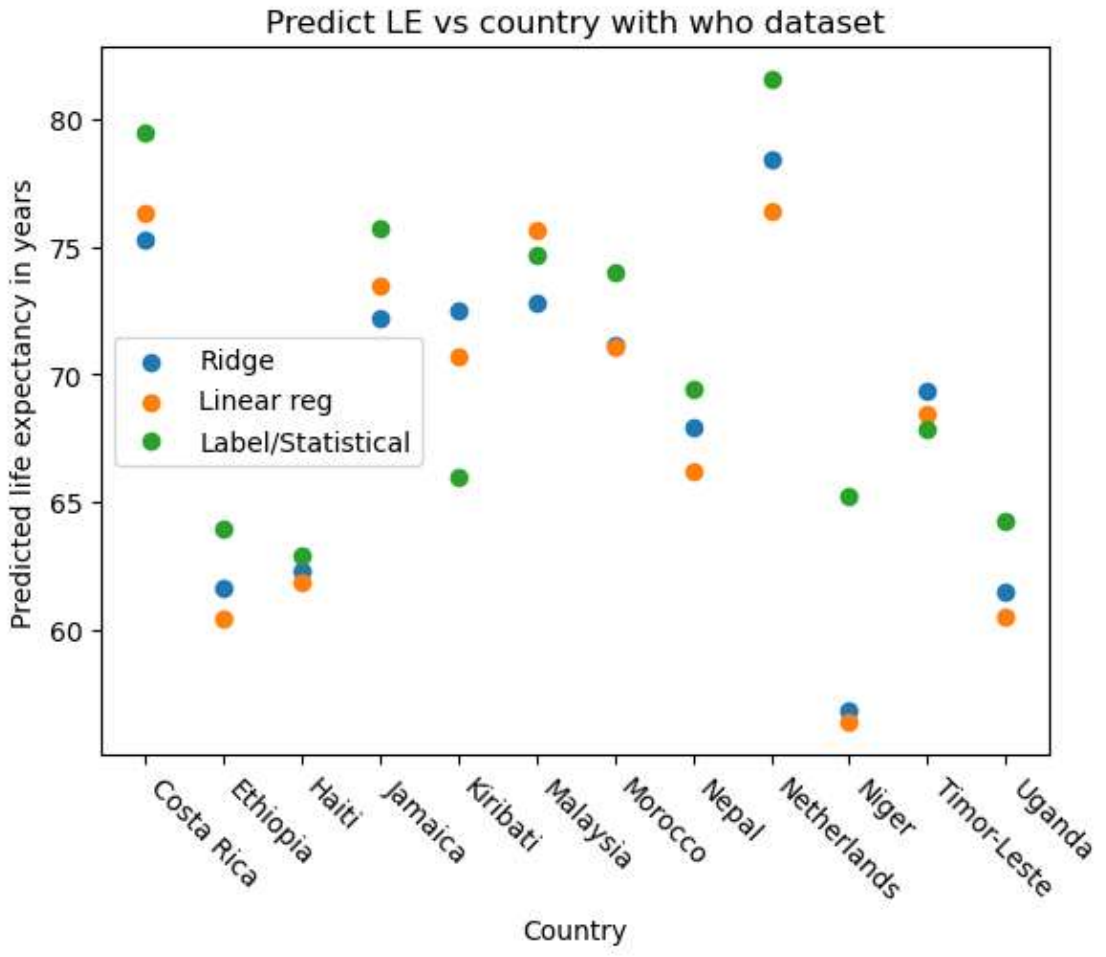
[30]:
```
# Plot the training and testing error vs degree
legends = ['Training','Testing','Validation']
fig, ax = plt.subplots()
plt.plot(degrees,training_error_who)
plt.plot(degrees,testing_error_who)
plt.plot(degrees,validation_error_who)
```

```
ax.legend(legends)
ax.set_title('MSE error vs degree with who dataset')
ax.set_ylabel('MSE error')
ax.set_xlabel('Polynomial degree')
plt.savefig('MSE_error_vs_degree_who.png', bbox_inches='tight', dpi=200)
plt.show()
```



[31]:
```
# plot the degrees in x-axis and predicted outcome to y-axis
df = X.loc[X['Country'].isin(countries_val)].set_index('Country',drop=True)
countries = list(df.index.unique())
predictions = np.zeros((len(countries),3))
legends = ['Ridge','Linear reg','Label/Statistical']
fig, ax = plt.subplots()
for ix,country in enumerate(countries):
    ts = np.mean(df.loc[country,['Adult Mortality','Alcohol','infant deaths',
                'percentage expenditure','Hepatitis B','Measles','Diphtheria',
                'HIV/AIDS','GDP','Population','thinness  1-19 years',
                'thinness 5-9 years','ICOR','Schooling']].to_numpy(),axis=0) #␣
 ↪take mean of several years
```

```
    ts = np.expand_dims(ts,axis=0)
    predictions[ix,0] = rdg_who.predict(ts)[0]
    predictions[ix,1] = model_dict_who[1][1].predict(model_dict_who[1][0].
 ↪transform(ts))[0]
    predictions[ix,2] = np.mean(df.loc[country,'Life expectancy'].to_numpy()) #␣
 ↪take mean of several years

plt.scatter(x=np.arange(predictions.shape[0]),y=predictions[:,0])
plt.scatter(x=np.arange(predictions.shape[0]),y=predictions[:,1])
plt.scatter(x=np.arange(predictions.shape[0]),y=predictions[:,2])
ax.set_xticks(ticks=np.arange(len(countries)),labels=countries)
# Rotate the tick labels and set their alignment.
plt.setp(ax.get_xticklabels(), rotation=-45, ha="left")
ax.legend(legends)
ax.set_title("Predict LE vs country with who dataset")
ax.set_ylabel('Predicted life expectancy in years')
ax.set_xlabel('Country')
plt.savefig('Predicted_LE_vs_country_who.png', bbox_inches='tight', dpi=200)
plt.show()
```

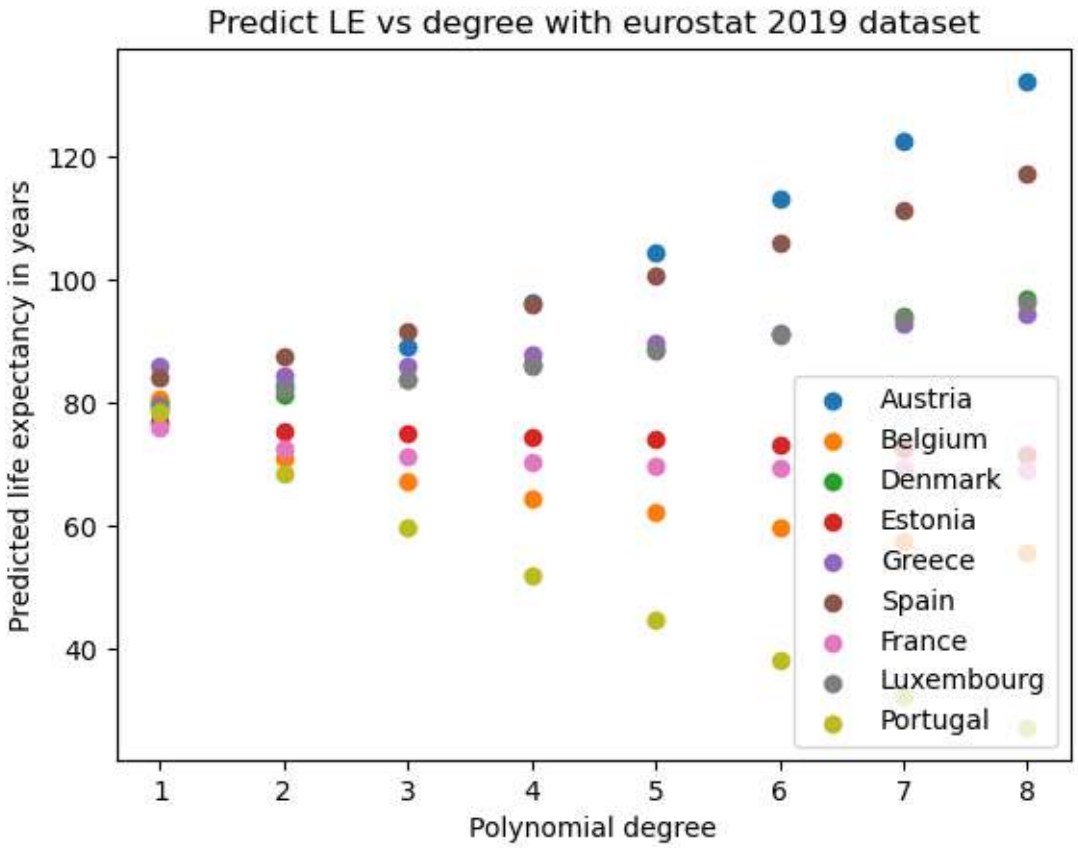


[32]:
```
# training loop of polynomial for eurostat dataset
# df14,df19, y14, y19
countries = df14.index
n_countries = len(countries)
# the division is done by 80% 20%
training_len = int(np.round(0.8*n_countries))
testing_len = int(n_countries - training_len)
countries_tr = countries[0:training_len]
countries_tst = countries[training_len:-1]
Xtr = df14.loc[countries_tr].to_numpy()
ytr = y14.loc[countries_tr].to_numpy()
Xtst = df14.loc[countries_tst].to_numpy()
ytst = y14.loc[countries_tst].to_numpy()

Xval = df19.loc[common_countries].to_numpy()
yval = y19.loc[common_countries].to_numpy()
```

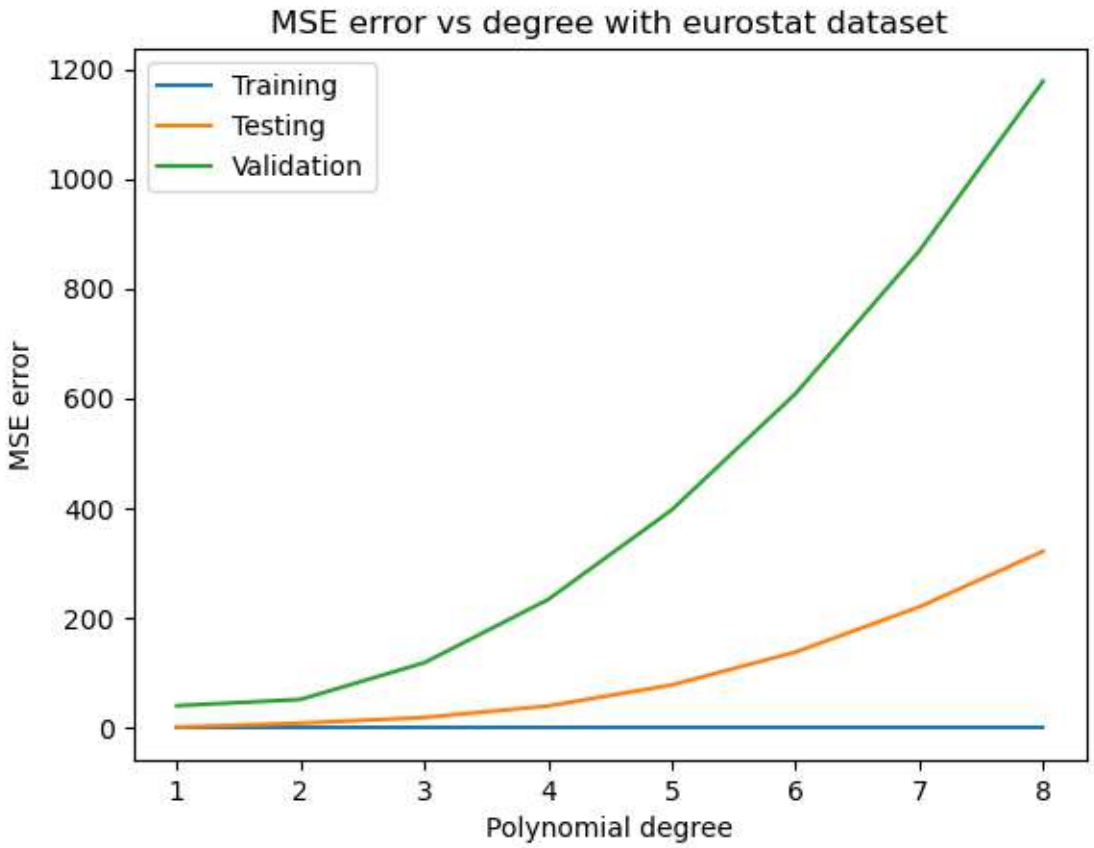
[33]:
```
degrees = list(range(1,9))
model_dict_eurostat =␣
 ↪dict(zip(degrees,[(PolynomialFeatures(degree=i),LinearRegression(fit_intercept=False))␣
 ↪for i in degrees]))
training_error_eurostat = np.zeros(len(degrees))
testing_error_eurostat = np.zeros(len(degrees))
validation_error_eurostat = np.zeros(len(degrees))
for key in model_dict_eurostat.keys():
    X_poly = model_dict_eurostat[key][0].fit_transform(Xtr)
    model_dict_eurostat[key][1].fit(X_poly,ytr)
    y_pred = model_dict_eurostat[key][1].predict(X_poly)
    training_error_eurostat[key-1] =␣
 ↪mean_squared_error(y_true=ytr,y_pred=y_pred)

    X_poly = model_dict_eurostat[key][0].transform(Xtst)
    y_pred = model_dict_eurostat[key][1].predict(X_poly)
    testing_error_eurostat[key-1] =␣
 ↪mean_squared_error(y_true=ytst,y_pred=y_pred)

    print(f"degree : {key} ; tr error : {training_error_eurostat[key-1]} ; tst␣
 ↪error : { testing_error_eurostat[key-1]}")

for key in model_dict_eurostat.keys():
    X_poly = model_dict_eurostat[key][0].transform(Xval)
    y_pred = model_dict_eurostat[key][1].predict(X_poly)
    validation_error_eurostat[key-1] =␣
 ↪mean_squared_error(y_true=yval,y_pred=y_pred)
```

```
degree : 1 ; tr error : 3.365806528942984e-28 ; tst error : 0.12425173888015897
degree : 2 ; tr error : 4.7121291405201775e-28 ; tst error : 7.381158676554432
degree : 3 ; tr error : 7.202825971937985e-27 ; tst error : 17.78833442688966
degree : 4 ; tr error : 1.1107161545511846e-27 ; tst error : 38.75813321180635
degree : 5 ; tr error : 1.8175355256292112e-27 ; tst error : 77.04833323784263
degree : 6 ; tr error : 7.640380820700573e-27 ; tst error : 137.0580619999219
degree : 7 ; tr error : 1.2453484157089039e-26 ; tst error : 219.28588929775404
degree : 8 ; tr error : 5.459338189945519e-26 ; tst error : 320.70586699916015
```

[34]:
```
# plot the degrees in x-axis and predicted outcome to y-axis
df = df19
predictions = np.zeros((len(df.index),len(degrees)))
legends = list(df.index)
fig, ax = plt.subplots()
for country in range(len(df.index)):
    ts = df.loc[legends[country]]
    ts = np.expand_dims(ts.to_numpy(),axis=0)
    for i in range(len(degrees)):
        predictions[country,i] = model_dict_eurostat[i+1][1].
 ↪predict(model_dict_eurostat[i+1][0].transform(ts))[0]
    plt.scatter(x=degrees,y=predictions[country,:])

ax.legend(legends)
ax.set_title("Predict LE vs degree with eurostat 2019 dataset")
ax.set_ylabel('Predicted life expectancy in years')
ax.set_xlabel('Polynomial degree')
plt.savefig('Predicted_LE_vs_degree_eurostat.png', bbox_inches='tight', dpi=200)
plt.show()
```



[35]:
```
# Plot the training and testing error vs degree
legends = ['Training','Testing','Validation']
fig, ax = plt.subplots()
plt.plot(degrees,training_error_eurostat)
plt.plot(degrees,testing_error_eurostat)
plt.plot(degrees,validation_error_eurostat)
ax.legend(legends)
ax.set_title('MSE error vs degree with eurostat dataset')
ax.set_ylabel('MSE error')
ax.set_xlabel('Polynomial degree')
plt.savefig('MSE_error_vs_degree_eurostat.png', bbox_inches='tight', dpi=200)
plt.show()
```

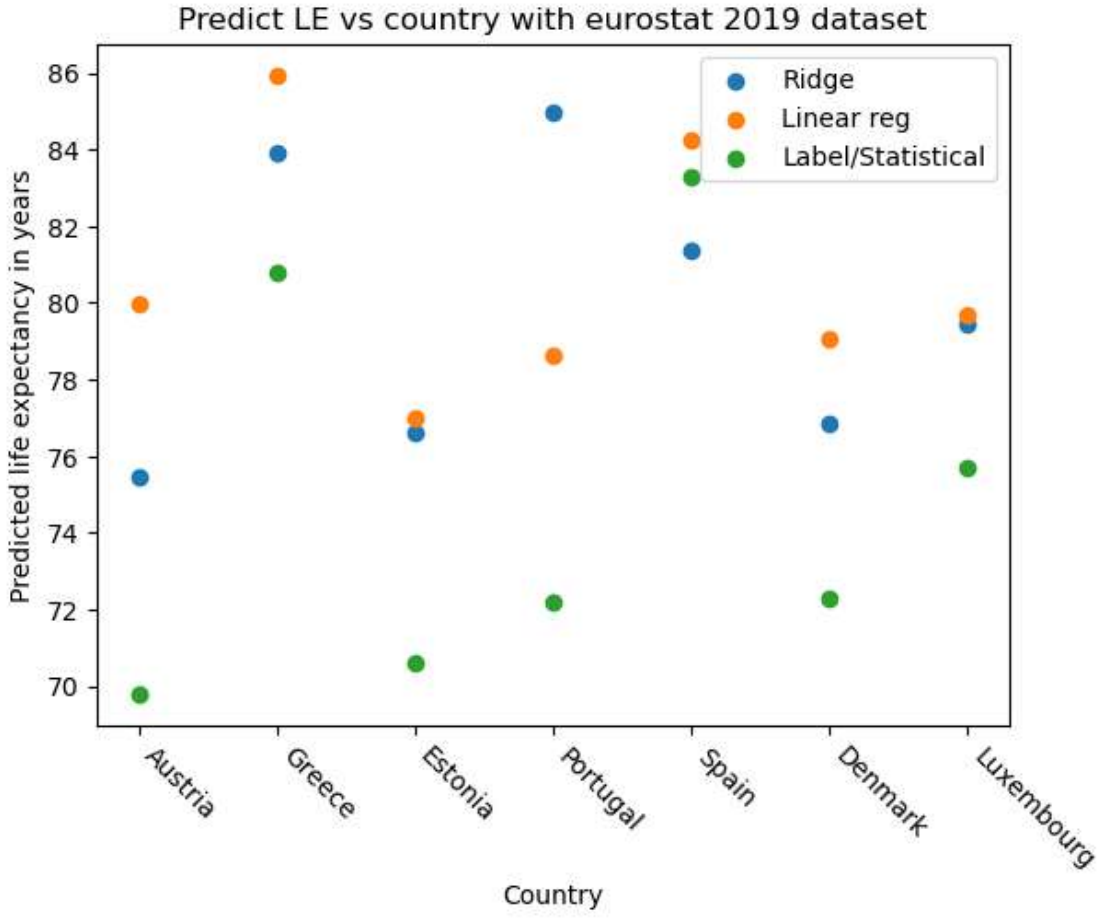


[36]:
```
# Training the ridge model
alpha = 1
rdg_eurostat = Ridge(alpha=alpha)
rdg_eurostat.fit(Xtr,ytr)
tr_error = mean_squared_error(y_pred=rdg_eurostat.predict(Xtr),y_true=ytr)
tst_error = mean_squared_error(y_pred=rdg_eurostat.predict(Xtst),y_true=ytst)
val_error = mean_squared_error(y_pred=rdg_eurostat.predict(Xtst),y_true=ytst)
print(f"tr error : {tr_error} ; tst error : {tst_error} ; val error␣
 ↪{val_error}")
```

```
tr error : 0.0025334816222466945 ; tst error : 0.031547048313643015 ; val error
0.031547048313643015
```

[37]:
```
# plot the degrees in x-axis and predicted outcome to y-axis
countries = common_countries
df = df19.loc[countries]
predictions = np.zeros((len(df.index),2))
actual = y19.loc[countries].to_numpy()
legends = ['Ridge','Linear reg','Label/Statistical']
fig, ax = plt.subplots()
```

```
for country in range(len(df.index)):
    ts = df.loc[df.index[country]]
    ts = np.expand_dims(ts.to_numpy(),axis=0)
    predictions[country,0] = rdg_eurostat.predict(ts)[0]
    predictions[country,1] = model_dict_eurostat[1][1].
 ↪predict(model_dict_eurostat[1][0].transform(ts))[0]

plt.scatter(x=np.arange(predictions.shape[0]),y=predictions[:,0])
plt.scatter(x=np.arange(predictions.shape[0]),y=predictions[:,1])
plt.scatter(x=np.arange(predictions.shape[0]),y=actual)
ax.set_xticks(ticks=np.arange(len(df.index)),labels=list(df.index))
# Rotate the tick labels and set their alignment.
plt.setp(ax.get_xticklabels(), rotation=-45, ha="left")
ax.legend(legends)
ax.set_title("Predict LE vs country with eurostat 2019 dataset")
ax.set_ylabel('Predicted life expectancy in years')
ax.set_xlabel('Country')
plt.savefig('Predicted_LE_vs_country_eurostat.png', bbox_inches='tight',␣
 ↪dpi=200)
plt.show()
```



Appendix 7.2.B - WHO regressions, k-fold, 60-20-20 split, Random Forest

October 9, 2024

[28]:
```
# import the important libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

# This python file contains the acronyms for the EUROSTAT acronyms
# File contains every eurostat quantity used in the project and corresponding␣
 ↪dictionary
# with explanations.
# Example quantity_exp[quantity_acronym]
import acronyms

# import sklearn applications

dataset_paths = {
    "Pollution,crime and other environmental problems" : "./data/
 ↪estat_ilc_mddw02.tsv.gz",
    "Healthy life years by sex" : "./data/estat_hlth_hlye.tsv.gz",
    "Body mass index (BMI) by sex, age and educational attainment level" : "./
 ↪data/estat_hlth_ehis_bm1e.tsv.gz",
    "Daily smokers of cigarettes by sex, age and educational attainment level" ␣
 ↪: "./data/estat_hlth_ehis_sk3e.tsv.gz",
    "Frequency of heavy episodic drinking by sex, age and educational␣
 ↪attainment level" : "./data/estat_hlth_ehis_al3e.tsv.gz",
    "Performing (non-work-related) physical activities by sex, age and␣
 ↪educational attainment level" : "./data/estat_hlth_ehis_pe3e.tsv.gz",
    "Daily consumption of fruit and vegetables by sex, age and educational␣
 ↪attainment level" : "./data/estat_hlth_ehis_fv3e.tsv.gz",
}
```

[29]:
```
def read_pollution_dataset():
    # This dataset reads information on pollution between year 2003-2023
```

```
    # Read more here:
    df = pd.read_csv(dataset_paths["Pollution,crime and other environmental␣
 ↪problems" ],sep="\t",compression='gzip')
    country = df[df.columns[0]].apply(lambda x : x.split(',')[-1])
    unit = df[df.columns[0]].apply(lambda x : x.split(',')[2])
    print(df.columns[0])
    # TODO: put the correct information to correct lines
    df['Country'] = country
    df = df.drop(columns=[df.columns[0]])
    return df

def read_healthy_life_years_dataset():
    # This dataset reads infromation about frequency,unit,indicator health,sex,␣
 ↪and country.
    # between years 2004-2022 source eurostat
    # https://ec.europa.eu/eurostat/cache/metadata/en/hlth_hlye_esms.htm
    df = pd.read_csv(dataset_paths["Healthy life years by sex"␣
 ↪],sep="\t",compression='gzip')
    # alternative df['freq,unit,sex,indic_he,geo\TIME_PERIOD']
    df['Country'] = df[df.columns[0]].apply(lambda x : x.split(',')[-1])
    freq = df[df.columns[0]].apply(lambda  x : acronyms.freq_exp[x.
 ↪split(',')[0]])
    #df['Indicator health'] = df[df.columns[0]].apply(lambda x : acronyms.
 ↪indic_he_exp[x.split(',')[3]])
    df['Indicator health'] = df[df.columns[0]].apply(lambda x : x.split(',')[3])
    df['Unit of measure'] = df[df.columns[0]].apply(lambda  x : x.split(',')[1])
    df['Sex'] = df[df.columns[0]].apply(lambda  x : x.split(',')[2])
    df = df.drop(columns=[df.columns[0]])
    return df

def read_bmi_dataset():
    # returns pandas dataset of bmi based on education
    # contains frequency,unit, bmi, education level according to standard, sex,␣
 ↪age, country and time year 2014 or 2019
    # https://ec.europa.eu/eurostat/cache/metadata/en/hlth_det_esms.htm
    df = pd.read_csv(dataset_paths["Body mass index (BMI) by sex, age and␣
 ↪educational attainment level" ],sep="\t",compression='gzip')
    df['Country'] = df[df.columns[0]].apply(lambda x : x.split(',')[-1])
    #df['Time Frequency'] = df[df.columns[0]].apply(lambda x : x.split(',')[0])
    df['Unit of measure'] = df[df.columns[0]].apply(lambda x : acronyms.
 ↪unit_exp[x.split(',')[1]])
    df['BMI description'] = df[df.columns[0]].apply(lambda x : acronyms.
 ↪bmi_exp[x.split(',')[2]])
    df['Education level'] = df[df.columns[0]].apply(lambda x : x.split(',')[3])
    df['Sex'] = df[df.columns[0]].apply(lambda x : x.split(',')[4])
    df['Age class'] = df[df.columns[0]].apply(lambda x : x.split(',')[5])
```

```
    df = df.drop(columns=[df.columns[0]])
    return df

def read_smoking_dataset():
    # returns pandas dataset of bmi based on education
    # contains frequency,unit, smoking, education level according to standard,␣
 ↪sex, age, country and time year 2014 or 2019
    # https://ec.europa.eu/eurostat/cache/metadata/en/hlth_det_esms.htm
    df = pd.read_csv(dataset_paths["Daily smokers of cigarettes by sex, age and␣
 ↪educational attainment level"],sep="\t",compression='gzip')
    df['Country'] = df[df.columns[0]].apply(lambda x : x.split(',')[-1])
    df['Smoking levels'] = df[df.columns[0]].apply(lambda x : x.split(',')[2])
    df['Unit of measure'] = df[df.columns[0]].apply(lambda x : x.split(',')[1])
    df['Sex'] = df[df.columns[0]].apply(lambda x : x.split(',')[4])
    df['Age class'] = df[df.columns[0]].apply(lambda x : x.split(',')[5])
    df = df.drop(columns=[df.columns[0]])
    return df

def read_alcohol_dataset():
    # Frequency of heavy episodic drinking by sex, age and educational␣
 ↪attainment level
    # freq,unit,frequenc,isced11,sex,age,geo\TIME_PERIOD  2014
    df = pd.read_csv(dataset_paths["Frequency of heavy episodic drinking by␣
 ↪sex, age and educational attainment level"],sep="\t",compression='gzip')
    #df['Time Frequency'] = df[df.columns[0]].apply(lambda x : x.split(',')[0])
    df['Country'] = df[df.columns[0]].apply(lambda x : x.split(',')[-1])
    df['Unit of measure'] = df[df.columns[0]].apply(lambda x : x.split(',')[1])
    df['Sex'] = df[df.columns[0]].apply(lambda x : x.split(',')[4])
    df['Drinking freq'] = df[df.columns[0]].apply(lambda x : x.split(',')[2])
    df['Education level'] = df[df.columns[0]].apply(lambda x : x.split(',')[3])
    df['Age class'] = df[df.columns[0]].apply(lambda x : x.split(',')[5])
    df = df.drop(columns=[df.columns[0]])
    return df

def read_exercise_dataset():
        #freq,unit,physact,isced11,sex,age,geo\TIME_PERIOD
        df = pd.read_csv(dataset_paths["Performing (non-work-related) physical␣
 ↪activities by sex, age and educational attainment␣
 ↪level"],sep="\t",compression='gzip')
        #df['Time Frequency'] = df[df.columns[0]].apply(lambda x : x.
 ↪split(',')[0])
        df['Country'] = df[df.columns[0]].apply(lambda x : x.split(',')[-1])
        df['Unit of measure'] = df[df.columns[0]].apply(lambda x : x.
 ↪split(',')[1])
        df['Exercise level'] = df[df.columns[0]].apply(lambda x : x.
 ↪split(',')[2])
```

```
        df['Sex'] = df[df.columns[0]].apply(lambda x : x.split(',')[4])
        df['Education level'] = df[df.columns[0]].apply(lambda x : x.
↪split(',')[3])
        df['Age class'] = df[df.columns[0]].apply(lambda x : x.split(',')[5])
        df = df.drop(columns=[df.columns[0]])
        return df
```

Below I'm loading the life expectancy dataset from the World Health Organization (WHO). It's obtained from Kaggle (which was marked as a recommended dataset source on MyCourses): https://www.kaggle.com/datasets/kumarajarshi/life-expectancy-who It contains data of 193 countries from 2000 to 2015 included.

[30]:
```
who = pd.read_csv("data/WHO_life_exp.csv")
who.columns = who.columns.str.strip() #Removing extra spaces
who = who.rename(columns={'Income composition of resources': 'ICOR'}) #Renaming␣
↪this column to something shorter
# ICOR measures how effectively a country utilizes their resources. A high ICOR␣
↪score is believed to be highly correlated with higher life expectancy
who.head()
```

[30]:

	Country	Year	Status	Life expectancy	Adult Mortality	\
0	Afghanistan	2015	Developing	65.0	263.0	
1	Afghanistan	2014	Developing	59.9	271.0	
2	Afghanistan	2013	Developing	59.9	268.0	
3	Afghanistan	2012	Developing	59.5	272.0	
4	Afghanistan	2011	Developing	59.2	275.0	

	infant deaths	Alcohol	percentage expenditure	Hepatitis B	Measles	…	\
0	62	0.01	71.279624	65.0	1154	…	
1	64	0.01	73.523582	62.0	492	…	
2	66	0.01	73.219243	64.0	430	…	
3	69	0.01	78.184215	67.0	2787	…	
4	71	0.01	7.097109	68.0	3013	…	

	Polio	Total expenditure	Diphtheria	HIV/AIDS	GDP	Population	\
0	6.0	8.16	65.0	0.1	584.259210	33736494.0	
1	58.0	8.18	62.0	0.1	612.696514	327582.0	
2	62.0	8.13	64.0	0.1	631.744976	31731688.0	
3	67.0	8.52	67.0	0.1	669.959000	3696958.0	
4	68.0	7.87	68.0	0.1	63.537231	2978599.0	

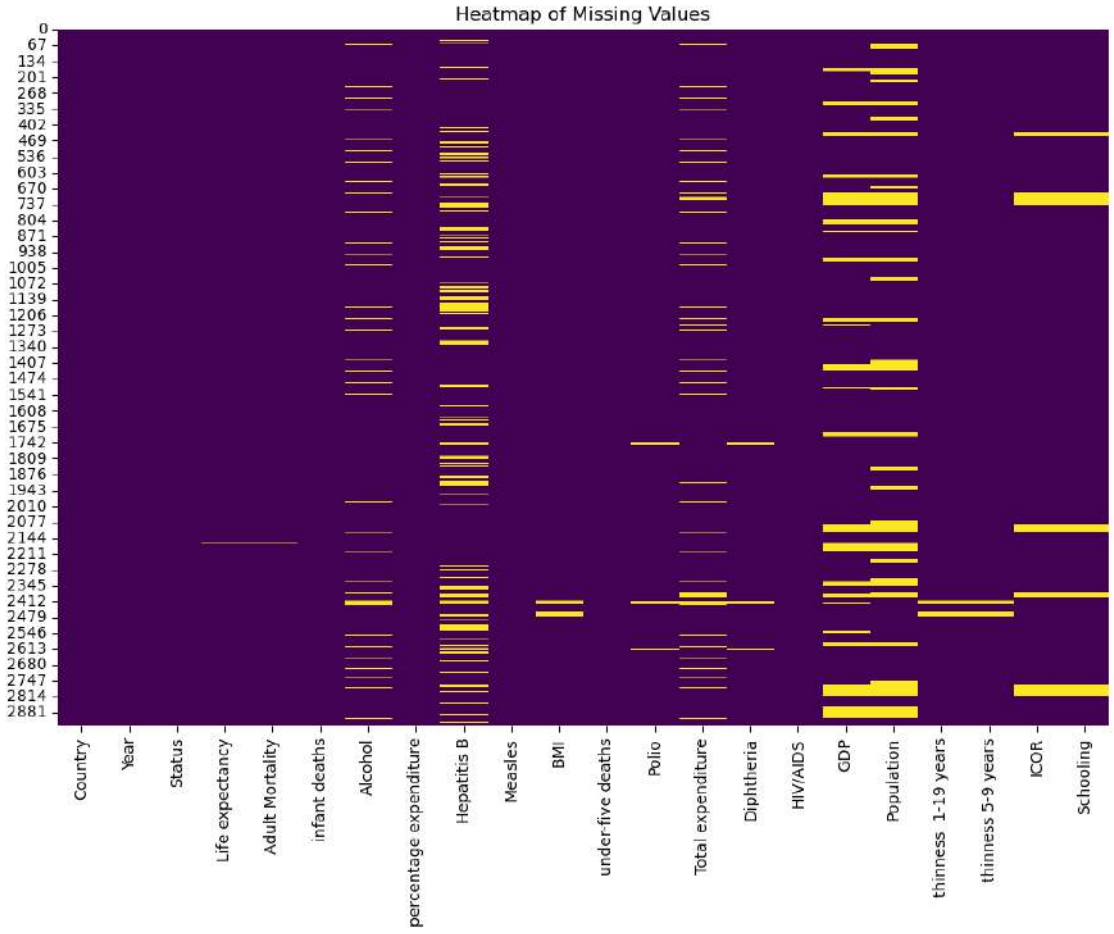
	thinness 1-19 years	thinness 5-9 years	ICOR	Schooling
0	17.2	17.3	0.479	10.1
1	17.5	17.5	0.476	10.0
2	17.7	17.7	0.470	9.9
3	17.9	18.0	0.463	9.8
4	18.2	18.2	0.454	9.5

```
[5 rows x 22 columns]
```

[31]:
```
start_year = who['Year'].min()
end_year = who['Year'].max()
print(f"The dataset starts in {start_year} and ends in {end_year}.")
```

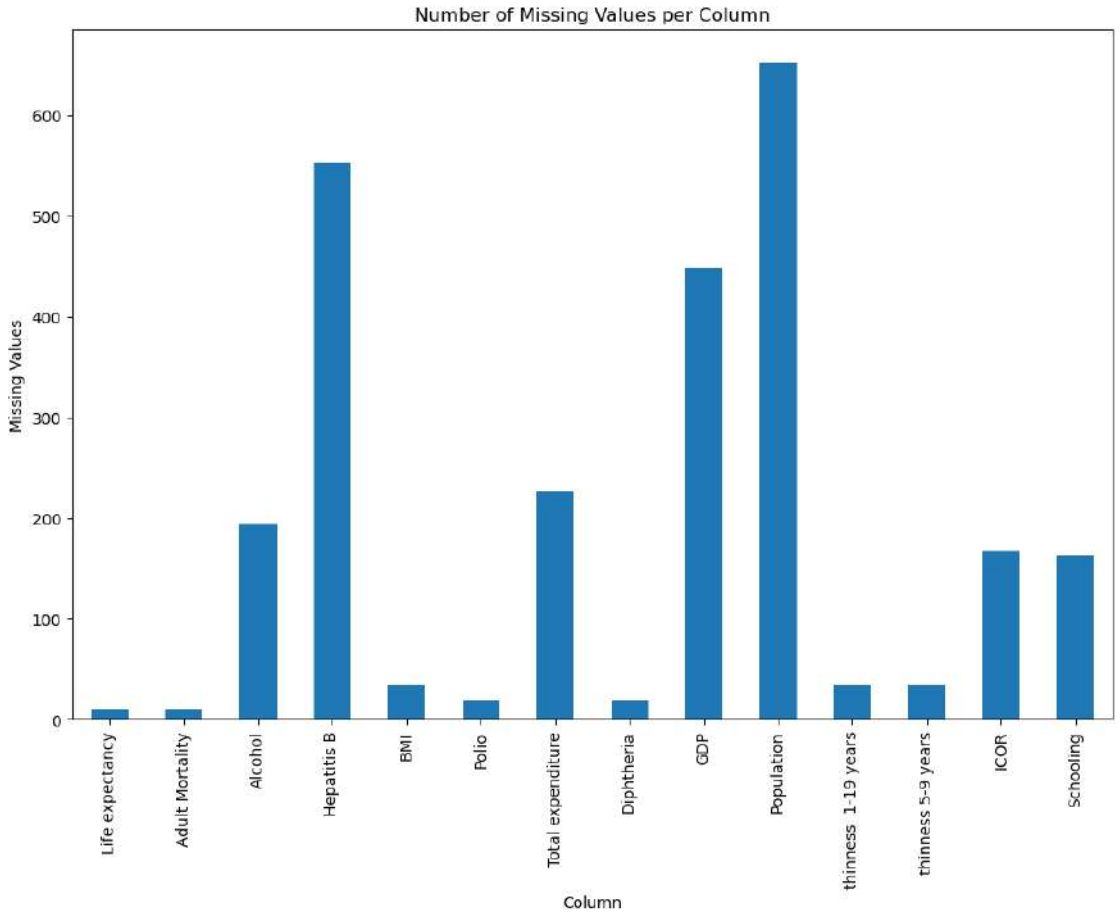
```
The dataset starts in 2000 and ends in 2015.
```

[32]:
```
# Create a heatmap of missing values
plt.figure(figsize=(12, 8))
sns.heatmap(who.isnull(), cbar=False, cmap='viridis')
plt.title('Heatmap of Missing Values')
plt.show()
```



[33]:
```
# Count missing values for each column
missing_counts = who.isnull().sum()
```

```
# Plot missing values
plt.figure(figsize=(12, 8))
missing_counts[missing_counts > 0].plot(kind='bar')
plt.title('Number of Missing Values per Column')
plt.xlabel('Column')
plt.ylabel('Missing Values')
plt.xticks(rotation=90)
plt.show()
```



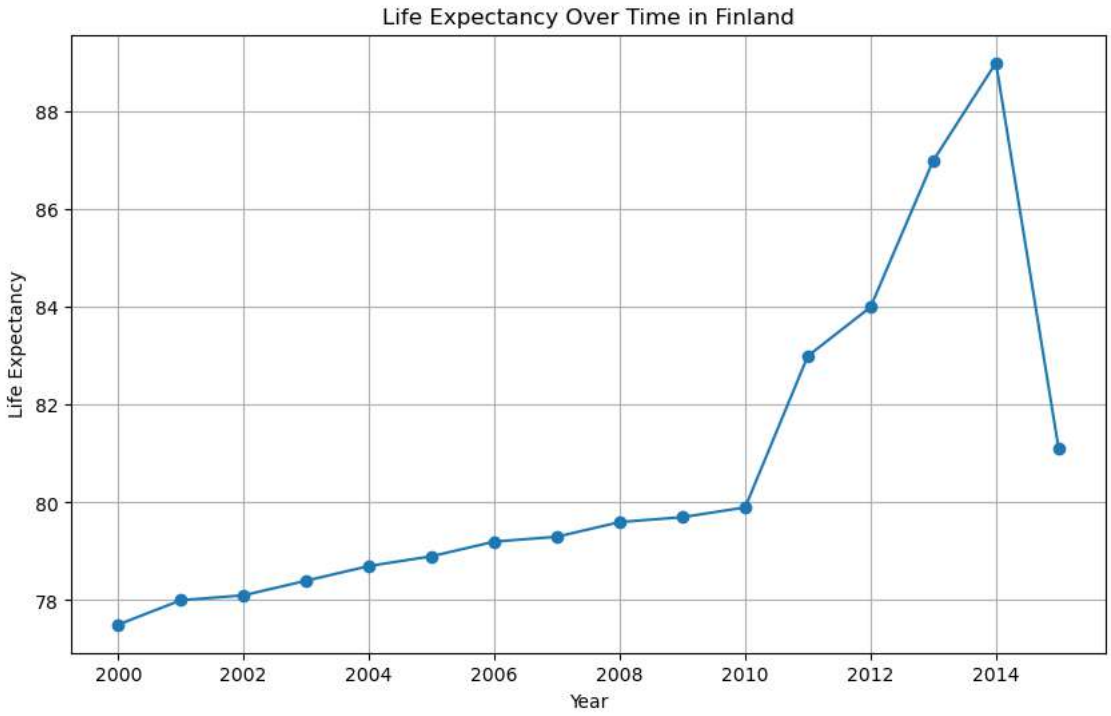
[34]:
```
# Display summary statistics for missing values
print(who.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2938 entries, 0 to 2937
Data columns (total 22 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Country                 2938 non-null   object
 1   Year                    2938 non-null   int64
```

```
 2   Status                   2938 non-null   object
 3   Life expectancy          2928 non-null   float64
 4   Adult Mortality          2928 non-null   float64
 5   infant deaths            2938 non-null   int64
 6   Alcohol                  2744 non-null   float64
 7   percentage expenditure   2938 non-null   float64
 8   Hepatitis B              2385 non-null   float64
 9   Measles                  2938 non-null   int64
 10  BMI                      2904 non-null   float64
 11  under-five deaths        2938 non-null   int64
 12  Polio                    2919 non-null   float64
 13  Total expenditure        2712 non-null   float64
 14  Diphtheria               2919 non-null   float64
 15  HIV/AIDS                 2938 non-null   float64
 16  GDP                      2490 non-null   float64
 17  Population               2286 non-null   float64
 18  thinness  1-19 years     2904 non-null   float64
 19  thinness 5-9 years       2904 non-null   float64
 20  ICOR                     2771 non-null   float64
 21  Schooling                2775 non-null   float64
dtypes: float64(16), int64(4), object(2)
memory usage: 505.1+ KB
None
```

[35]:
```python
def life_exp_overtime(country_name):
    # Plot life expectancy over time for a specific country
    # Filter the data for the specified country
    country_data = who[who['Country'] == country_name]

    # Check if the country exists in the dataset
    if country_data.empty:
        print(f"No data found for {country_name}.")
        return

    # Plot the data
    plt.figure(figsize=(10, 6))
    plt.plot(country_data['Year'], country_data['Life expectancy'], marker='o')
    plt.title(f'Life Expectancy Over Time in {country_name}')
    plt.xlabel('Year')
    plt.ylabel('Life Expectancy')
    plt.grid(True)
    plt.show()
```

[36]:
```python
life_exp_overtime('Finland')
```



[37]:
```
# We want to know how many countries changed status (Developing/Developed) over␣
 ↪the years
# Reminder: we have a total of 193 countries in the dataset

# Group by 'Country' and get unique statuses for each country
status_per_country = who.groupby('Country')['Status'].nunique()

# Filter countries that have only one unique status
countries_with_unchanged_status = status_per_country[status_per_country == 1]

# Display the number of such countries
num_countries_with_unchanged_status = countries_with_unchanged_status.shape[0]
print(f"Number of countries that did not change status over the years:␣
 ↪{num_countries_with_unchanged_status}")
```

```
Number of countries that did not change status over the years: 193
```

[38]:
```
from sklearn.preprocessing import LabelEncoder

# Create a LabelEncoder instance
le = LabelEncoder()

# Fit and transform the country names
who['Country_encoded'] = le.fit_transform(who['Country'])
```

```
# Encode 'Status' column as 0 for Developing and 1 for Developed
who['Status_encoded'] = who['Status'].map({'Developing': 0, 'Developed': 1})
```

[39]:
```
print(who.columns)
```

```
Index(['Country', 'Year', 'Status', 'Life expectancy', 'Adult Mortality',
       'infant deaths', 'Alcohol', 'percentage expenditure', 'Hepatitis B',
       'Measles', 'BMI', 'under-five deaths', 'Polio', 'Total expenditure',
       'Diphtheria', 'HIV/AIDS', 'GDP', 'Population', 'thinness  1-19 years',
       'thinness 5-9 years', 'ICOR', 'Schooling', 'Country_encoded',
       'Status_encoded'],
      dtype='object')
```

[40]:
```
who.head()
```

[40]:

	Country	Year	Status	Life expectancy	Adult Mortality	\
0	Afghanistan	2015	Developing	65.0	263.0	
1	Afghanistan	2014	Developing	59.9	271.0	
2	Afghanistan	2013	Developing	59.9	268.0	
3	Afghanistan	2012	Developing	59.5	272.0	
4	Afghanistan	2011	Developing	59.2	275.0	

	infant deaths	Alcohol	percentage expenditure	Hepatitis B	Measles	…	\
0	62	0.01	71.279624	65.0	1154	…	
1	64	0.01	73.523582	62.0	492	…	
2	66	0.01	73.219243	64.0	430	…	
3	69	0.01	78.184215	67.0	2787	…	
4	71	0.01	7.097109	68.0	3013	…	

	Diphtheria	HIV/AIDS	GDP	Population	thinness 1-19 years	\
0	65.0	0.1	584.259210	33736494.0	17.2	
1	62.0	0.1	612.696514	327582.0	17.5	
2	64.0	0.1	631.744976	31731688.0	17.7	
3	67.0	0.1	669.959000	3696958.0	17.9	
4	68.0	0.1	63.537231	2978599.0	18.2	

	thinness 5-9 years	ICOR	Schooling	Country_encoded	Status_encoded
0	17.3	0.479	10.1	0	0
1	17.5	0.476	10.0	0	0
2	17.7	0.470	9.9	0	0
3	18.0	0.463	9.8	0	0
4	18.2	0.454	9.5	0	0

[5 rows x 24 columns]

[41]:
```
# Comparing developing (0) and developed (1) countries on some variables in the␣
 ↪year 2015
who_2015 = who[who['Year'] == 2015]

# Pivot table for the year 2015
pivot_table_2015 = pd.pivot_table(who_2015,
                                  values=['Life expectancy', 'Adult Mortality',␣
 ↪'Alcohol', 'Schooling', 'ICOR'],
                                  index='Status_encoded',
                                  aggfunc='mean')

print(pivot_table_2015)
```

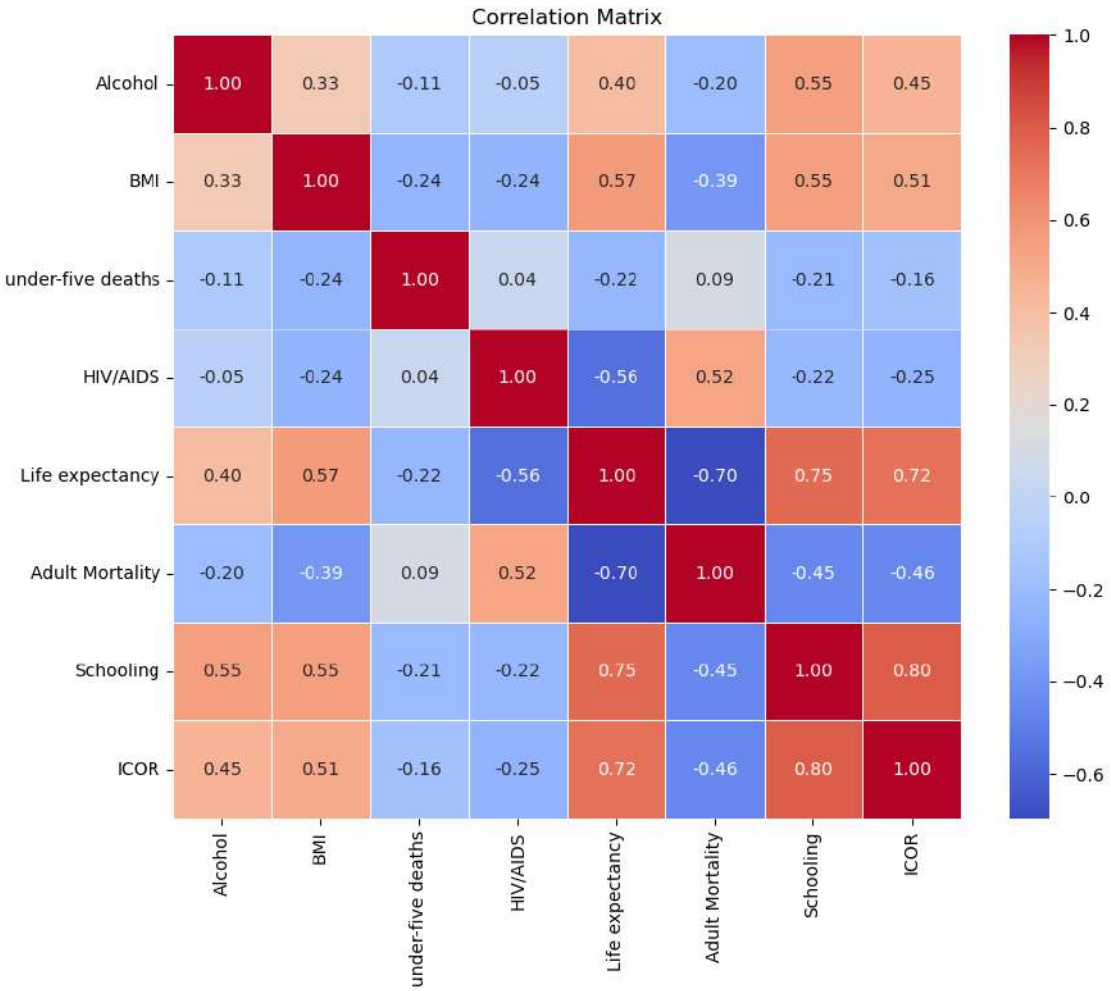
```
                 Adult Mortality  Alcohol      ICOR  Life expectancy  Schooling
Status_encoded
0                     169.390728    2.305  0.653431        69.690066  12.200000
1                      74.875000    6.780  0.881966        80.709375  16.537931
```

[42]:
```
# Compute correlation matrix
correlation_matrix = who[['Alcohol', 'BMI', 'under-five deaths', 'HIV/AIDS',␣
 ↪'Life expectancy', 'Adult Mortality', 'Schooling', 'ICOR']].corr()


print(correlation_matrix)
```

```
                     Alcohol       BMI  under-five deaths  HIV/AIDS  \
Alcohol             1.000000  0.330408          -0.112370 -0.048845
BMI                 0.330408  1.000000          -0.237669 -0.243717
under-five deaths  -0.112370 -0.237669           1.000000  0.038062
HIV/AIDS           -0.048845 -0.243717           0.038062  1.000000
Life expectancy     0.404877  0.567694          -0.222529 -0.556556
Adult Mortality    -0.195848 -0.387017           0.094146  0.523821
Schooling           0.547378  0.546961          -0.209373 -0.220429
ICOR                0.450040  0.508774          -0.163305 -0.249519

                   Life expectancy  Adult Mortality  Schooling      ICOR
Alcohol                   0.404877        -0.195848   0.547378  0.450040
BMI                       0.567694        -0.387017   0.546961  0.508774
under-five deaths        -0.222529         0.094146  -0.209373 -0.163305
HIV/AIDS                 -0.556556         0.523821  -0.220429 -0.249519
Life expectancy           1.000000        -0.696359   0.751975  0.724776
Adult Mortality          -0.696359         1.000000  -0.454612 -0.457626
Schooling                 0.751975        -0.454612   1.000000  0.800092
ICOR                      0.724776        -0.457626   0.800092  1.000000
```

[43]:
```
# Plot heatmap
plt.figure(figsize=(10, 8))
```

```
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f',␣
 ↪linewidths=0.5)
plt.title('Correlation Matrix')
plt.show()
```



The cell below runs a Random Forest to try to figure out which features are the most important in predicting life expectancy. We're doing this to see how this compares with the correlation matrix. The correlation matrix only shows linear relations, whereas the Random Forest can detect also non-linear relations. Since we have 15 years of data for every country, we are using the median values for each country over those 15 years.

[44]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# Drop rows with missing values to ensure a clean dataset
```

```
who_cleaned = who.dropna()

# Aggregate data by country using median values
aggregated_data = who_cleaned.groupby('Country_encoded').agg({
    'Life expectancy': 'median',
    'Alcohol': 'median',
    'Adult Mortality': 'median',
    'Schooling': 'median',
    'ICOR': 'median',
    'HIV/AIDS': 'median',
    'BMI': 'median',
    'GDP': 'median',
    'percentage expenditure': 'median',
    'under-five deaths': 'median',
    'Polio': 'median',
    'Population': 'median',
    'Measles': 'median',
    'infant deaths': 'median',
    'Diphtheria': 'median',
    'Hepatitis B': 'median',
}).reset_index()

# Separate features and target
X = aggregated_data.drop(columns=['Life expectancy', 'Country_encoded'])
y = aggregated_data['Life expectancy']


X = X.apply(pd.to_numeric, errors='coerce')


X = X.dropna()
y = y[X.index]

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Initialize and train Random Forest
model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Predict
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")

# Feature importance
```

```
importances = model.feature_importances_
feature_importance_df = pd.DataFrame({
    'Feature': X.columns,
    'Importance': importances
}).sort_values(by='Importance', ascending=False)

print(feature_importance_df)
```

```
Mean Squared Error: 6.3449132592592505
                   Feature  Importance
1          Adult Mortality    0.505150
3                     ICOR    0.270556
4                  HIV/AIDS    0.094272
6                      GDP    0.071533
7   percentage expenditure    0.017500
5                      BMI    0.006775
2                Schooling    0.006135
11                 Measles    0.004885
8        under-five deaths    0.004707
0                  Alcohol    0.004552
12           infant deaths    0.004043
10              Population    0.002817
13              Diphtheria    0.002595
9                    Polio    0.002330
14             Hepatitis B    0.002151
```

Adult mortality is unsurprisingly the biggest factor, but ICOR comes in second at 0.27. That's 3 times the importance of the third most important factor (HIV/AIDS). So this is an interesting finding.

[45]:
```
who.head()
```

[45]:

	Country	Year	Status	Life expectancy	Adult Mortality	\
0	Afghanistan	2015	Developing	65.0	263.0	
1	Afghanistan	2014	Developing	59.9	271.0	
2	Afghanistan	2013	Developing	59.9	268.0	
3	Afghanistan	2012	Developing	59.5	272.0	
4	Afghanistan	2011	Developing	59.2	275.0	

	infant deaths	Alcohol	percentage expenditure	Hepatitis B	Measles	…	\
0	62	0.01	71.279624	65.0	1154	…	
1	64	0.01	73.523582	62.0	492	…	
2	66	0.01	73.219243	64.0	430	…	
3	69	0.01	78.184215	67.0	2787	…	
4	71	0.01	7.097109	68.0	3013	…	

	Diphtheria	HIV/AIDS	GDP	Population	thinness	1-19 years	\
0	65.0	0.1	584.259210	33736494.0		17.2	

```
1        62.0       0.1  612.696514    327582.0                  17.5
2        64.0       0.1  631.744976  31731688.0                  17.7
3        67.0       0.1  669.959000   3696958.0                  17.9
4        68.0       0.1   63.537231   2978599.0                  18.2

   thinness 5-9 years   ICOR  Schooling  Country_encoded  Status_encoded
0                17.3  0.479       10.1                0               0
1                17.5  0.476       10.0                0               0
2                17.7  0.470        9.9                0               0
3                18.0  0.463        9.8                0               0
4                18.2  0.454        9.5                0               0

[5 rows x 24 columns]
```

[46]:
```
print(aggregated_data.columns)
```

```
Index(['Country_encoded', 'Life expectancy', 'Alcohol', 'Adult Mortality',
       'Schooling', 'ICOR', 'HIV/AIDS', 'BMI', 'GDP', 'percentage expenditure',
       'under-five deaths', 'Polio', 'Population', 'Measles', 'infant deaths',
       'Diphtheria', 'Hepatitis B'],
      dtype='object')
```

[47]:
```
aggregated_data.head()
```

[47]:
```
   Country_encoded  Life expectancy  Alcohol  Adult Mortality  Schooling  \
0                0             57.8     0.01            284.0       8.55
1                1             75.6     4.95             17.5      11.80
2                2             74.4     0.46            119.0      13.10
3                3             50.3     7.93            363.0       9.20
4                5             75.4     8.11            126.0      16.40

     ICOR  HIV/AIDS    BMI          GDP  percentage expenditure  \
0  0.4240       0.1  15.45   321.199783               17.029434
1  0.7080       0.1  52.15  1390.308084              100.361243
2  0.7050       0.1  52.80  3868.831230              320.323924
3  0.4915       2.5  20.40  3438.689835              202.288337
4  0.7920       0.1  58.00  7193.617640              961.177468

   under-five deaths  Polio  Population  Measles  infant deaths  Diphtheria  \
0              111.5   58.0   2924815.0   1794.0           81.0        62.5
1                1.0   98.0    290456.5      8.5            1.0        98.0
2               23.0   95.0  33777915.0    107.0           20.0        95.0
3              118.0   70.5   2845076.5   2128.0           76.5        72.0
4               11.0   94.0   4382389.0      0.0           10.0        94.0

   Hepatitis B
0         64.0
```

```
1          98.0
2          94.0
3          72.5
4          88.0
```

Let's now run the linear regression on this aggregated_data dataframe (which includes only 1 row per country since we aggregated the 15 years using median values). First without ridge.

[48]:
```
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split

aggregated_data['GDP per capita'] = aggregated_data['GDP'] /␣
 ↪aggregated_data['Population']
features = ['Alcohol', 'Schooling', 'HIV/AIDS', 'ICOR', 'under-five deaths',␣
 ↪'Adult Mortality', 'GDP per capita']
X = aggregated_data[features]  # Predictor variables

# Target variable
y = aggregated_data['Life expectancy']

#  Standardize the feature matrix
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,␣
 ↪random_state=42)

#  Fit the linear regression model
lin_reg = LinearRegression()
lin_reg.fit(X_train, y_train)

#  Make predictions
y_pred = lin_reg.predict(X_test)

#  Evaluate the model performance
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)


print(f"Linear Regression MSE: {mse}")
print(f"Linear Regression R-squared: {r2}")
```

```
Linear Regression MSE: 5.637236095364524
Linear Regression R-squared: 0.9217976446397866
```

Now we're going to do linear regression with ridge, to see how it compares.

[49]:
```
# 1. features and label
aggregated_data['GDP per capita'] = aggregated_data['GDP'] /␣
 ↪aggregated_data['Population']
features = ['Alcohol', 'Schooling', 'HIV/AIDS', 'ICOR', 'under-five deaths',␣
 ↪'Adult Mortality', 'GDP per capita']
X = aggregated_data[features]  # Predictor variables
y = aggregated_data['Life expectancy']

# 2. Standardize the feature matrix
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 3. Train-Test Split (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,␣
 ↪random_state=42)

# 4. Create and fit the Ridge regression model
alpha = 1  # Regularization strength
ridge_reg = Ridge(alpha=alpha)
ridge_reg.fit(X_train, y_train)

# 5. Make predictions
y_pred_ridge = ridge_reg.predict(X_test)

# 6. Evaluate the Ridge regression performance
mse_ridge = mean_squared_error(y_test, y_pred_ridge)
r2_ridge = r2_score(y_test, y_pred_ridge)


print(f"Ridge Regression MSE: {mse_ridge}")
print(f"Ridge Regression R-squared: {r2_ridge}")
```

```
Ridge Regression MSE: 5.739135139146359
Ridge Regression R-squared: 0.9203840538130257
```

Cross validation k-fold

[50]:
```
from sklearn.model_selection import cross_val_score

# Perform k-fold cross-validation
k = 7  # Set the number of folds
ridge_scores = cross_val_score(ridge_reg, X_scaled, y, cv=k,␣
 ↪scoring='neg_mean_squared_error')

# Average MSE across all folds (negate back to positive)
avg_mse_ridge = -ridge_scores.mean()
```

```
print(f"Ridge Regression Avg MSE (5-fold): {avg_mse_ridge}")
```

```
Ridge Regression Avg MSE (5-fold): 7.795198787911717
```

Now let's run the polynomial regression

[51]:
```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline


aggregated_data['GDP per capita'] = aggregated_data['GDP'] /␣
 ↪aggregated_data['Population']
features = ['Alcohol', 'Schooling', 'HIV/AIDS', 'ICOR', 'under-five deaths',␣
 ↪'Adult Mortality', 'GDP per capita']
X = aggregated_data[features]  # Predictor variables
y = aggregated_data['Life expectancy']  # Target variable

# polynomial degree
degree = 2
poly_features = PolynomialFeatures(degree=degree)

# ridge regularization strength
alpha = 1.0

# pipeline that first applies polynomial transformation, scales the data and␣
 ↪then fits ridge reg
pipeline = make_pipeline(PolynomialFeatures(degree), StandardScaler(),␣
 ↪Ridge(alpha=alpha))

# k-fold cross-validation
k = 7  # Number of folds
cv_scores = cross_val_score(pipeline, X, y, cv=k,␣
 ↪scoring='neg_mean_squared_error')

# calculate the average MSE across all folds (convert the negative MSE back to␣
 ↪positive)
avg_mse = -cv_scores.mean()
print(f"Polynomial Regression with Ridge (degree={degree}, alpha={alpha}) Avg␣
 ↪MSE (k={k}): {avg_mse}")


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# fit the model on the training data
pipeline.fit(X_train, y_train)
```

```
# predictions on the test data
y_pred = pipeline.predict(X_test)

# evaluate performance on the test set
mse_test = mean_squared_error(y_test, y_pred)
r2_test = r2_score(y_test, y_pred)

print(f"Polynomial Ridge Regression Test MSE: {mse_test}")
print(f"Polynomial Ridge Regression Test R-squared: {r2_test}")
```

```
Polynomial Regression with Ridge (degree=2, alpha=1.0) Avg MSE (k=7):
12786.56805189435
Polynomial Ridge Regression Test MSE: 4.538089663731771
Polynomial Ridge Regression Test R-squared: 0.9370455140540438
```

[52]:
```
aggregated_data['GDP per capita'] = aggregated_data['GDP'] /␣
 ↪aggregated_data['Population']
features = ['Alcohol', 'Schooling', 'HIV/AIDS', 'ICOR', 'under-five deaths',␣
 ↪'Adult Mortality', 'GDP per capita']
X = aggregated_data[features]  # Predictor variables
y = aggregated_data['Life expectancy']  # Target variable


degree = 1  # You can choose the degree you want (e.g., 2 or 3)
poly_features = PolynomialFeatures(degree=degree)


alpha = 1.0  # Regularization strength


pipeline = make_pipeline(PolynomialFeatures(degree), StandardScaler(),␣
 ↪Ridge(alpha=alpha))

# k-fold cross-validation
k = 7  # Number of folds
cv_scores = cross_val_score(pipeline, X, y, cv=k,␣
 ↪scoring='neg_mean_squared_error')

# calculate the average MSE across all folds (convert the negative MSE back to␣
 ↪positive)
avg_mse = -cv_scores.mean()
print(f"Polynomial Regression with Ridge (degree={degree}, alpha={alpha}) Avg␣
 ↪MSE (k={k}): {avg_mse}")


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

```
# fit the model on the training data
pipeline.fit(X_train, y_train)

# make predictions on the test data
y_pred = pipeline.predict(X_test)

# evaluate performance on the test set
mse_test = mean_squared_error(y_test, y_pred)
r2_test = r2_score(y_test, y_pred)

print(f"Polynomial Ridge Regression Test MSE: {mse_test}")
print(f"Polynomial Ridge Regression Test R-squared: {r2_test}")
```

```
Polynomial Regression with Ridge (degree=1, alpha=1.0) Avg MSE (k=7):
7.822919889716687
Polynomial Ridge Regression Test MSE: 5.7384727154833755
Polynomial Ridge Regression Test R-squared: 0.92039324326151
```

Let's look at the feature importance by running a random forest

[53]:
```
X = aggregated_data.drop(columns=['Life expectancy', 'Country_encoded', 'GDP'])
y = aggregated_data['Life expectancy']


X = X.apply(pd.to_numeric, errors='coerce')


X = X.dropna()
y = y[X.index]


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# random Forest
model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")

# feature importance
importances = model.feature_importances_
feature_importance_df = pd.DataFrame({
    'Feature': X.columns,
    'Importance': importances
```

```
}).sort_values(by='Importance', ascending=False)

print(feature_importance_df)
```

```
Mean Squared Error: 5.646193814814858
                   Feature  Importance
1          Adult Mortality    0.529644
3                     ICOR    0.306252
4                 HIV/AIDS    0.093281
6   percentage expenditure    0.021688
2                Schooling    0.008895
5                      BMI    0.006908
10                 Measles    0.005651
0                  Alcohol    0.005303
7        under-five deaths    0.004208
14          GDP per capita    0.003742
11           infant deaths    0.003554
12              Diphtheria    0.003349
8                    Polio    0.002890
13             Hepatitis B    0.002643
9               Population    0.001993
```

Separate test and validations sets (60/20/20 split):

[54]:
```
from sklearn.model_selection import cross_val_score


X = aggregated_data.drop(columns=['Life expectancy', 'Country_encoded', 'GDP'])␣
  ↪ # Drop GDP, keep GDP per capita
y = aggregated_data['Life expectancy']


X = X.apply(pd.to_numeric, errors='coerce')


X = X.dropna()
y = y[X.index]


scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# ridge for linear model
ridge_reg = Ridge(alpha=1)

# k-fold cross-validation for linear ridge
k = 5  # Number of folds
```

```
ridge_scores = cross_val_score(ridge_reg, X_scaled, y, cv=k,␣
 ↪scoring='neg_mean_squared_error')
avg_mse_ridge = -ridge_scores.mean()  # Negate to get positive MSE
print(f"Ridge Regression Avg MSE (k={k}): {avg_mse_ridge}")

# poly ridge reg with degree=2
poly = PolynomialFeatures(degree=2)
X_poly = poly.fit_transform(X_scaled)

# ridge for poly
poly_ridge_reg = Ridge(alpha=1.0)

# k-fold cross-validation for poly ridge
poly_ridge_scores = cross_val_score(poly_ridge_reg, X_poly, y, cv=k,␣
 ↪scoring='neg_mean_squared_error')
avg_mse_poly_ridge = -poly_ridge_scores.mean()  # Negate to get positive MSE
print(f"Polynomial Ridge Regression (degree=2) Avg MSE (k={k}):␣
 ↪{avg_mse_poly_ridge}")

# final evaluation with test set after cross-validation
X_train_val, X_test, y_train_val, y_test = train_test_split(X, y, test_size=0.
 ↪2, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_train_val, y_train_val,␣
 ↪test_size=0.25, random_state=42)  # 0.25 * 0.8 = 0.2 overall

# refit Ridge (degree=1) on the train/validation split
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)
X_test_scaled = scaler.transform(X_test)

ridge_reg.fit(X_train_scaled, y_train)

# Predict on validation and test set (ridge regression)
y_val_pred = ridge_reg.predict(X_val_scaled)
mse_val_ridge = mean_squared_error(y_val, y_val_pred)
print(f"Ridge Regression Validation MSE: {mse_val_ridge}")

y_test_pred = ridge_reg.predict(X_test_scaled)
mse_test_ridge = mean_squared_error(y_test, y_test_pred)
r2_test_ridge = r2_score(y_test, y_test_pred)
print(f"Ridge Regression Test MSE: {mse_test_ridge}")
print(f"Ridge Regression Test R-squared: {r2_test_ridge}")

# refit polyy ridge reg on the train/validation split
X_train_poly = poly.fit_transform(X_train_scaled)
X_val_poly = poly.transform(X_val_scaled)
X_test_poly = poly.transform(X_test_scaled)
```

```
poly_ridge_reg.fit(X_train_poly, y_train)

# predict on validation and test set (poly ridge reg)
y_val_pred_poly = poly_ridge_reg.predict(X_val_poly)
mse_val_poly = mean_squared_error(y_val, y_val_pred_poly)
print(f"Polynomial Ridge Regression Validation MSE: {mse_val_poly}")

y_test_pred_poly = poly_ridge_reg.predict(X_test_poly)
mse_test_poly = mean_squared_error(y_test, y_test_pred_poly)
r2_test_poly = r2_score(y_test, y_test_pred_poly)
print(f"Polynomial Ridge Regression Test MSE: {mse_test_poly}")
print(f"Polynomial Ridge Regression Test R-squared: {r2_test_poly}")
```

```
Ridge Regression Avg MSE (k=5): 6.284689072645273
Polynomial Ridge Regression (degree=2) Avg MSE (k=5): 77.54516635509388
Ridge Regression Validation MSE: 4.556204431078149
Ridge Regression Test MSE: 10.414653186126133
Ridge Regression Test R-squared: 0.855523097554042
Polynomial Ridge Regression Validation MSE: 208.15754701805494
Polynomial Ridge Regression Test MSE: 5.930109869029756
Polynomial Ridge Regression Test R-squared: 0.9177347637285741
```

[ ]:

[ ]: